THÈSE DE DOCTORAT de l'UNIVERSITÉ LILLE 1

Spécialité

Informatique

Préparée au sein de l'**École Doctorale Science Pour l'Ingénieur** et du **Laboratoire d'Informatique Fondamentale de Lille**

Présentée par

Olivier NICOL

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ LILLE 1

Data-driven evaluation of Contextual Bandit algorithms and applications to Dynamic Recommendation

Soutenue publiquement le 18 décembre 2014 devant un jury composé de

M. **Philippe PREUX**	Université Lille 3	Directeur de thèse
M. **Jérémie MARY**	Université Lille 3	Co-directeur de thèse
M. **Olivier CAPPÉ**	CNRS	Rapporteur
M. **Ludovic DENOYER**	Université Pierre et Marie Curie	Rapporteur
M. **Rémi Gilleron**	Université Lille 3	Examinateur
M. **Olivier CHAPELLE**	Criteo	Examinateur
M. **Lihong LI**	Microsoft Research	Examinateur

Table of Contents

Acknowledgments/Remerciements

I want to express my sincere gratitude to the person who guided me with boundless patience throughout this long and difficult endeavor: Philippe Preux, my advisor. For over four years, he has been continuously supportive and he has enlightened me with his vast knowledge, his enthusiasm, his motivation and his wise advice. I also want to give a special thank to Jérémie Mary. Not only for his advice and guidance - which were precious - but also for his perspicacity, his ideas and his ability to make the coarsest of mine blossom into useful pieces of research, his impressive mathematical skills, his joyous mood, the talk rehearsals, the organization of the 2012 challenge and much more.

I would also like to extend my deepest appreciation to Olivier Cappé and Ludovic Denoyer - the reviewers of my work - for reading my thesis with the greatest care, for their insightful remarks and their challenging questions. My appreciation is extended as well to the other members of my thesis jury for the great interest they took in my work and their helpful comments: Rémi Gilleron, Olivier Chapelle and more particularly Lihong Li, who did much more than read my thesis by taking the time to annotate the entire document with various ways to improve both the style and the exposed research.

Working within the SequeL research group was motivating, inspiring and fun at the same time. For these reasons, I want to thank all its past and current members that I had the chance to work alongside with. I also thank the LIFL, INRIA and the University of Lille 1 for the logistic support they provided as well as the Universities of Lille 1 and Lille 3 for giving me the exciting opportunity to teach.

Finally I want to thank warmly my friends and family and in particular Sophia's family for putting up with me. Last but not least, very special thanks to Manuelle for her unstinting moral support and to Sophia for her great help, her encouragements and simply being with me even in the stress of deadlines and without whom the greatest accomplishments would make so little sense.

Keywords/Mots-clés

Mots-Clés

Mots-clés généraux

Algorithmique, Statistiques, Probabilités, Apprentissage automatique, Fouille de données, Apprentissage séquentiel/en ligne, Prise de décision dans l'incertain, Évaluation de recherche.

Mots-clés spécifiques

Jeux de bandits (contextuels), Évaluation hors ligne, Évaluation basée sur les données, Environement non stationnaire, Recommandation, Recommandation d'articles de journaux, Recommandation dynamique, Analyse de biais/variance/concentration, Bootstrap (méthode statistique d'estimation de propriétés d'un estimateur), Validation croisée, Inférence Bayésienne, Classification, Biais contre variance, Dilemme entre exploration et exploitation, Validation entremêlée, Expansion de données, Méthodes de rejeu.

Keywords

General keywords

Algorithmics, Statistics, Probabilities, Machine Learning, Data mining/Knowledge discovery, Sequential/online learning, Decision making under uncertainty, Research evaluation.

Specific keywords

(Contextual) bandit games, Offline evaluation, Data-driven evaluation, Non stationary environment, Recommendation, News recommendation, Dynamic recommendation, Bias/variance/-concentration analysis, Bootstrapping (statistical method of estimation of estimator properties), Cross-validation, Bayesian inference, Classification, Bias *versus* Variance trade-off, Exploration *versus* Exploitation dilemma, Entangled validation, Data expansion, Replay methodologies.

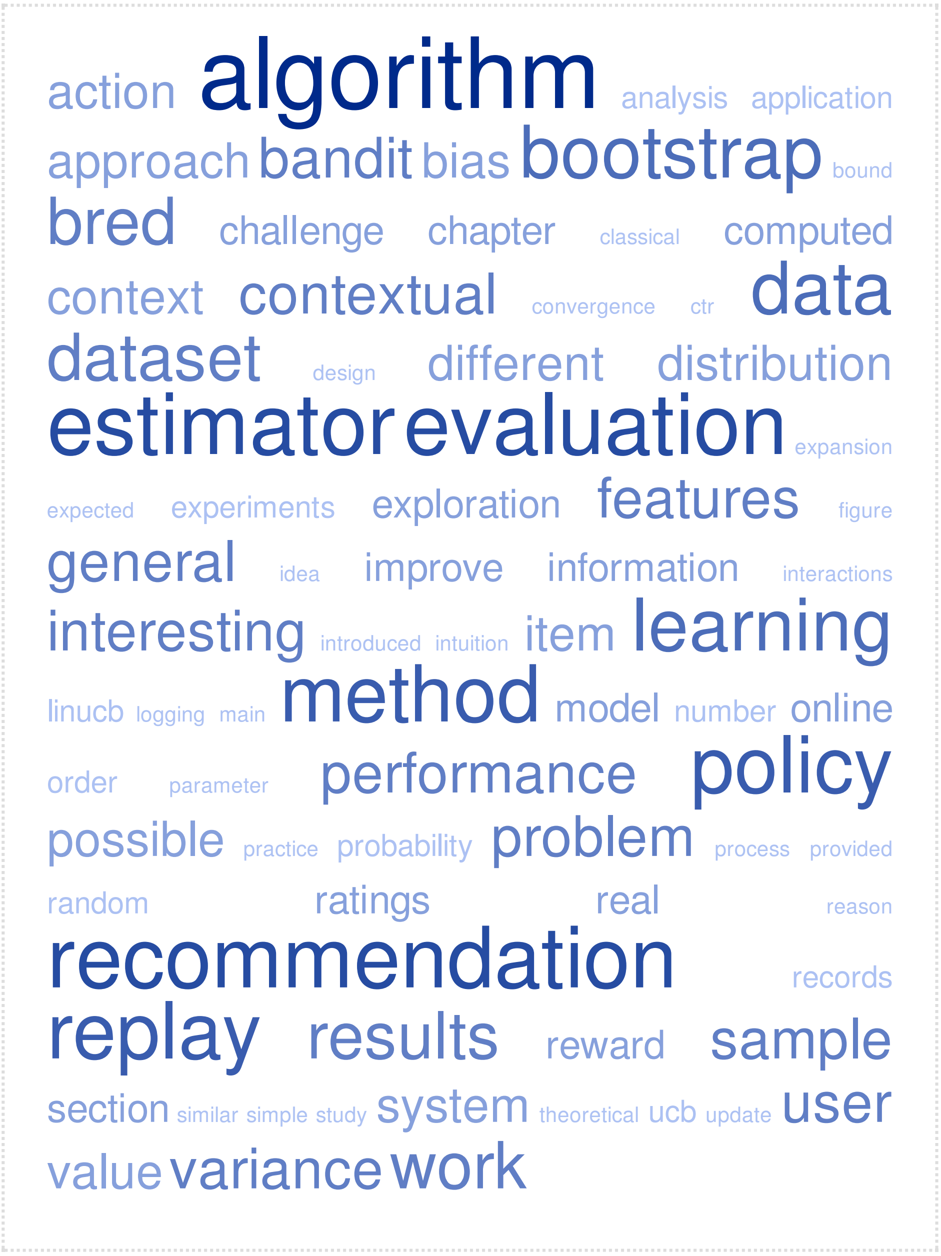

action algorithm analysis application
approach bandit bias bootstrap bound
bred challenge chapter classical computed
context contextual convergence ctr data
dataset design different distribution
estimator evaluation expansion
expected experiments exploration features figure
general idea improve information interactions
interesting introduced intuition item learning
linucb logging main method model number online
order parameter performance policy
possible practice probability problem process provided
random ratings real reason
recommendation records
replay results reward sample
section similar simple study system theoretical ucb update user
value variance work

Figure 1: A tag cloud of the 80 most popular words inside this document. Note that the most popular English words (it, the, is *etc.*) as well as common structure words in scientific/argumentive text (indeed, consider, based *etc.*) have been removed.

Short Summary/Résumé court

Short summary in english/Résumé court en anglais

The context of this thesis work is dynamic recommendation. Recommendation is the action, for an intelligent system, to supply a user of an application with personalized content so as to enhance what is refered to as "user experience" *e.g.* recommending a product on a merchant website or even an article on a blog. Recommendation is considered dynamic when the content to recommend or user tastes evolve rapidly *e.g.* news recommendation. Many applications that are of interest to us generates a tremendous amount of data through the millions of online users they have. Nevertheless, using this data to evaluate a new recommendation technique or even compare two dynamic recommendation algorithms is far from trivial. This is the problem we consider here. Some approaches have already been proposed. Nonetheless they were not studied very thoroughly both from a theoretical point of view (unquantified bias, loose convergence bounds...) and from an empirical one (experiments on private data only). In this work we start by filling many blanks within the theoretical analysis. Then we comment on the result of an experiment of unprecedented scale in this area: a public challenge we organized. This challenge along with a some complementary experiments revealed a unexpected source of a huge bias: time acceleration. The rest of this work tackles this issue. We show that a bootstrap-based approach allows to significantly reduce this bias and more importantly to control it.

Short summary in french/Résumé court en français

Ce travail de thèse a été réalisé dans le contexte de la recommandation dynamique. La recommandation est l'action de fournir du contenu personnalisé à un utilisateur utilisant une application, dans le but d'améliorer son utilisation *e.g.* la recommandation d'un produit sur un site marchant ou d'un article sur un blog. La recommandation est considérée comme dynamique lorsque le contenu à recommander ou encore les goûts des utilisateurs évoluent rapidement *e.g.* la recommandation d'actualités. Beaucoup d'applications auxquelles nous nous intéressons génèrent d'énormes quantités de données grâce à leurs millions d'utilisateurs sur Internet. Néanmoins, l'utilisation de ces données pour évaluer une nouvelle technique de recommandation ou encore comparer deux algorithmes de recommandation est loin d'être triviale. C'est cette problématique que nous considérons ici. Certaines approches ont déjà été proposées. Néanmoins elles sont très peu étudiées autant théoriquement (biais non quantifié, borne de convergence assez large...) qu'empiriquement (expériences sur données privées). Dans ce travail nous commençons par combler de nombreuses lacunes de l'analyse théorique. Ensuite nous discutons les résultats très surprenants d'une expérience à très grande échelle : une compétition ouverte au public que nous avons organisée. Cette compétition nous a permis de mettre en évidence une source de biais considérable et constamment présente en pratique : l'accélération temporelle. La suite de ce travail s'attaque à ce problème. Nous montrons qu'une approche à base de bootstrap permet de réduire mais surtout de contrôler ce biais.

Substantial summary in french/Résumé substantiel en français

Évaluation basée sur des données d'algorithmes de bandits contextuels et application à la recommandation dynamique

Introduction

Ce travail de thèse entre dans le cadre très large de l'apprentissage automatique et de la prise de décision dans un environnement incertain. En particulier nous considérons un agent logiciel (aussi appelé un système) intelligent capable de prendre des décisions dans différents contextes et d'apprendre des conséquences de celles ci, c'est à dire d'adapter son comportement de façon à atteindre un objectif. Un tel fonctionnement est en général appelé apprentissage en ligne, par opposition à l'apprentissage hors ligne qui implique que toutes les décisions soient préparées à l'avance par un apprentissage statistique sur une base de données quelconque. Dans le formalisme particulier que nous considérons, appelé l'apprentissage par renforcement [1], la conséquence d'une décision est une récompense à valeur réelle et l'objectif que nous visons, bien que d'autres soient envisageables, est de maximiser la valeur moyenne de cette récompense.

Nous appliquons le formalisme précité à une application majeure d'internet: les *systèmes de recommandation*. La recommandation est une forme de filtrage de l'information personnalisée et dont le but est de présenter des éléments d'information (article de journal, livre, film, musique...) susceptibles d'intéresser l'utilisateur. En théorie, dans l'apprentissage par renforcement une décision peut altérer l'environnement et donc changer les conséquences des futures décisions de l'agent. Ici il serait ridicule de considérer que recommander un livre à un utilisateur altère l'opinion de toute la communauté. C'est pourquoi nous considérons un sous formalisme de l'apprentissage par renforcement et de l'apprentissage en ligne: les *bandits contextuels* [2], dans lequel une décision n'a pas de conséquence sur l'environnement. Notons néanmoins que cela ne signifie pas que nous supposions que l'environnement soit stationnaire, bien au contraire. C'est même l'une des originalités de ce travail.

Dans le contexte que nous venons de définir, ce travail de thèse s'attaque à l'évaluation d'algorithmes de bandits contextuels (*i.e.* de systèmes de recommandation) basée sur des données d'applications réelles. Ce problème est crucial car il est impensable de mettre en ligne un nouvel algorithme sans avoir quelques garanties quant à ses performances auparavant. En effet, cela pourrait diminuer la qualité de ce que l'on appelle "l'expérience utilisateur", ce qui aurait pour effet de diminuer les revenus de l'entreprise à court terme (moins d'achats le jour du test) ou même à moyen terme (non retour d'un utilisateur déçu).

Chapitre 1

Ce premier chapitre met en place les notations nécessaires à la présentation de ce travail de thèse et en décrit le contexte. Pour ce faire nous décrivons brièvement les approches classiques pour construire un système de recommandation. Nous parlons également des principales approches au problème de bandits contextuels mentionné en introduction. Ce problème itératif a pour principe de choisir une action (un article à recommander), étant donné un contexte (un utilisateur) dans le but de maximiser une récompense (par exemple le nombre moyen de clics des utilisateurs sur les articles proposés). L'essence d'un algorithme efficace est de choisir à chaque étape entre faire l'action ayant la plus grande espérance de gain estimée dans le contexte proposé (exploitation) et faire une autre action dans le but d'acquérir plus de connaissance sur les caractéristiques du problème de façon à être plus efficace par la suite (exploration).

Néanmoins, ce chapitre contient aussi une contribution très importante. Il s'agit de la motivation de l'utilisation de ce formalisme dans le contexte de la recommandation dynamique, chose absente de la littérature jusqu'à présent.

Sans entrer dans les détails, les méthodologies de recommandation se basent sur l'idée que ce qui était vrai hier le sera aussi aujourd'hui et demain. C'est pourquoi elles construisent des modèles statistiques (principalement par *Collaborative filtering*) à partir de la foule de données de recommandations passées pour prédire les goûts des utilisateurs. Ces modèles sont ensuite utilisés pour faire des recommandations aux utilisateurs du site web. C'est donc de l'apprentissage hors ligne. L'immense majorité de la recherche en recommandation se concentre sur ces modèles [3]. Néanmoins, lorsque le contenu est dynamique, c'est à dire qu'il évolue très rapidement, ces modèles hors ligne montrent leurs limites: comment prédire si un utilisateur aimera un article qui n'a jamais été recommandé par le passé ? Les actualités sont un exemple très caractéristique puisque qu'un article de journal ne "vit" qu'une fois et ce en général que pour quelques heures. En utilisant les bandits contextuels, l'objectif est d'apprendre en ligne les caractéristiques des nouveaux articles rapidement de façon à les recommander judicieusement et donc sans utiliser de données passées. Néanmoins des méthodes hors ligne (*Content based filtering* [4]) existent aussi pour ce type de recommandation à contenu dynamique. Ces techniques ne sont pas comparées dans la littérature alors qu'elles ont chacune leurs forces et faiblesses. La première contribution de ce travail de thèse est de palier à ce manque. Entre autres choses, nous avançons les arguments que l'apprentissage en ligne est une méthode plus souple à mettre en place que le *Content based filtering*, ne nécessitant pas un travail d'étiquetage permanent de tous les nouveaux articles à recommander, plus réactive à d'éventuels changements dans l'environnement ou plus susceptible de prendre en compte la spécificité de chaque article à recommander.

Chapitre 2

Nous commençons le coeur de ce travail de thèse par la participation et la victoire au challenge *Exploration vs. Exploitation* n°2 organisé par PASCAL2, réseau d'excellence européen. Les résultats ont été présenté à la Conférence Internationale en Machine Learning à Bellevue, USA (ICML 2011) [5]. La contribution principale liée à ce challenge est la conception d'un algorithme de bandits contextuels *GAP*, basé sur un autre algorithme conçu par Microsoft: *AdPredictor* [6]. Rappelons que dans le formalisme des bandits contextuels, chaque décision se base sur un élément de contexte. Pour la recommandation en ligne ce contexte est le profil de l'utilisateur au sens large (page web, profil, session...). *GAP* améliore l'état de l'art de deux façons:

1. Il est plus souple. En effet, deux algorithmes existaient dans la littérature. LinUCB [7] utilise de l'information contextuelle sous forme d'un vecteur réel (ce qui est la norme dans le domaine), AdPredictor sous forme d'un ensemble de valeurs discrètes. GAP peut

utiliser de l'information regroupant valeurs discrètes et réelles.

2. Il est rapide. La complexité temporelle de l'algorithme considéré comme référence en bandits contextuels (LinUCB) est en cubique en la dimension de l'information contextuelle. Celle de GAP est linéaire.

Au delà de cette intéressante contribution, la façon de gagner le challenge a motivé le sujet du reste de ce travail de thèse: l'évaluation basée sur des données réelles. Avant d'aller plus loin, définissons quelques concepts. En général, la qualité d'un système de recommandation est mesurée par le taux de clics des utilisateurs sur les articles recommandés par le système (Click Through Rate ou CTR en anglais). En effet, si la satisfaction client ou le volume des ventes sont les vrais paramètres qui nous intéressent, leur corrélation avec le système de recommandation est difficilement mesurable. C'est pourquoi le CTR, facilement mesurable et directement lié à la qualité des recommandations fournies, est considéré. Sans mettre un système en ligne, mesurer ce CTR devient plus difficile. Dans la littérature, on utilise souvent des méthodes dites indirectes pour palier à ce problème [8]. L'idée est simplement de mesurer autre chose que le CTR en partant du principe que cette autre chose y est corrélée. On peut par exemple citer la précision de la prédiction de "ratings" d'articles faits par les utilisateurs (ou encore du CTR), utilisée dans le Netflix Prize, la plus connue des compétitions ayant lien à la recommandation (2006-2009 [9]). Bien entendu ce type de méthode s'oppose aux méthodes directes, qui fournissent une estimation du dit CTR. Le challenge que nous avons gagné définissait une méthode indirecte d'évaluation de la recommandation dynamique et plus spécifiquement, une tâche d'apprentissage en ligne, légèrement différent de la recommandation était proposée aux participants. Malheureusement, pour gagner nous n'avons ni exploré, ni pris en compte les dynamiques du problème et pire encore: notre approche est exclusivement basée sur la sélection d'un utilisateur, chose impossible en recommandation. Dès lors une question majeure se pose. Comment développer des systèmes de recommandation dynamiques performant dans le monde réel sans méthodes d'évaluation appropriées ? L'absence de réponse à cette question justifie la suite donnée à ce travail de thèse: l'étude et la proposition de méthodes *directes* d'évaluation de tels systèmes.

Chapitre 3

Un algorithme d'évaluation de la littérature (*replay*) a été publié en 2008 [10] puis il a été démontré qu'il était non biaisé en 2011 [11], coïncident avec ce nouveau choix d'orientation. Le travail présenté dans ce chapitre part du constat suivant. Si *replay* semble prometteur, il reste très peu étudié et cela dans deux domaines principaux. Théoriquement, la notion même de biais (ou de non-biais) le concernant est très floue. Empiriquement, ses caractéristiques ne sont mis en valeurs que sur des données privées et par des expériences non répétables. Dans ce chapitre nous nous attaquons au premier domaine cité.

Nous commençons par préciser quelques détails concernant le type de données utilisable par cette méthode. En quelques mots, parce que nous voulons pouvoir évaluer n'importe quel algorithme, il est nécessaire que la politique utilisée lors de l'acquisition des données explore tout l'espace. En d'autres termes et en simplifiant un tout petit peu, il est nécessaire que pour chaque contexte, la politique d'acquisition ait une probabilité non nulle de choisir toute action. Dans toute cette thèse, nous faisons l'hypothèse que les données sont acquises de façon uniformément aléatoire, ce qui bien entendu respecte les conditions précitées. Nous prouvons en fin de chapitre que bien que cette hypothèse puisse paraître bien trop forte, il est impossible d'évaluer un algorithme quelconque sans qu'elle soit respectée. L'hypothèse peut être assouplie de façon significative pour l'évaluation de politiques stationnaires mais pas pour un algorithme d'apprentissage.

Dans la partie principale de ce chapitre, nous analysons le résultat de la littérature concernant *replay*. Nous montrons que ce résultat ne prouve pas que l'estimateur n'est pas biaisé mais seulement qu'il n'est pas biaisé *asymptotiquement*. Aussi nous prouvons que dans un cadre fini, *replay* présente un léger biais que nous quantifions. Néanmoins nous reformulons le résultat de non biais asymptotique de façon à démontrer sa grande valeur. En effet au delà de fournir un estimateur non biaisé asymptotiquement et ce pour n'importe quel algorithme, la méthode d'évaluation a une caractéristique majeure : une séquence d'interactions qu'elle simule ne peut pas être différenciée statistiquement d'une séquence d'interactions avec l'environnement réel. De ce fait, nous avons possibilité de simuler sans biais le processus d'apprentissage d'un algorithme dans un environnement stationnaire. Après l'étude du biais, nous proposons une variante l'estimateur, *replay**, qui est non biaisée même en temps fini. Le coût intuitif est une légère perte de précision comme souvent en statistiques dans ce cas de figure.

Néanmoins, la suite de notre étude de *replay* est on ne peut plus surprenante. Nous commençons par reprendre un résultat de concentration avec une forte probabilité de Li *et al.* [11] et le rendons plus précis et ce de façon significative. Nous proposons ensuite un résultat similaire pour replay*. De ces deux bornes, celle de replay* est plus concentrée pour n'importe quelle valeur des paramètres, ce qui est évidemment contraire à l'intuition de départ. Nous utilisons une astuce pour montrer que la technique de preuve utilisée interdit de battre *replay**, chose que nous faisons pourtant en pratique. C'est pourquoi nous analysons la précision de ces deux méthodes sous un autre angle: leur variance. Cette nouvelle analyse permet de confirmer que replay est plus précis que replay* sur des jeux de données de taille raisonnable. Étant une méthode exacte, elle nous permet aussi de voir ce qu'il se passe lorsque moins de données sont accessibles. Dans certains cas que nous mettons en évidence, il reste préférable d'utiliser *replay**.

Chapitre 4

Sans l'accès à une application réelle, il est difficile d'évaluer une méthode d'évaluation. En effet, à quoi peut on alors comparer les résultats obtenus sur des données ? Une solution serait d'utiliser des données synthétiques pour lesquels la "vérité absolue" est connue et ainsi évaluer la méthode. Néanmoins cela ne nous dirait que peu de choses sur l'efficacité de la méthode utilisée sur de vraies données. C'est pourquoi afin de pousser *replay* dans ses retranchements et tester ses limites, nous avons organisé un tout nouveau challenge. L'idée principale est de soumettre l'estimateur à des chercheurs dont le but n'est pas de prouver qu'il fonctionne mais plutôt d'exploiter la moindre faille leur permettant de maximiser leur score. Les résultats ont été annoncés à ICML en 2012, soit un an après celui que nous avions remporté. Les données utilisées ont été fournies par Yahoo! [12], ont été parfaitement acquises et proviennent d'une grosse application de recommandation d'actualités. Notons que le but de ce challenge n'est évidemment pas uniquement de tester *replay* mais aussi d'inciter au développement de nouveaux algorithmes de bandits contextuels.

Une fois de plus, les résultats ont été très surprenants. Comme le challenge précédent, celui ci était organisé en deux phases. Lors de la première, les participants pouvaient soumettre un algorithme sur un serveur distant, évalué par *replay* sur le tiers des données Yahoo!. Cette phase a duré trois mois pendant lesquels les participants étaient invités à essayer d'améliorer leur algorithme par soumissions successives. Lors de la seconde, les organisateurs reprennent simplement la meilleur approche de chacun et la rejouent sur le reste des données de façon à mesurer la quantité de surapprentissage induite par les soumissions successives. Notons que lors du premier challenge, notre algorithme n'avait que très peu souffert de ce phénomène et avait remporté la seconde phase également. Sans surprise, la première phase s'est jouée entre les participants les plus assidus. De façon extrêmement surprenante néanmoins, le vainqueur de la seconde phase n'avait soumis que 4 algorithmes lors de la première (quand une quinzaine de participants dépassaient les 100, et 3 les 500 soumissions) et s'y trouvait à la vingtième place. Il avait en fait soumis un algorithme de la littérature - UCB [13] - *ne prenant pas en compte les contextes*. De cette façon, le surapprentissage était quasiment impossible.

Une conclusion évidente est que le surapprentissage a eu un effet colossal sur les résultats. Néanmoins avant de faire d'autres conclusions hâtives, nous complétons les résultats du challenge par quelques expériences supplémentaires. Nous testons tout d'abord le seul algorithme de bandits contextuels de la littérature: LinUCB [7]. Nous constatons qu'il est moins performant que UCB, algorithme non contextuel, constituant une seconde surprise de taille. Cela signifie-t-il que les contextes sont inutiles ? C'est ce que nous vérifions avec une technique de notre invention : *classify-to-recommend* (décrite en annexe B), permettant d'utiliser des algorithmes de classification pour construire un système de recommandation à partir de données. Il apparaît en fait possible (et ce de façon certaine) d'utiliser les contextes pour améliorer de façon très significative une approche simplement basée sur le meilleur article (non contextuel). Dès lors, une seule conclusion s'impose : si les algorithmes contextuels sont battus par les algorithmes non contextuels, c'est qu'ils n'ont pas le temps d'apprendre. En y réfléchissant à deux fois, cela a du sens. Pour ne pas être biaisé, *replay* rejette un grand nombre d'enregistrements du jeu de données. Pour être précis, il en garde un sur K, K étant le nombre d'actions. De plus, les données de Yahoo! ont certainement été acquises certes uniformément mais sur une portion de l'audience uniquement, mettons une personne sur L. De ce fait, une évaluation par replay dans notre challenge se fait avec $K.L$ fois moins d'évènements qu'en réalité. Avec une réalité statique, le biais resterait négligeable. Néanmoins, la recommandation dynamique, comme son nom l'indique est loin de l'être. Un algorithme a donc $K.L$ fois moins de temps pour apprendre, ce qui biaise énormément son évaluation. Nous appelons ce phénomène "accélération temporelle" et tâchons d'en réduire les effets dans le dernier chapitre.

Chapitre 5

Ce chapitre est de loin le plus riche et le plus complexe de ce document. Y sont exposées les contributions les plus importantes de ce travail de thèse. Dans ce chapitre, nous apportons des contributions distinctes à l'évaluation d'algorithmes d'apprentissage et de politiques stationnaires. Évidemment ces deux types de systèmes n'ont pas vocation à être utilisé sur le même type de problème de recommandation. Un algorithme d'apprentissage serait plutôt utilisé dans un contexte de recommandation très évolutif et avec peu d'actions quand une politique stationnaire serait apprise sur des données récentes et utilisée sur un problème moins dynamique et avec plus d'actions.

Nous proposons dans les deux cas une méthode de ré-échantillonage, BRED, inspirée du bootstrap [14], une méthode statistique destinée à estimer la distribution d'un autre estimateur. De cette façon, il est possible de dériver des intervalles de confiance et autres objets statistiques d'intérêt. Ici, nous construisons un estimateur de la distribution des données que nous avons (le problème de bandit + la politique d'acquisition). Cette distribution empirique est ensuite utilisée pour générer autant d'échantillons que nécessaire pour diminuer l'effet d'accélération temporelle induite par replay. La façon la plus simple d'estimer la distribution des données est simplement de procéder à des tirages avec replacement du jeu de données. Évidemment de cette façon un algorithme d'apprentissage est susceptible de rencontrer plusieurs fois le même enregistrement et de surapprendre plus qu'en réalité. C'est pourquoi nous ajoutons un autre mécanisme, que nous appelons Jittering et qui est inspiré d'une méthode appelée le "smoothed bootstrap" [15]. L'idée est d'utiliser une méthode d'estimation de la distribution des données un peu plus élaborée appelée Estimation de Densité par Kernel. En pratique, il suffit simplement d'ajouter du bruit sur les enregistrements (sur les contextes en fait dans notre cas). De cette façon, l'effet de surapprentissage est limité. Des expériences sur des données synthétiques (pour avoir accès à la vérité absolue) et des données réelles (moyennement une astuce) montrent un extraordinaire gain en terme de biais *et* de variance.

Il y a un autre aspect de cette méthode qui ne concerne que les politiques stationnaires. En effet, le bootstrap est à l'origine une technique d'estimation de distribution d'estimateur. En particulier, nous sommes intéressés par des intervalles de confiance de façon à minimiser (et contrôler) le risque de mettre une nouvelle politique en ligne. Néanmoins le bootstrap ne fonctionne que pour certains estimateurs et il ne fonctionne pas pour *replay* avec un algorithme d'apprentissage. Cependant, en étudiant théoriquement le cas des politiques stationnaires, nous obtenons des résultats surprenants. Pour cela, nous analysons deux aspects de notre méthode: la variance de son estimation du CTR de la politique et l'estimation bootstrap de la distribution de cette estimateur de CTR. Avec une forte probabilité, l'erreur de replay sur le CTR d'une politique se concentre en $O\left(\sqrt{\frac{K}{T}}\right)$, K étant le nombre d'actions et T le nombre d'enregistrements. Nous montrons ensuite que l'erreur de BRED se concentre en $O\left(\sqrt{\frac{\Gamma}{T}}\right)$, ou $\Gamma \in [1, K]$ quantifie l'accélération temporelle, ou l'inverse de la quantité d'information contenue dans les données. Ce qu'il est important de retenir, c'est que nous prouvons que Γ vaut moins que K si la politique peut faire deux choix distincts face à un enregistrement qui lui est proposé. Plus ce cas de figure est fréquent, plus Γ tend vers 1. C'est évidemment le cas pour des politiques stochastiques, dont nous démontrons l'utilité pratique dans ce chapitre. Il est extrêmement intéressant de noter que Jittering, ou en d'autres termes ajouter du bruit sur les contextes permet de forcer des choix différents, même pour une politique déterministe. Nous montrons empiriquement que cela accroît la précision d'évaluation.

Nous pouvons dès lors dresser des conclusions intuitives en ce qui concerne la précision d'évaluation d'algorithmes d'apprentissage. Trois mécanismes permettent à un tel algorithme de faire des choix différents étant donné un contexte:

1. l'exploration, qui consiste justement à faire des choix différents de la politique jugée optimale pour acquérir de l'information,
2. Jittering, pour les même raisons que pour les politiques stationnaires,
3. l'ordre des enregistrements: en effet, il est possible de rejouer l'algorithme plusieurs fois. Il verra alors des enregistrements à des époques différentes de son processus d'apprentissage et aura la possibilité de faire des choix différents.

C'est pourquoi en plus de réduire le biais du à l'accélération temporelle, nous réduisons la variance d'évaluation d'un algorithme d'apprentissage de façon extrêmement significative.

Pour les politiques stationnaires, nous allons plus loin en étudiant le bootstrap (qui ne marche pas pour un algorithme d'apprentissage). Nous prouvons que la concentration de l'erreur sur l'intervalle de confiance sur le CTR est en $O\left(\frac{\sqrt{\Gamma}}{T}\right)$ avec une forte probabilité, soit une concentration bien supérieure à celle du CTR lui même. Or soulignons que si l'on a une estimation d'un tel intervalle de confiance, il est inutile d'avoir une estimation du CTR quand l'objectif est de minimiser le risque de mettre une nouvelle politique en ligne. De ce fait, nous augmentons encore la précision de notre méthode en estimant de surcroît un objet plus utile.

Conclusion

Nous concluons en mentionnant trois catégories de perspectives par ordre d'importance croissante. Tout d'abord, bien que BRED soit une méthode tout à fait fonctionnelle en l'état, elle peut être améliorée pour les algorithmes d'apprentissage. L'ajout de Jittering introduit un biais incontrôlable. Pour palier à cela, nous pouvons introduire un mécanisme très intéressant que nous décrivons à la fin du chapitre 5: la validation entremêlée. Ce mécanisme permet d'interdire tout biais positif sans introduire de variance supplémentaire. Rappelons qu'un biais positif (surestimation du CTR) est extrêmement dangereux et fait exploser le risque que nous essayons de minimiser. Au contraire, un biais négatif n'engendre aucun risque. Nous commençons aussi à étudier cette méthode de plus près dans l'annexe A mais il serait très intéressant d'aller plus loin. Une autre perspective serait de justifier théoriquement le gain de performance induit par le Jittering.

La seconde perspective est l'élaboration d'une méthode de construction itérative de systèmes de recommandation. Avec l'énorme gain de performance de BRED comparé à replay, il devient possible de considérer ce genre de technique pour n'importe quel type de problème de recommandation. Aussi, si la politique évaluée est proche de la politique d'acquisition, le taux de concentration devient proche de $1/T$. K n'apparaît plus, rendant ce genre de technique possible quelque soit le nombre d'actions possibles. Se basant sur ces conclusions, il serait tout à fait possible de construire une méthode qui améliorerait ses politiques itérativement, utilisant les données acquises par la politique précédente pour évaluer la suivante.

La troisième et plus importante perspective est simplement d'utiliser les méthodes proposées dans ce travail de thèse pour commencer à travailler différemment sur les problèmes de recommandation. En effet, elles permettent dès à présent à n'importe quel chercheur, même s'il n'a pas à disposition une application réelle de faire deux choses:

- construire des algorithmes de bandits contextuels efficaces pour des applications de recommandation dynamique,
- évaluer des systèmes de recommandation classiques avec une méthode directe d'estimation du CTR.

Introduction

Abstract *This section is the general introduction of this thesis work. It introduces its historical context as well as the contemporary problems it tackles. The last paragraph justifies the organization of the presentation of this thesis. It is also meant as a map to navigate through the various chapters of this document.*

This thesis work is about problems from decision theory in which an agent has to make decisions in a uncertain environment, that is of which he does not have a full knowledge. This set of problems is commonly referred to as *decision making under uncertainty* and represents the heart of the aforementioned theory. Indeed in the real world a decision maker rarely has a hold over all the variables describing his environment for a lot of them can be influenced by political decisions, the climate, other people or even randomness. As a consequence solutions to those theoretical problems find a wide range of applications in areas such as finance, gaming, engineering, robotics, logistics *etc.* Decision making under uncertainty have been considered an important problem for a while now. Indeed, work on the subject can be traced back to the 17th century. Blaise Pascal in his posthumous *Pensées* [16] published in 1670 describes how to bet using what we now call *expected values of random variables.* A few decades later, in 1738, Daniel Bernoulli published a highly influential paper entitled *Exposition of a New Theory on the Measurement of Risk* [17]. In this founding work, he introduces the notion of utility functions based on risk. He justifies his new approach with the famous *Saint Petersburg paradox*, first formulated by his cousin Nicolas Bernoulli in 1713. It illustrates that a criterion uniquely based on the expected value can take very unreasonable decisions even when the expected values under consideration are infinite. He gives a famous example after which the paradox was named in which a modest Dutch merchant has to decide whether or not he should send a ship to Saint Petersburg in the winter knowing that there is a 5% chance that it will be lost along with its cargo. Given the relatively small probability of shipwreck, the expected value of gain is bound to be positive. Yet such an accident would most likely lead the merchant to never be able to buy a new cargo again and thus to go bankrupt. Therefore, contrarily to what the study of the sole expected value of the gain would suggest, sending the ship is highly unreasonable. Nevertheless, a very wealthy merchant who possess an entire fleet of ships and plethora of goods to sell may be willing to take that risk for loosing one ship would not hurt his business. In a nutshell Bernoulli [17] proposes to consider the expected logarithmic wealth instead of the expected gain so as to reflect the usefulness of an amount of money with respect to how much of it one already has.

Ever since these founding works were published and given their wide range of application, decision making under uncertainty have raised a lot of interest from many people. We can for instance mention a paper by Wald (1939) [18] in which he points out that the two central procedures of the statistical theory - hypothesis testing and parameter estimation - are in fact special cases of the general decision problem. Even nowadays, a lot of work is still going on on that subject in various communities such as statistics, economics, medicine, artificial intelligence *etc.* In mathematical optimization, decision making under uncertainty is usually addressed under the name *stochastic programming* referring to the fact that as opposed to regular optimization, in most real problems, many variables are unknown or only known within certain bounds (*robust optimization* in that case). A recent book entitled *Approximate Dynamic Programming: Solving the curse of dimensionality* [19] proposes a set of tools to apply these methods to large scale real life problems, *e.g.* fleet management, smart energy resource planning or locomotive management. It is also a very hot topic in management science as the essence of a manager is to take decisions given highly uncertain variables such as the state of mind of employees, political issues, the market situation *etc.* See Prelec *et al.* [20] for a classical approach of the problem in that context. Finally the problem of *decision making under uncertainty* is also addressed in more unexpected communities. For instance a recent work by environmental biologists [21] sees the conservation of endangered species as a management problem in an uncertain environment.

In this thesis work, as well as in machine learning in general, we focus on a special class of decision making problems. In such problems an agent has to make *sequential* decisions in an environment that provides outcomes after each action performed. A typical goal for the agent is to maximize the sum of real valued outcomes. This work also focuses on methods to *evaluate* the behavior of an agent interacting with its environment thanks to datasets of past interactions. This sequential problem is well studied in machine learning under the name *reinforcement learning* [1]. The environment is modeled as a *Markov decision process* that is to say a stochastic state machine: when the environment is in a given state, an action performed by the agent can make the environment jump from one state to another. More precisely, in a given state some other states are reachable with unknown probabilities. The probabilities of transition to the different states as well as the outcome of each action obviously depend on the state the environment is in. Also, before the beginning of what one may call a *game*, that is a sequence of interactions of fixed length or not, the agent has no knowledge of the transition probabilities and the outcome, or *reward* distributions. Consequently to achieve an optimal behavior, the agent has to *learn* how to navigate efficiently in this uncertain state machine. Note that here, *uncertain* refers both to the stochasticity of the environment and the fact that its characteristics are unknown. A simpler version of this problem which has been well studied as well is the *multi armed bandit problem* in which the environment has only one state. This means that the actions of the agent does not affect the environment and that the only thing to learn is how the environment maps actions to outcomes. In this work we will actually focus on a less studied but much more practical variant of this problem called *Contextual Bandits* in which the agent can use some contextual information to make each decision. Indeed in the modern world, information is everywhere and failing to use it would be a shame. A classical example is advertisement display of the web. An agent (here a piece of software) has to choose an ad to display given plethora of contextual information: user profile, current web page, location, time of the day, current session, social networks *etc.* Note that if the environment is not assumed to be altered by the actions of the agent, it is however important to have in mind that the environment can be *dynamic* that is some of its characteristics can evolve over time due to external events. This feature is rarely taken into account in the literature but will be addressed in this work as most of the real life applications can be considered to be dynamic.

Many applications of *bandits* are discussed in the literature so let us mention the two main categories while keeping in mind that plenty of others exist. Bandits find a lot of applications in the medical field. For instance they can be used to automatically assign a medicine to a patient, learn from his response to the treatment so as to converge toward the optimal one. Bandits are also a typical solution in brain computer interfaces in which one does not know at first how the user will respond to various stimuli. Nevertheless the main applications we will consider in this thesis are the intelligent systems (that is an agent whose behavior was coded into a software) on the web that have to make millions of decisions a day. In particular we will focus on probably the most well-known of them: recommender systems. Note however that one may think of many other systems such as crowd sourcing, ad display or even video game-oriented artificial intelligence. In a nutshell a recommender system is a software that has to decide which content to provide to a user whose current preferences are highly uncertain given some partial information we may have about him (contextual information). Typically one may want to maximize the *Click through rate* (CTR) or the *Conversion rate*[1] if the recommended content is a product. Note that for this kind of applications it is perfectly reasonable to assume that a decision, that is the recommendation of some content to a user does not affect the environment, that is the preferences of all the community of the website.

Having a sense of the performance of those systems before using them in the real world can be crucial. The reasons in the medical field are obvious enough not to be mentioned although they are usually checked by an expert. With intelligent web systems, especially on big website a significant drop of performance of the system could turn into a significant economic issue. Indeed it would impact negatively the present sales and user experience. Worse it could discourage users from coming back and so impact the business of the company in the long run. Furthermore updating or changing a recommender system can be expensive in terms of engineering efforts so that people in charge would not be willing to agree with doing so without hard evidence that the generated added value will be worth the trouble and money. In the past people have designed models to try their learning algorithms but such models are biased by nature and making them better is probably much more difficult than designing the system itself. Nevertheless a very nice characteristic of web applications is that they generate a lot of data. Although using that data to evaluate a system is far from being straightforward given its learning nature, being able to do so would bring a lot more confidence in the results than with a hand-made model. Building accurate such methods is the main contribution of this thesis work. Also, with the same concerns in mind as Bernoulli in his work from 400 years ago, we will pay attention to estimating more than the expected value of some performance metric (*e.g.* the CTR). Indeed since putting online a “bad” system might be very costly, an estimation of the performance of a new system is not enough. We need to quantify and control the risk of putting a new system online, that is the risk of having overestimated its performance. In this work we will not design nor use utility functions that allows such control. Yet we will keep that concern in mind and provide what is needed to do so, namely variance analysis, error bounds and tools to estimate accurate confidence intervals.

Finally, before diving into this work and its contributions that we detail below, the reader should have one last detail in mind. In this thesis we tackle the *Contextual Bandit Problem* and in particular ways to evaluate its solutions accurately using real data. We claim that the main application of this work is recommendation. Nevertheless it should be perfectly clear that this work does not tackle the evaluation of recommender systems the way it is most commonly tackled in the literature or in the industry. In fact the evaluation of a recommender system is classically considered as a very difficult problem: so difficult that people generally use performance metrics that do not directly measure the quantities they are truly interested in (indirect methods of evaluation). In this work we consider and propose direct methods, that estimate exactly what we want. Another point worth stressing out is that *Bandit algorithms* only apply to a special case of recommendation problems that we call *dynamic recommendation.* Dynamic recommendation has two main features:

- maximum 100 items are available for recommendation at a given moment
- and one item is available for a limited amount of time (typically between one hour and a day on a big website).

The evaluation methods that we propose and then study are first designed to evaluate contextual bandit algorithms played in the context of dynamic recommendation. Nonetheless we will see throughout this work (but mostly by the end) that the methods we propose can be used way

[1]The conversion rate is the sales volume or sales revenue per recommendation.

beyond that scope and in particular in the context of classical recommendation, with thousands or millions of different items.

Dynamic recommendation problems, our main application, are getting more and more common, its most well known example being news recommendation. This contrasts with some of the typical applications such as e-commerce or movie rental in which millions of items can be recommended. In the limited literature about dynamic recommendation, online learning approaches and bandits in particular are claimed without further explanation to be *the* solution. Yet bandits or even online learning solutions are never mentioned in the main handbooks about recommendation. Indeed the news recommendation issue is typically addressed with a technique called *content-based* recommendation. Since such dynamic recommendation is the main motivation of this work, we thought crucial to clarify all this and give a precise sense of when and why bandit algorithms should be used in recommendation instead of the classical approaches from the literature such as *Collaborative Filtering*. One may consider that the in-depth motivation of the use of online learning and bandits in particular is our first contribution since to our knowledge, this was never done before (see section 1.4.1).

Basically this work brings two kinds of contributions: (i) learning algorithms to deal with the contextual bandit problem and (ii) data-driven methods to evaluate them. If the second kind clearly gathers the most significant contributions of this work, it was not the primary purpose of this thesis. Indeed while studying the literature and starting working on bandit algorithms meant for real problems, it appeared quite clearly that their evaluation was a not very well-studied yet crucial problem. As we explain it in what follows, such an evaluation based on real data is not trivial. Yet providing some solutions would definitely be a leap forward for the field. Indeed, as already mentioned, industrial applications would naturally benefit from less biased evaluations but that is not all. They would also benefit from contributions from the academic world who would be able to design algorithms that tackle real life issues. Indeed, most of the current research, by lack of means to deal with real data, is only evaluated by theoretical analysis and evaluations on toy models.

This thesis document should really be seen as four parts of a whole, four steps of a full research process. As previously mentioned, it starts with the motivation of the use of bandits in the context of recommendation. The second part, our first work, is the design of an online learning algorithm, based on real recommendation data within the context of a competition than we won. Considering the fact that this thesis is primarily about evaluation, this part might seem slightly out of place. Yet it is a perfect illustration of all the problems raised by the lack of evaluation methods based on real data. This work also highlights the difficulty of designing such an evaluation method and what can occur with the use of an ill-suited one. While trying to draw conclusions from this experience, it appeared very clearly that given the current state of the art, a thesis work on offline evaluation would be much more significant for the field than a work on real-data-bandits. This is what the last parts, which are also the most important in both size and significance are about. The first one is a theoretical and empirical study of a new evaluation method from the literature and a very similar method of our own. This study in turn revealed issues we had not been predicted and which we deal with in the second big part. Thus this document can be seen as the story of how we came to work on data-driven evaluation methods and the solutions we brought to problems we were not fully aware of at the beginning. As the reader will probably realize while reading this document, each part of this thesis is in fact a research work motivated by the previous one.

More specifically the document is organized as follows:

- Chapter 1 sets the background and the notations necessary to understand this work while performing a literature review. Note however that different approaches could have been envisioned for this review. It may sound surprising but we chose to center everything around recommendation. Indeed recommendation is not the only application for Contextual Bandits. Respectively Contextual Bandits are only one of the solutions to one specific recommendation problem (dynamic recommendation). The reasons for this focus are in fact simple. First recommendation is, in our opinion, the main and most promising application for *Contextual Bandits*. Second because the bandit approach or even more generally the online learning approach is not considered in the recommendation textbooks which is a shame in our opinion. This is why we thought crucial to motivate in-depth the use of *Bandits* in this context and to specify their area of utility. This extensive justification can even be considered as our first contribution for to the best of our knowledge, it has only been done quite shallowly in the literature (mainly in the abstract or introduction of papers on bandit theory). So in this chapter we review the two main approaches to recommendation: Collaborative Filtering and Content-Based Filtering. Then we review the main bandit algorithms and motivate their significance and usefulness in the context of recommendation. We also introduce the *Multi-armed bandit problem*, a simplification of contextual bandits which is not fit for personalized recommendation but was intensively studied for decades. Finally as this is the subject of our main contributions, we briefly tackle the evaluation of recommendation.

- Chapter 2 is actually the beginning of the story. As our goal was to design solutions to the contextual bandit problem and its applications such as news recommendation, we entered a challenge proposing to deal with a similar task, evaluated on real data. The main part of this chapter relates how we won the challenge and describes the algorithm we used and another simpler one that we derived to compare with the state-of-the-art contextual bandit algorithms. Among other things, this new algorithm has the advantage of being much more computationally efficient than its state-of-the-art equivalent and much more flexible in terms of types of contextual information it can use. Finally this challenge acted as an eye opener on the need for the design of new evaluation methods for contextual bandits. Indeed the evaluation method turned out to be so biased that the solution we came up with to win would be of absolutely no use to perform actual recommendations.

- Chapter 3 describes a data driven evaluation method for contextual bandit algorithms that was made popular by a paper published roughly at the time of the workshop of the aforementioned challenge organized within the *International Conference in Machine Learning* (ICML) 2011. This replay method is said to be *unbiased* in the title of the paper featuring it. Nonetheless although theoretical guarantees are given for various settings, the notion of unbiasedness remains quite fuzzy. This is why in this chapter, after describing the method and reviewing the results obtained by its authors, we derive new theoretical results in order to lift this annoying blur. This analysis leads us to define a variant of the method that we analyze as well and which is strictly unbiased contrarily to the existing one. We then go a lot further by sharpening existing concentration results and exhibiting that surprisingly enough, they provide a rather counter intuitive idea. We tackle this counter-intuition by a different approach to analyze statistically the error of an estimator: a variance analysis. We also study the importance of the way the data is acquired and provide theoretical results with that regard.

- After the relative failure of the previous challenge, Yahoo! offered us the opportunity to organize a new one, based on the replay method that we describe in chapter 3. To do so they provided us with a brand new dataset acquired on a news recommendation application which receives millions of visits a day. Therefore chapter 4 is about this new challenge which can be considered as a large scale experiment to push the replay method to its limits without being biased by our preconceived ideas. We also took advantage of this opportunity to test the empirical performance of various state-of-the-art bandit algorithms (both classical and contextual) along with those of two less common algorithms that we designed and have interesting properties. We finally argue in this chapter that the surprising outcome of the challenge is due to a totally unexpected bias in the evaluation method caused by a phenomenon that we named *time acceleration*. Empirical evidence of this statement are also provided in this chapter.

- Chapter 5 is presents the main contributions of this thesis work: an attempt to provide a solution to the time acceleration problem that we exhibited thanks to the challenge we organized. In this chapter we first lay the theoretical foundations for this solution based on the popular *bootstrapping* techniques. As it is explained at length in this chapter, this new method - which could also be qualified as a set of methods - goes way beyond a simple accuracy gain. In addition to that, which is already a significant contribution, we provide ways to quantify the knowledge extracted from a dataset. More importantly, we derive tools that allow us to build more than a simple estimator of performance. These tools allow us to output confidence intervals, which we argue to be much more useful than simple estimations of performance for they allow to control the risk of putting a new system online. Finally, all these new tools are analyzed theoretically and justified empirically on both real data and synthetic data. One extremely interesting feature that comes out of the analysis is that we are able to estimate what matters the most in practice - *i.e.* confidence intervals - with a precision of higher order than the one of the estimation of performance. Last but not least, we also introduce a concept that we call "iterative improvements". We argue that such a concept can allow us to use replay methodologies with an acceptable accuracy on a much wider range of recommendation problems than simply what we define as dynamic recommendation.

- We conclude chapter 5 by introducing a totally novel concept - with ideas borrowed from the supervised learning field that we call *entangled validation*. Indeed our method of evaluation introduces a bias so as increase the accuracy that can lead to overfitting. Entangled validation aims at preventing this from happening. The technique has guarantees and is already quite mature. Yet it lacks empirical justification so most of its presentation and leads of future research are presented in appendix (A). However we do believe this method to have a huge potential in the context of bandits and in possibly other fields so we do encourage the reader to read appendix A. This document contains other appendices that shall be referred to when needed throughout the presentation of this work.

To summarize this thesis work contains three major contributions:

1. A new contextual bandit algorithm that is faster and more flexible than its state-of-the-art equivalent. Its performance are demonstrated by both a victory in an international challenge and a set of experiments on synthetic data.

2. A deep theoretical analysis of the *replay method* and a large empirical study. Among other things, this study comprises a completely unique experiment in terms of scale and nature: the organization of the *Exploration versus Exploitation challenge 3*. This study led us to identify the tremendous bias induced by the *time acceleration phenomenon*.

3. The last contribution a set of methods that reduces time acceleration, improves replay in term of both variance and bias and allows the computation of confidence interval.

Note that this third set of contributions is definitely the richest and most significant of this thesis work.

Chapter 1

Background, motivation and notations

Abstract of the chapter *This chapter sets the scene and justifies the significance of this thesis work within the current state of the art. With this aim in mind, we start with a brief literature review of recommendation and its two main approaches: Collaborative filtering and content-based filtering. We also tackle the ways recommender systems are generally evaluated offline. Finally, we make a review of a problem called "Contextual bandits". We argue that its solutions can be used in some specific domains of recommendation for which the classical approaches and the ways they are evaluated exhibit severe limitations. In this part will also be set most of the notations in use in the rest of this thesis. Making the literature review of this thesis work mainly on recommendation is far from trifling for two reasons. The first is that we are interested in online learning systems in general and the one modeled by bandits in particular. Dynamic recommendation as we named it is only one example of such problems. Yet it is clearly the most widely used and most emblematic. The second reason is that online learning is not typically mentioned as a solution to recommendation. Consequently, this literature review will also be a pretext to highlight how and why online learning should be included in the state-of-the-art solutions to recommendation.This is actually the first contribution of this thesis work.*

Keywords Recommendation, Collaborative Filtering, Content-based Filtering, Evaluation, Bandits, Contextual Bandits, Online learning.

Contents

1.1 Introduction

This chapter is meant to set the scene of this thesis work but most of all to justify its significance which will mostly be done in the section about *Contextual Bandits*. The literature review, the vocabulary used throughout this document and the thesis in general is mostly oriented toward recommendation. Yet this could actually have been completely different. Most of the contributions presented in this document are indeed related to contextual bandits in general that have other applications than recommendation. Nevertheless, we think that it is important when working on theoretical problems such as bandits to keep in mind their applications to see, for instance, if the assumptions that are made are reasonable. This is why in this thesis work even when talking about theory, recommendation will always be somewhere just around the corner. Recommendation came as an obvious choice for several reasons. Firstly it is the very reason why *Contextual bandits* were introduced under the name of *Bandits with side information* by Chih-Chun *et al.* [22] in 2005 or under the aforementioned denomination by Langford *et al.* [23] in 2007. Secondly recommendation is a very hot topic in research and is in use all over the web in commercial applications, a few of which are extraordinarily big and generate a lot of money. Thirdly most of the experiments concerning *Contextual Bandits* in the literature are oriented toward recommendation and the same goes for the data available online. Note however that despite this focus on recommendation we mention a few of other applications in this chapter. We do not go into the details. Yet the interested reader should be aware that all the results presented in this thesis do apply to these other areas of application of (contextual) bandit algorithms for no assumptions specific to recommendation are made.

As a consequence, this chapter is organized as follows.

- We first talk about recommendation in general and then about its main two approaches: Collaborative filtering (CF) and Content-based (CB) recommendation, insisting on the strengths and limitations of these methods.

- Then we dive into the closer background of this work. We first motivate the use of online learning and in particular the *Contextual bandit problem* formulation to overcome some limitations of the classical approaches to recommendation in some special yet widespread cases. We refer to these special cases as *dynamic recommendation* since they present a small-sized pool of items that constantly and rapidly evolves. Such a thorough motivation was never done to the best of our knowledge, making it the first true contribution of this work. Unfortunately contextual bandits are not very well-studied, which is one of the motivations of this research. On the contrary, a sub-problem called *Multi armed Bandits* was intensively studied over the last decade and work on the subject can even be traced back to the 30s. Therefore before reviewing the very few approaches to contextual bandits, we also review the major approaches to the simpler problem. In a nutshell, in *Multi Armed Bandits* one has to learn which is the best action assuming that the outcomes they produce are *i.i.d.* and without further information. In *Contextual Bandits*, one can rely on a piece of contextual information to make each choice of item. This problem is obviously much more realistic when considering web applications given the plethora of information available (user profiles, sessions, current web page, location, social networks *etc.*). Yet this makes the problem much more tricky to analyze theoretically because of the multitude of forms that information can take, hence the little amount of work on the subject. This is also what makes it much more useful in practice.

- Finally as the main part of this thesis work is about offline evaluation, we make a brief review of how recommendation is evaluated and of the benchmark considered as standards for new algorithms. However evaluating online learning algorithms using real data has always been a tricky question in Machine Learning and contextual bandit algorithms are no exception to that rule. This is the starting point of the main contributions of this thesis work.

1.2 Recommendation: a definition

To give a definition, *recommendation* is the action of *recommending* or proposing some hopefully interesting/helpful content to a user visiting a web site. Regardless of the nature of the content, we call an *item* a piece of that content that can be recommended to a user. A software performing such a task is often called a *recommender system*. Each Internet user keeps coming across recommender systems whose results are displayed in frames entitled for instance "Customers Who Bought This Item Also Bought", "Frequently Bought Together", "You might also like" or more soberly "Recommended for you" (figure 1.1). For example if the starting page of your browser is a big web portal, the first thing you probably see when you open it is one or several news recommended to you.

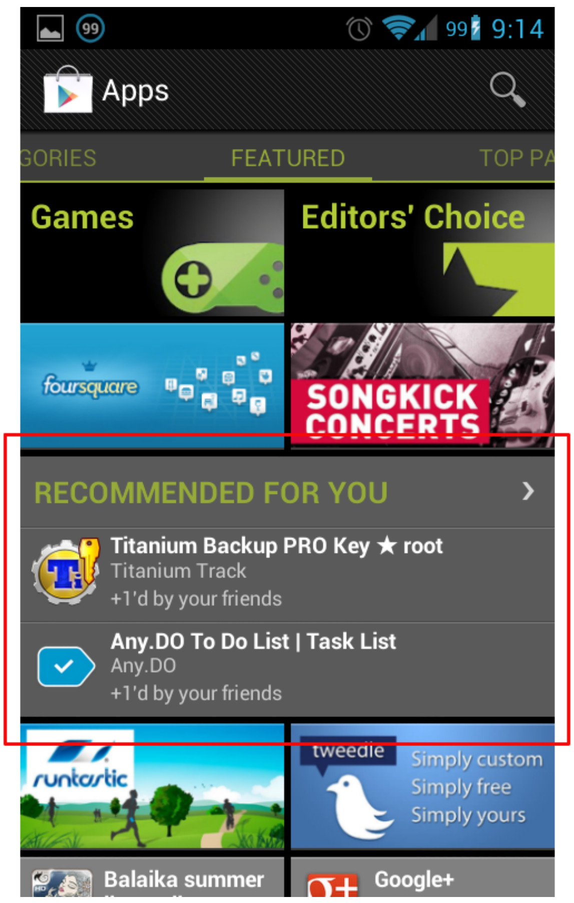


Figure 1.1: Recommendations on the application market of a smartphone.

Notice that we are talking about *personalized* recommendation: that is to say each user is being proposed a different item or list of items depending on his or her tastes. Theoretically, a website displaying its top-sales or most read articles could be considered to be performing low cost recommendation which could even turn out to be quite efficient if some items are indeed liked by everyone. However what is meant by recommendation is offering personalized content to users and this is what is considered in this field of research. Indeed, *Harry Potter* may have been the best-selling book in *2007*, people who never read fantasy novels or people who have already bought the book might find such a recommendation unhelpful.

Even though recommender systems are getting more and more pervasive, it is also interesting to have in mind that they are in general only a side feature of a big application. Such an application can range from e-commerce to online dating, news websites or even *YouTube*. The main application can very well exist without the recommender system but its addition aims at enhancing the user experience of the application. To do so the system allows the user to find appealing content he or she would have had trouble finding by him or herself or that he or she might not have even looked for in the first place. Nevertheless there exists a few exceptions, that is applications that are a recommender system and *vice versa*. The most well-known example is *Pandora* which is a completely personalized web-radio based on user ratings, demographics and an extraordinary endeavor of music manual labeling: *The music genome project*. It took over five years of development to complete the labeling of all the songs included in the resulting dataset. The extremely high quality of the recommendations resulting from such a dataset fully justifies the recommender system to be a standalone application. However datasets of that quality are not legion which proves that Pandora can only be an exception. To a lesser extent we can also mention *Digg*, a successful news recommendation website which serves as a gateway to other news websites.

Recommendation has a story quite parallel to Web 2.0. It emerged in the early 1990s as the flow of Internet users was increasing exponentially and that the idea of letting the users contribute to the web was making its way. Someone came out with the seductive but simple idea to harvest the opinion of this growing mass in order to help each person individually get what he or she wants. At that time fairly simple methods led to very encouraging results which resulted in a great enthusiasm and a lot of applied research papers started popping up. Ever since recommendation has grown into a very hot topic in information retrieval, machine learning, artificial intelligence or even marketing for the obvious reason that, for a company, the first consequence of a good recommender system for one of its web application is a revenue increase which can be direct (more product sales) or indirect (more advertising revenue due to more numerous and longer user visits). Furthermore in the early 2000s the availability of numerous big datasets from these new applications created a big excitement within the academic research community. The Netflix prize [9], launched in 2006 is probably the most well-known example. It brought together thousands of researchers and led to major advances.

1.3 State-of-the-art Recommendation

The idea of the chapter is to introduce the notations on contextual bandits but mostly to motivate their use in the context of (dynamic) recommendation. To achieve the latter, we first need a review of recommendation and its state-of-the-art solutions: Collaborative filtering and Content based filtering. After identifying the strength and weaknesses of these approaches, we will be able to justify why contextual bandits should be seriously considered when dealing with dynamic recommendation problems.

Note that a few other approaches exist in the literature but we will do nothing more than mentioning them. Most of the content presented in this section is inspired by the reading of two excellent books. The first one, *Recommender systems handbook* [3] is a collection of major works concerning many different aspects of recommendation. The second one, *Recommender Systems: An Introduction* [24] is significantly less comprehensive but the fact that it consists in one coherent overview of recommendation makes it smoother to read.

1.3.1 Collaborative filtering

Collaborative filtering (CF) is a method based on a very seductive yet quite simple idea: using the past opinion/behavior of an entire community can help predicting the current tastes of one particular user. The basic principle is that *similar* people in the past will in general remain so in the future. The idea and terminology were introduced by Goldberg *et al.* [25] in a famous article entitled *Using collaborative filtering to weave an information Tapestry* and published in 1992. At the time, CF was already much more than a theory since the authors of the aforementioned article had implemented their ideas in the Xerox PARC Tapestry system, a system able to recommend documents to its users thanks to the community's *annotations* in addition to the content of the documents. The paper considers the general case of annotations that could be complex objects (made of text, numerical evaluations of various criteria) and not only one single rating or vote. This system can be seen as the ancestor of popular recommendation websites such as *del.icio.us* and *Digg*. It is also considered by many to be the first work on recommendation in general. Besides the obvious scientific interest of this founding paper with regard to recommendation, it is really amazing to see how clearly web 2.0 had already been pictured more than 20 years ago. At the time (in 1992), even the web 1.0 barely existed as the Internet only consisted in less than 1,000,000 interconnected computers, the WWW had been opened to the public only a year ago and the Internet society was just under creation [26, 27].

Collaborative Filtering (CF) is a rich topic. Also because it is not directly connected to the core of our work, we only provide in this chapter a shallow overview of its classical and current approaches. Yet a more thorough review can be found in appendix C with much more details and references on what is exposed here. We strongly encourage the reader who is not familiar with recommendation to go through that appendix for it will provide a better understanding of the background of this work and its motivation.

CF is based *only* on a matrix of ratings of items we want to recommend issued by the users of the application (one row per user, one column per item). A rating is either explicit (a user rated an item) or implicit: no rating is issued but they are inferred from the user behavior (click rate, purchases, time spent on a page *etc.*). The first approaches - among which the one by Goldberg *et al.* [25] - were what we call either user-based CF [25, 28] (appendix C.1) or item-based CF [29] (appendix C.2). The idea is to either compute similarity measures between the users (based on their vector of ratings on the items) or between the items (based on their vector of ratings by the users). We then use the K most similar vectors to fill in the gaps of the - very sparse - vector of ratings corresponding to either a user or an item and make recommendations based on the highest predicted ratings. Note that item-based CF is what is used by Amazon with phrases such as "users who bought this also bought...". Such approaches have had quite some success until it became impracticable to compute the similarities in real time. In general, approaches that require to have access to all the data at all time are referred to as memory-based. From this point, people started coming up with model-based approaches that precompute a compressed version of the data to make recommendations. The first ones were user or item based approaches with precomputed similarity measures. Soon enough however new approaches that turned out to be more efficient were derived.

The most emblematic technique is matrix factorization [30–32] which gained a tremendous popularity after the Netflix Prize [9] started in 2006 and completed in 2009. Yet it was introduced in the context of recommendation way before that (in 2000 to the best of our knowledge) by Sarwar *et al.* [32]. The whole idea of this method is to infer latent features from the rating matrix, characterizing both users and items. These factors can then be used to fill in the blanks in the matrix by allowing to reconstruct it entirely and thus make predictions. A intuitive way to look at this technique is to consider that it allows to decompose an item (say a movie) into characterizing features such as its amount of romance, action, suspense *etc.*and similarly, to decompose a user into their attract to each of the categories. More formally, we compute a Singular Value Decomposition so that R, the M by N matrix of ratings is decomposed into $U\Sigma V^T$, with Σ a F by F diagonal matrix containing the singular values, F being the arbitrary number of latent features. The rows of U (resp. V) are the decompositions (or compression) of each user (resp. item) of size F in the latent features space. We can then use this decomposition to fill in any gap in the original matrix and make recommendations.

Before discussing the strengths and weaknesses of this approach, let us make one remark. CF in general and matrix factorization (MF) in particular is a very generic way of tackling recommendation. Yet it was shown to work extremely well in practice and contributed to the victory in the most popular competition in the field. Also it is a nice bridge between a practical application and an extremely theoretical field: linear algebra. As such it allows a wide range of theoretical researcher to work on a very practical issue. As a consequence MF acquired a tremendous popularity in both the industrial and the academic world, so big that sometimes MF almost becomes a synonym of recommendation (see appendix C.3.4). Yet the reader should keep in mind that MF is only a very small part of what makes a recommender system. Indeed, MF is about finding the best way of completing the matrix such that the error on *rating prediction* is minimized. However if we only consider the Machine Learning part and skip things such as the user interface, one can think of other issues such as:

- the data we need to acquire to improve the system (active learning),
- the cold start problem (new system, new users, new items),
- the offline evaluation process,
- what to actually recommend given the predictions, in particular when several recommendations can be issued at the same time or when session information is available.

Despite that, CF has many strengths:

- it works extremely well,
- it is generic so (i) all recommendation domains - as different as they may be from one another - take advantage of advances in this field and (2) no feature engineering effort is needed as in most machine learning systems,
- it requires little effort from the user (no profile),
- it provides interpretable results (similarities between users/items) for both the engineers and the users [29] which proved to improve the perceived quality of the recommender system [33, 34],

It has however a few weaknesses such as:

- it needs a lot of data to perform well,
- it can not handle new items, new users (cold start problem),
- it can not use the very rich side information available on the web (user profiles, textual or visual representation of the items, prices, social networks *etc.*).
- it reaches a “performance wall”.

This “wall” was exhibited in the Netflix challenge and all the following work. On the Netflix data, people ended up with an error of about 0.9 on a scale of range 4. This is huge, especially considering that it would be hard to go much further given the tremendous endeavor that led to these results. This exhibits the limits of pure collaborative filtering and demonstrates the need to use additional sources of information such as item descriptions, user profiles, social networks *etc* in order to make better predictions/recommendations. Also the problem of new items and new users can sometimes be secondary when the amount of data is gigantic. Yet in the problem that we refer to as dynamic recommendation (*e.g.* news recommendation), the problem is a “constant cold start” because the old items die out as new ones arrive. This justifies even more the need for complementary approaches in this context.

1.3.2 Content-based recommendation

In the previous section and in appendix C we studied collaborative filtering, the most popular and well-studied way to build a recommender system. However we have seen that even though CF has many advantages, it suffers from a major limitation: it can not handle the *cold start problem.* This means that CF does not provide any information whatsoever as to how we should start when no data is available. More importantly, even when the system is running, CF does not tell us what to do with new users and new items which is a constant concern in recommendation. Indeed when speaking of cold start, the reader has to keep in mind that we do not only talk about the application launch but about any time when a new item or a new user pops out. Some very interesting work has been done to let us know how to recommend new items in order to get an accurate estimate of their characteristics as quickly as possible [35]. Yet it still takes some time and this "exploration" would possibly deteriorate the user experience. Indeed in a recommender system, it would be desirable to use information on past users and past items to at least have a clue of what is going to happen with a new user and especially with a new item on which information can be available (text data, information from the manufacturer, expert knowledge...). Moreover in some dynamic recommendation domains, such as news recommendation where items come and go, one may never get enough information on any particular item to be able to use a CF approach, hence the need for an alternative solution.

In most cases, side information is available on the users and especially on the items and the other major limitation of pure CF is that it does not allow us to use it. Yet if we know for example that Alice likes fantasy books and that a new fantasy book comes out, recommending it to her may seem reasonable. The information on the user is typically given in a profile that he or she has to fill in. We can also think about other information such as the location, the browser information or the previous search engine queries that Google gives you access to. In general for items such as books, movies or products in general, information such as genre, synopsis, length, *etc* is given in an electronic form by the provider or manufacturer. For textual items such as news, one may want to use the plethora of work on text mining to automatically build a set of features. Note that we usually have more information on the items than on the users hence the name Content-based.

Content-based recommendation is thus simply the name given to all the approaches using this information in addition to the ratings in order to recommend items. As in machine learning in general, the more features we have and the more informative they are, the easier it is to build an accurate predictor. Some characteristics of an item might be easy to acquire. One can think about all the information provided by the manufacturer or knowledge mined from textual content. Yet other very informative features might be harder to extract automatically and accurately *e.g.* the visual aspect of the product, the appeal of the description displayed along with the recommendation, the perceived value for money of a product, the intrinsic quality of a movie or the quality of the promotion campaign and so on and so forth. Adding that kind of information may not be possible and is obviously quite expensive as it requires manual labeling. The same thing goes for user profiles: they may help us but it may also be annoying for a user to fill in his/her profile. The most famous labeling undertaking is the *music genome project* that went on for 5 years. Actual musicians used more than one hundred features to describe thousands of songs. This genome is used by Pandora, a highly personalized web radio that obviously quickly learns the tastes of its users thanks to the huge amount of high quality features at its disposal.

To be more specific, a content-based recommendation algorithm takes a set of features describing items as input in order to predict if a user will like them or not (the outputs are generally implicit ratings). The pure approach that is used for Pandora [36] considers each user individually. Therefore no information on the user is needed, and the conclusions made for one user are not shared to help recommending content to the others. This is usually what is done for music recommendation [4] because the data describing the content is generally very rich. Also in music recommendation each single user tends to expect really personalized content, based on his behavior only. This is possible because users also spend a lot of time listening to music. Thus the system has enough data to make highly personalized and educated recommendations to its users. In such an application this is what is needed as a user in generally aware of what he likes. In other applications, it might be more difficult to identify the user individually. The set of features describing the items might also be less rich. In this context the level of expected accuracy and personalization would be lower. In that case it is possible to use user information to make the task easier. To have more data per user, one can cluster them using their profile information. See the article by Linden *et al.* [29] describing Amazon's recommender system for more insight on the subject. Another idea is to try to predict the affinities of the features describing the items with some attributes of the profiles instead of doing so with each single user [37]. Note that besides the regular demographics and whatever the user is willing to add in his profile, social information have also been used successfully in this context [38]. Note also that these approaches are in fact trying to use the behavior of several user to recommend content to each of them individually. This is the definition of collaborative filtering. Therefore we can almost consider these methods as hybrid. *Almost* because in general such approaches are considered as content-based only. The CF side of what the community refers to as hybrid methods is usually based on classical CF techniques such as nearest neighbors, matrix factorization, *etc.*

In the previous section we mentioned the most commonly used approaches to recommendation using Collaborative Filtering (more details are also given in appendix C). They are quite original and recent as they were introduced in the last twenty years. We will not do so for content-based recommendation. Indeed Content-based recommendation is about overcoming the limitations of CF by learning using features which is in fact what machine learning in general is all about. Reviewing the actual techniques would in fact come down to reviewing supervised learning, unsupervised learning, feature selection, feature engineering and almost the entire scope of machine learning. This is maybe another reason why CF is so dominant in the field of recommendation in academic research. All the content-based approaches can in fact be used in many other domains of machine learning, data mining, *etc.* What is very interesting however is that, reciprocally, content-based recommendation can take advantage of almost all the historical work in those various fields. This is in fact the case as many different approaches from various inspirations can be found in the literature. See Pazzani *et al.* for a very comprehensive literature review [39].

1.3.3 Conclusion

To conclude on the classical approaches to recommendation, let us summarize what we have seen in a few words. Collaborative filtering is the most representative and dedicated approach to recommendation. It has the advantage of requiring little work from the user and of being almost domain-independent since it only uses a matrix of ratings as input. Yet it has a major limitation: the cold start problem. CF needs a lot of data and is unable to handle novelty. To overcome this issue people use what is called content-based recommendation. This is basically a way to use the classical tools from machine learning to perform recommendations. This approach has the shortcoming of requiring some work from possibly both sides (profiles to fill in by the users, item labeling and targeted machine learning by the application owner). There has obviously been work on hybrid systems that try to take the best of both approaches but this goes beyond the scope of this literature review. The ensemble approach that we talked about for the Netflix Prize (section C.3.3) is a simple example of a way to combine two approaches. The reader interested in more details or more subtle methods to build hybrid systems should refer to a survey by Robin Burke [40]. This is a rather old work (2002) but it pictures really well the classical approaches to construct an hybrid recommender system. For a more specific and recent example implemented in the MoRE recommender system, the reader can refer to Lekakos *et al.* [41]. There are a few other things that we did not talk about as they are further away from what want to do here. The classical methods we described are ineffective to recommend items that are not bought very often (cars, houses...). For that kind of domain, people use what is called *knowledge-based* recommendation or *conversational* recommendation in which they try to establish some kind of conversation with the user to help the system or even help the user know what he or she wants (see chapter 4 of *Recommendation: an introduction* [24] for an overview). Nevertheless these kinds of methods apply to different situations and can almost be considered as another domain.

1.4 Dynamic recommendation: (Contextual) bandits

This section presents an online learning approach to the recommendation problem. Even though this approach is becoming more and more popular over the years, it is still not very common. For instance there is no mention of it in *Recommendation: an introduction* [24]. *Recommender systems handbook* [3] has a chapter about active learning which is related but which is not exactly the same thing either. Indeed active learning is about interacting with an environment with the only purpose of improving the system with respect to a given criterion. Thus active learning is only activated once in a while *e.g.* when many new items or users make their apparition. Furthermore it is only tackled in the context of collaborative filtering. The goal of online learning is relatively different. As we detail it in this section, the goal of an online algorithm is to decide on its own when to improve the system (exploration), and when to only use it for its primary purpose, that is provide the best possible recommendation (exploitation).

Therefore since the two major textbooks about recommendation that were respectively published in 2010 and 2011 do not mention this relatively new approach, before discussing it, it is really important to take some time to motivate the need to use it in the context of recommendation. Indeed even in the literature about this approach, it is generally just claimed to be the answer to some limitations of the classical recommendation algorithms without much explanations. Note that this lack of motivation in the literature is probably due the harsh space constraints that are in vigor in conference articles. The first part of this section will thus clarify all this by specifying when this approach is necessary and why. We use many examples of applications to support our claims. Then we will discuss how this problem is usually formalized in the literature and describe a few state-of-the-art solutions. Finally in this section will also be set most of the notations in use in the rest of the document.

1.4.1 Motivation

Let us first consider the problem of a newly online recommender system that has access to no data whatsoever at the beginning. This is the strongest instance of the *cold start problem*, sometimes also referred to as *extreme cold start*. In that case collaborative filtering can be discarded right away for obvious reasons (at least on its own). The classical content-based approaches require a little bit of labeled data (items and/or users with features) in order to train a model. This cannot be done either at the beginning. What we actually want to do is to design an online learning algorithm that is able to decide without prior information which recommendations to make and learn from their outcomes in order to get more and more efficient. These so called outcomes are typically binary values (clicks) but one may also think of real valued outcomes such as ratings or time spent on the page.

The “strict” *cold start problem* that we just described is an important and interesting issue in recommendation. Nonetheless as important as that step can be (if a web application’s success depends on its recommender system, a good start may be crucial) it only happens once in the life of a recommender system. One may thus wonder if it is worth writing a whole thesis on the subject. The present document is the proof that we believe it actually is. Indeed there are applications in which all the items have a limited lifetime and that can be considered to be continuously *cold*. Since the pool of items is constantly changing, we call this type of recommendation task *dynamic recommendation*. These kinds of applications are all over the web. The most well-known example is news recommendation that can be found on most big web portals, newspaper’s website and dedicated news recommendation websites such as Digg.

We can think of other applications such as private sales or auctions in which the users can buy travels, seasonal clothing or some objects put on sale by private owners. A plethora of these websites have popped out these last years such as the European leader *ventesprivees.com*. Video recommendation on *Youtube* for example could also be considered this way as there is constantly new content and a regular user of such an application is not very interested in old videos (although they remain available). Some people finally think of ad display as a kind of dynamic recommendation. We will not defend the idea too vehemently though. Indeed the purpose of classical recommendation is to help the user in its task whereas the purpose of ad display is more to divert him or her from it. In our opinion this significant difference of purpose is important enough to consider that ad display is different from recommendation. Moreover the click rates are also typically much lower even though this may be compensated by the huge number of users they display content to (nowadays ad display companies are in charge of thousands of different websites). Nevertheless the core of ad display share a similar mathematical structure with recommendation (ignoring budget issues, the various bidding processes *etc.*). Consequently it could also benefit from the online approaches we describe here.

Collaborative filtering is obviously not an option for dynamic recommendation but this is actually for such applications that content-based recommendation was introduced. Most of these kinds of approaches use the information given by the manufacturer of the good to build a predictor. In applications such as emails or news recommendation, it is also possible to automatically extract features. Nevertheless the classical content-based approaches still suffer from several limitations. For example in many applications in which the content in not textual such as videos, images or objects for sale, using a content-based approach would require a great effort of manual labeling such as in the music genome project. Moreover this effort would have to be continuous as new items keep coming in. Even when feature extraction is possible from textual description for example, a lot of aspects will always remain hidden from the system such as the perceived value for money of a product, the visual aspect of the featuring picture, the appealing character of the title and/or short description, the amount of advertisement preceding the release of a movie or a video game *etc.* To be more specific even for news recommendation that may seem easier since everything but the featuring picture is textual, the system can not guess today's social, political or economical context that dramatically influence the appeal of people to news information. The list of elements that influence user tastes and that are not easily accessible could actually go on and on. The point is that although it is theoretically possible, it is incredibly difficult to label all these things in order to build a predictor that would take them into account. On the contrary an online learning algorithm could do that for us by learning online the tastes of the audience about each new item *in the environmental context it is released* and considerably reduce the amount of labeling effort.

So far we have considered dynamic item pools which are the main issue but we can also mention that the user pool can generally be considered dynamic. First, the taste of a user can change. In general this is considered to be a slow process but in news recommendation for instance, an important event can dramatically alter how some particular pieces of information are perceived. Moreover it is funny to see that in most of the literature an assumption on the users is made: they can be identified from one visit to another. This is a reasonable thing to assume on applications on which the user has to authenticate so as to use them (although an account could be shared by several people, or a user could have different tastes depending on the context). However this is not the case on many websites such as search engines, web portals or even for e-commerce as some users may tend to log in only when it is required, that is when it is time to pay and thus after their searching process. In fact most of the web applications can be used in both a connected and non-connected fashion. The reader who is familiar with web applications in general is probably thinking about other ways to identify the users such as cookies or IP addresses. Nevertheless both methods have their shortcomings. IP addresses are usually shared by many people (a family, an entire company, a university *etc.*). Moreover one given individual typically uses the Internet from different places and thus with different IP addresses. Cookies could seem like a better option since they are stored on the browser so as to allow the system to differentiate people using the same address. Yet people tend to use several devices (smartphone, tablet, personal laptop, computer at the office *etc.*). Furthermore and this may be the most important issue: a study made by comScore [42] revealed that 31% of American Internet users regularly delete their cookies and that this causes the number of perceived unique users to be multiplied by a factor of 2.5 in average.

Dynamic recommendation's primary application is news recommendation which is mostly done on websites that do not require to be connected. Therefore solutions to dynamic recommendation cannot make the assumption that users can be identified. CF definitely clearly does that assumption as well as many content-based approaches, in particular the highly personalized ones such as Pandora. Note however that some approaches such as the one clustering users using features or computing affinities of items with these same features do not make the assumption. Anyway because the basic idea of context-based recommendation is to use past data to predict the future tastes of users, it naturally struggles to adapt to changes of behavior that are typical of news recommendation. Online learning does not use past data, or only very recent. This is another reason why it should be introduced as a solution to dynamic recommendation.

1.4.2 About bandits

Formalization and specification of the problem

Now that the problem is motivated, let us talk about it more precisely and provide a formal framework to study it. An agent, or an algorithm starts with no knowledge of what users like or not. At each time step, the algorithm faces a dilemma: it basically has to choose between displaying content it believes to be the most interesting (exploitation) and displaying content it has less knowledge about in order to improve its model and thus its future recommendations (exploration). This *trade-off* is known as the *Exploration vs. Exploitation dilemma* and is well studied in the literature. In this section we will review a few solutions to this dilemma that are based on a very simple theoretical problem called *multi armed bandits* and its extension that is more fitted to recommendation: *Contextual bandits.*

One way to model this exploration *vs.* exploitation dilemma is as a reinforcement learning problem [43]. In such a problem an agent faces an environment with different states and actions. At each time step, the agent performs an action which leads to a real valued outcome called a reward and a possible state transition. One of the possible goals is to maximize the sum of rewards. To do so one has to carefully balance between exploration, so as to learn about the environment and exploitation in order to obtain high rewards.

The bandit problem, also known as the multi-armed bandit problem, is the simplest version of reinforcement learning in which there is only one state and thus no transition is necessary. Since recommending a product to one person is not supposed to change the state of the entire world (*i.e.* the tastes of the whole community), this setting suits our needs almost perfectly. Indeed note that it does not model any evolution in the environment such as new items or changes in user tastes. Yet modeling such dynamics would prove quite complicated and require unwanted assumptions that would make us lose generality. Also as we will see it in this work, most (contextual) bandit algorithms are capable of handling new items without being modified. The multi-armed bandit problem was extensively studied in the past ten years but it can be traced back to Robbins and Munro in 1952 [44] and even Thompson in 1933 [45]. There are many variations in the definition of the problem but a basic setting is as follows.

Let us consider a game based on a bandit machine[1] $\mathcal{B}$ that has K multiple arms. At each

[1]A *One-Armed Bandit* or slot machine is a casino gambling machine with three or more reels which spin

time step, the player performs one out of K actions: we denote a_j the action of pulling arm j. When performing action a_j, the player receives a reward drawn from $[0, 1]$ according to a probability distribution ν_j. In the classical setting that we consider, all the ν_j are independent. Let μ_j denote the mean of ν_j and j^* be the arm with maximum expected reward. Let μ^* be the expected reward of that arm ($\mu^* = \mu_{j^*} = \max_j \mu_j$). At the beginning of the game, ν_j, μ_j, j^* and μ^* are unknown. The most common objective (but not the only one) is to maximize the cumulative reward after T consecutive pulls. T is usually called the horizon. In some settings it is assumed to be known. On the web the volume of visitors can be predicted accurately but we will not assume the knowledge of the horizon. Let us denote by j_t the arm pulled at time t and r_t the corresponding reward. More specifically the player aims at maximizing:

$$\sum_{t=1}^{T} r_t.$$

In most of the theoretical analysis and even in the empirical studies in the literature, people are trying to minimize what is called the (cumulative) regret. When an arm j is pulled, the regret corresponds to the *expected* loss that it causes that is: $\mu^* - \mu_j$. The cumulative regret is the expected sum of these losses after T time steps that is:

$$\begin{aligned} & \mathbb{E} \sum_{t=1}^{T} \mu^* - r_t \\ = \quad & T\mu^* - \sum_{t=1}^{T} \mu_{j_t}. \end{aligned}$$

The regret is considered instead of the rewards because its "shape" is more informative. The cumulative reward always end up being more or less linear in the number of steps whereas the regret can have various forms from tending to a constant in the perfect case to being linear if the best arm is never identified. In the exact problem that we defined it can be proved that it is not possible to do better than a logarithmic regret in the number of steps [46]. More precisely, in the case of Bernoulli rewards, it is impossible to achieve a better regret than:

$$ln(T) \sum_{i:\Delta_i>0} \frac{\Delta_i}{KL(\nu_i, \nu*)} \qquad = O\left(ln\left(T\right)\right) \ ,$$

where T is the number of steps, $\Delta_i = \mu * - \mu_i$ and $KL(P, Q)$ is the Kullback-Leibler divergence which is a measure of dissimilarity between the two probability distributions P and Q. The interested reader is referred to Bubeck and Cesa-Bianchi [47] for more details and recent advances on regret bounds. Note that the main state-of-the-art algorithm that we will mention does achieve this minimal regret bound up to a constant. If regret is the usual standard measure, in this document we will only talk about maximizing the expected cumulative reward of an algorithm A that we note G:

$$G_A(T) = \sum_{t=1}^{T} \mathbb{E}\, r_t = \sum_{t=1}^{T} \mu_{j_t}.$$

Indeed if the regret is much nicer for analysis or even plotting, it is not known in practice as if it were, the problem would be solved. This is even more true for contextual bandits that we will discuss in the next section or dynamic bandit problems (*i.e.* possible shifting distributions,

when a button is pushed. Slot machines are also known as one-armed bandits because they were originally operated by a lever on the side of the machine (the arm) instead of a button on the front panel, and because of their ability to leave the gamer penniless (bandit).

arm additions or removals...) since the optimal policy does not consist in repeating one single action over and over again. Therefore since we want to work with real problems for which the optimal policy is unknown, working with the regret is tricky. Note however that maximizing the cumulative reward and minimizing the regret are two strictly equivalent goals.

Solving the problem with upper confidence bounds

To maximize the cumulated reward (or minimize the regret), at each time-step (except the last one), the player faces the aforementioned dilemma:

- either *exploit* by pulling the arm which seems the best according to the estimated values of the parameters (and receive a non-optimal reward if the estimation of parameters was not good enough to identify j^*);
- or *explore* to improve the estimation of the parameters of the probability distribution of an arm by pulling it (and receive a non-optimal reward if the pulled arm is not j^* but increase the chances to identify it for future actions).

None of these options leads to a maximal cumulated reward. An optimal strategy has to define a good trade-off between exploration and exploitation. The most straightforward approach is called epsilon-greedy and consists in choosing the arm with maximum estimated expectation with probability $1-\varepsilon$ and exploring at random with probability ε. This algorithm obviously has a linear regret and it is possible to achieve much better results both in theory and in practice.

The first major breakthrough in that regard was published by Lai and Robbins in 1985 [46] when they proved a lower bound for the regret of a bandit algorithm and proposed a solution achieving a regret of that order in a specific case. Their results were then simplified and extended to more cases (including non-parametric distributions) using an index-based policy by Burnetas and Katehakis [48] in 1996. These results were asymptotic but in 2002 Auer *et al.* [13] made a finite time analysis of index-based algorithms that are optimal (up to a constant) at all time.

This well-known approach to handle the exploration *versus* exploitation trade-off is the *Upper Confidence Bound* (UCB) strategy. At time t, the UCB strategy consists in *optimistically* playing the arm j_t with maximum upper confidence bound on its expectation:

$$j_t = \operatorname*{argmax}_j \hat{\mu}_j + \sqrt{\frac{\alpha \ln t}{n_{j,t}}},$$

where $\hat{\mu}_j$ denotes the current estimation of μ_j after $n_{j,t}$ pulls of arm j before time t. In the proof of optimality α is set to 2 but in practice it is a tunable parameter. For example with Bernoulli rewards, a good choice is to use an α of the same order of magnitude as the average success probability of the best arm. Note that in the end we choose the arm maximizing a simple formula (or index) computed using a few variables. This is why we call such a strategy an index-based algorithm.

Apart from being optimal up to a constant, UCB comes with two nice properties. First, it is fast to compute. Second, it clearly expresses the exploration-exploitation trade-off: while the leftmost term of the sum ($\hat{\mu}_j$) leads to the exploitation of the seemingly optimal arm, the rightmost term of the sum leads to explore arms that the algorithm is less confident about.

Let us discuss the behavior of UCB qualitatively. During preliminary time-steps, the value of the indexes is dominated by the rightmost term: indeed, for each arm, this term quantifies the uncertainty about the expected reward. During the first pulls, we have no knowledge at all, hence a very large uncertainty. After a while, the leftmost terms become predominant and the ones that influence the most the choice of the arm to pull. Thus on one hand the behavior of UCB is very similar to the strategy `Explore-Exploit` which consists in uniformly exploring the arms during a few steps, then focusing on the arm with the best empirical mean. On the other hand, UCB never stops exploring seemingly sub-optimal arms. This behavior ensures that with probability one, UCB will eventually find the best arm even if it is tricked by misleading rewards at the beginning.

Another strength of UCB lies in the following property: the number of pulls of a sub-optimal arm j is of the order $\frac{\ln T}{(\mu^* - \mu_j)^2}$. Hence, the continuous exploration of arm j only costs a loss of the order $\frac{\ln T}{\mu^* - \mu_j}$. As a corollary to that property, the exploration budget is non-uniformly spread among the set of arms. UCB does not lose time, hence rewards, playing arms which are likely to be non-optimal.

In short, while (i) UCB-like algorithms continuously explore to avoid focusing on a sub-optimal arm, (ii) the number of exploration steps is kept reasonable and (iii) the exploration budget is non-uniformly shared among the set of arms (most promising arms are more explored). This (automatically) finely tuned exploration grants UCB algorithms with an optimal performance (up to a constant), whereas `Explore-Exploit` and ε-greedy can be proved to be sub-optimal [13].

Other strategies to deal with bandits

After the work of Auer *et al.* [13] in 2002, a lot of algorithms based on upper confidence bounds have popped out in the literature. For instance *UCB-V*, introduced by Audibert *et al.* [49], is an algorithm that uses variance estimates to sharpen the upper confidence bounds. Their authors proved a better regret bound than the one of UCB. Another improvement is *KL-UCB* which was first introduced in the context of Reinforcement Learning [50]. The idea is to use the Kullback-Leibler divergence (a measure of difference between distributions) to compute the upper confidence bounds. This new algorithm was shown to be better than the state-of-the-art methods both theoretically (regret bound) and empirically on a wide range of problems [51,52]. Note however that since the measure of divergence requires a convex optimization, *KL-UCB* is slightly more computationally intensive than algorithms based on simpler indexes such as *UCB* or *UCB-V*. The original UCB, as well as UCB-V and KL-UCB do not make any assumption about the arm distributions apart from their having a finite support (such as $[0,1]$ as in our formalization of the bandit problem). Nevertheless it is obvious that when more assumptions are made, it is possible to do better. As an example, see Grünewalder *et al.* [53] for an analysis of the case in which the reward distributions of the arms are modeled by Gaussian processes.

UCB and its numerous variants are not the only solution to the problem. We can mention Gittins index [54] which is an old work (1979) and that can be considered optimal in the Bayesian sense (and not only up to a constant) but it is extremely heavy to compute and needs a precise knowledge of the horizon. Note that this technique is the first index-based method to the best of our knowledge. An even older work that we already mentioned is Thompson sampling [45], proposed in 1933. It was very recently proved to be optimal up to a constant in the same way as UCB [55]. Thompson sampling is a very elegant and pure Bayesian way of handling the exploration *versus* exploitation dilemma: at each time step, an action is chosen randomly according to its estimated probability of being the best one. This is quite different from the UCB strategy so let us describe it a little further.

This algorithm is in fact a simplistic yet quite efficient Bayesian algorithm. We assume that the likelihood function of the reward of each arm $P(r|a,\theta)$ is parametrized by a parameter θ (that can be a vector). At each time step the algorithm samples a parameter θ_t from the posterior distribution $P(\theta|h_t)$ which is computed using Bayes rule given the history of past events h_t and a prior distribution $P(\theta)$. The algorithm then pulls the arm with maximum expectation given θ_t. A more elaborate Bayesian algorithm typically makes decisions based on a utility function averaged over the posteriors (see the aforementioned Gittins index [54]).

More specifically, in the Bernoulli case which is well suited for most web applications (click versus non click), Thompson sampling associates to each arm a Beta distributions parametrized by α plus the number of clicks and β plus the number of non clicks. α and β are the parameters of the prior. At each time step the algorithm selects the arm corresponding to the greatest sample drawn from those K Beta distributions. All the details are given in algorithm 2 but as the way we implement bandit algorithms is significantly different from what can be found in the literature, one may want to finish reading section 1.4 before having a look at it.

The reader may have noticed the deterministic nature of UCB. Chapelle and Li [56] mention that this can be problematic when the rewards are delayed which is typical on web severs. Indeed, for technical reasons rewards may get lost, delayed and/or arrive in batch and as long as the model is not updated, UCB will keep recommending the same arm. On the contrary, Thompson Sampling does not suffer from this shortcoming thanks to its randomized exploration. Nevertheless this flexibility comes at a price. Contrarily to what is sometimes written in the literature [56], Thompson sampling is usually outperformed by a well tuned UCB. We will give an empirical example in section 4.4.1.

Finally note that other randomized approaches exist. For instance EXP3 [13] is an algorithm that is based on more or less the same ideas as UCB but with randomized choices. It was introduced for a bandit setting that is slightly more general and difficult than the one we described here and in which an *adversary* may have complete control on the rewards, hence the need for a non-deterministic policy. Auer *et al.* [13] prove in the paper that even though they lose something in terms of theoretical regret because of the randomization, it is the best that can be done in this new setting up to a constant. Yet if it is not why it was designed, it is also a solution to deal with delayed rewards. We will not go into the details of the algorithm since we will not consider this setting in this thesis. Indeed, the independent stochastic assumption is perfectly reasonable in most applications and on the web in particular as the algorithm does not face one adversarial person but a very wide audience.

Remark on dynamic bandit problems

The recommendation tasks that we consider are dynamic. In bandit terms, this means that the reward distribution can evolve. Most of all this means that some arms can be added and/or removed. Lucky for us the two main types of bandit algorithms can deal with dynamic pool of arms without begin modified. Indeed, their exploration strategy is based on the uncertainty on the expected reward of each arm. A new arm will have a big uncertainty and be pulled more than the others when it appears until the algorithm has a similar level of knowledge about it than about the other ones. Dealing with shifting distributions is not really a problem either. For UCB, it is enough to only consider a limited amount of past events in the indexes. In general it is slightly trickier with Thompson sampling because it works thanks to Bayesian updates that are generally approximate. Nevertheless the exact calculation to only consider a limited number of past events is straightforward in the Bernoulli case. For more details about how to handle non Bernoulli shifting distributions in a Bayesian way, the reader is referred to the literature [57].

1.4.3 Contextual bandits

Definition of the problem

The previous formulation of the bandit problem considers a set of K arms or possible actions (items to recommend) associated with stochastic, independent, bounded rewards (clicks for example). The goal is basically to find the best item. This setting is not fit for performing personalized recommendation as the user is not taken into account. We will refer to it as the non-contextual setting and to its solutions as non-contextual algorithms. To fit more accurately the online recommendation problem we described earlier, we use a more general setting defined

Algorithm 1 UCB: a non-contextual, index-based and optimistic algorithm (Auer *et al.* [13]).

```
parameter $\alpha$ : The parameter to tune exploration.

function INIT
    $\forall i \in \{1, K\}, r_i \leftarrow 0, t_i \leftarrow 0$
    $t \leftarrow 1$
end function
function CHOOSE($x \in \mathcal{X}, \{1, K\}$)
    return $\text{argmax}_{i \in \{1,K\}} \frac{r_i}{t_i} + \sqrt{\frac{\alpha \ln t}{t_i}}$
                                    ▷ By convention $\frac{0}{0} = \infty$ and ties are broken arbitrarily.
end function
procedure UPDATE($x \in \mathcal{X}, a \in \{1, K\}, r$)
    $r_a \leftarrow r_a + r; t_a \leftarrow t_a + 1; t \leftarrow t + 1$
end procedure
```

by Langford *et al.* [23] in 2007: the contextual bandit problem. The terminology introduced by Langford is the one in use nowadays. Nonetheless, similar problems were defined and tackled in the past. The oldest one to the best of our knowledge is a problem called associative reinforcement learning Barto *et al.* [58].

The notations introduced here are very similar to the ones in use in Langford *et al.* [23] and will be reused extensively in this document. The Contextual Bandit problem is defined as follows.

Let $\mathcal{X}$ be an arbitrary input space and $\{1..K\}$ be a set of K actions. Let $\mathcal{D}$ be a distribution over tuples $(x, \vec{r})$ with $x \in \mathcal{X}$ and $\vec{r} \in [0,1]^K$ a vector of rewards. The j^{th} component of $\vec{r}$ is the reward associated to action a_j, that is pulling arm j in the context x. In the recommendation problem, this reward corresponds to the reward associated with the recommendation of item j (click, scrolling percentage, explicit rating *etc.*) in a given context. Note that in general the reward only has to be bounded and not necessarily between 0 and 1 (naturally or after normalization). In this document we will only deal with CTR data (*i.e.* with rewards in $\{0, 1\}$ so the word click will often be used to talk about rewards. However unless the contrary is explicitly said, all the results and algorithms presented in this document are valid for bounded, real-valued rewards and not only for binary ones.

A contextual bandit problem $\mathcal{B}$ is an iterated game in which at each round t:

- $(x_t, \vec{r_t})$, where $\vec{r_t}$ is a real valued vector of length K (number of possible actions, is drawn from $\mathcal{D}$.
- x_t is provided to the player (or algorithm).
- The player chooses an action $a_t \in \{1..K\}$ based on x_t and on the knowledge it gathered from the previous rounds of the game.
- The reward $\vec{r_t}[a_t]$, the element of the reward vector corresponding to the chosen action, is revealed to the player whose score is updated.
- The player updates his knowledge based on this new experience.

Therefore a bandit game $\mathcal{B}$ is fully characterized by a couple $\{\mathcal{D}, T\}$ where $\mathcal{D}$ is a bandit distribution that determines the context distribution as well as the K reward distribution, each parametrized by a context x. T is the total number of steps, also called horizon. Note that in theory T may very well be infinite. In practice, $\mathcal{B}$ can be dynamic, that is $\mathcal{D}$ and K can differ from one step to another. In this case $(x_t, \vec{r_t})$ is drawn from $\mathcal{D}_t$ and a_t is chosen from $\{1..K_t\}$. Yet we will not use this setting in theoretical analysis.

Algorithm 2 Thompson sampling in the Bernoulli case: a non-contextual Bayesian algorithm (Thompson [45]).

```
parameter $\alpha$: prior on the number of clicks.
parameter $\beta$: prior on the number of non clicks.
                    ▷ Note that the parameters can differ from one arm to another.

function INIT
    $\forall i \in \{1, K\}, s_i \leftarrow 0, f_i \leftarrow 0$
end function
function CHOOSE($x \in \mathcal{X}, \{1, K\}$)
    for each $i \in \{1, K\}$ do
        draw $\theta_i \sim Beta\left(s_i + \alpha, f_i + \beta\right)$
    end for
    return $\operatorname{argmax}_{i \in \{1,K\}} \theta_i$
end function
procedure UPDATE($x \in \mathcal{X}, a \in \{1, K\}, r \in \{0, 1\}$)
    if r then
        $s_a \leftarrow s_a + 1$
    else
        $f_a \leftarrow f_a + 1$
    end if
end procedure
```

Finally it is important to note the partial observability of this game: the reward is known only for the action performed by the player, not for others. In the recommendation setting, this simply means that we know whether the displayed item has been clicked or not, but we have no information about would have happened had we made another choice.

On bandit algorithm representation

Theoretically we define a Contextual Bandit algorithm A as a function taking as input an ordered list of (x, a, r) triplets (called history) and outputting a policy π. A policy π maps $\mathcal{X}$ to $\{1, K\}$, that is it chooses an action given a context. We do so in order to avoid having to handle objects with an internal state. This is also how bandit algorithms are considered in general in the literature that deals about evaluating them [10,11] because objects with internal states would be very annoying things to handle within mathematical proofs. Algorithm 3 is meant to show how all this works. We will keep these notations without internal state in this thesis when dealing with evaluation methods in order to both be easily comparable with the state of the art and to keep the maths clear.

In practice however in would be highly inefficient to implement a bandit algorithm as a function of history for it would mean that everything is computed from scratch at each step. In real world implementations, an algorithm is an object able to *choose* an action given a context and *update* its model given a tuple (x, a, r). The distinction between those two functions is not usually done in the literature about bandit algorithm design as one choice usually leads to one update. Indeed a bandit algorithm is usually written with everything mixed up: the choice, the update and sometimes even the evaluation. Yet it is important to make this distinction for several reasons. First because it brings a clearer understanding of what is what. Second and mainly because when designing data-driven evaluation methods, which is one important goal of this thesis work, a call to the *update* function may not necessarily follow each call to the *choose* function. We will come to that in the next chapters. We already mentioned the third and last reason. For technical reasons the algorithm might have to choose an action several times in

Algorithm 3 Playing a bandit algorithm against a bandit problem (theoretical notation, without internal state).

```
parameter $\mathcal{B} = \{\mathcal{D}, T\}$: A contextual bandit problem, composed of a bandit distribution and an integer, the horizon.
parameter $A$: A contextual bandit algorithm.

$h_1 \leftarrow \emptyset$
for each $t \in [1..T]$ do
    $\pi_t \leftarrow A(h_t)$
    Draw $(x_t, \vec{r_t}) \sim \mathcal{D}$
    $a_t \leftarrow \pi(x_t)$
    $h_{t+1} \leftarrow h_t + \{(x_t, a_t, \vec{r_t}[a_t])\}$
end for
```

a row before being able to receive the corresponding updates. Furthermore some rewards can also get lost. A separate implementation of those two distinct things that a bandit algorithm has to be able to perform makes everything straightforward when rewards are lost or delayed. Finally we can also consider the fact that a medicine doctor for instance may choose not to do the action suggested by the bandit algorithm. It could happen for various reasons: allergy of the patient, some expertise that was not taken into account by the algorithm, a simple hunch *etc.* Anyway one may want the algorithm not to be troubled by such an event and be able to be updated with an action chosen by an expert. All this is why we will always give the implementation of bandit algorithms as three functions with obvious roles: *init, choose* and *update*. For example algorithm 1 and algorithm 2 are implementations of UCB and Thompson Sampling respectively using this representation. In practice, it is also strange to generate the entire reward vector r_t given that only one element is used. Furthermore in the real world we simply do not have access to the rewards corresponding to the non chosen actions. We use it anyway because although it may seem counter-intuitive, it has significant advantages. First it is the one in used everywhere in the literature and allows as such a faster understanding of our work for a reader familiar with the state of the art. Second it is incredibly convenient in theory for it allows to gather all the randomness of a bandit problem into one single distribution which generally simplifies the notations. Nonetheless, this is clearly not how things are implemented. Also we need in some proofs laid out in this document to distinguish the various distributions generating the different actions in a given context. The use of the classical notations in these specific cases drastically complicates the notations and thus the understanding of the math. This is why we introduce a second, more practical notation that will only be used in specific occasions that will be explicitly signaled to avoid any sort of confusion. Consequently, we define $\mathcal{C}$, the context distribution and $\mathcal{R}_{x,a}$ the reward distribution parametrized by an action and a context. The two notations can be linked very easily:

$$\begin{aligned} & (x, \vec{r} = [r_1, r_2, \dots, r_K]) \sim \mathcal{D} \\ \Leftrightarrow \quad & x \sim \mathcal{C}, \quad r_1 \sim \mathcal{R}_{x,a_1}, \quad r_2 \sim \mathcal{R}_{x,a_2}, \; \dots \;, \quad r_K \sim \mathcal{R}_{x,a_K}. \end{aligned}$$

Notice that this equivalence is only provided to clarify how the two notations are related. Indeed given a context x, it would not make a lot of sense to sample from $\mathcal{R}_{x,a_i}$ for all i (as in the second term) since the purpose of the notation is to be able to do just the opposite.

To make things crystal clear, algorithm 4 shows how those three functions are used when interacting with a contextual bandit problem $\mathcal{B}$ (using both notations) in practice.

Algorithm 4 Playing a bandit algorithm against a bandit problem (practical notation, an algorithm is an object with an internal state and three functions: *init, choose* and *update*).

parameter $\mathcal{B} = \{\mathcal{D}, T\}$: A contextual bandit problem, composed of a bandit distribution and an integer, the horizon.
parameter A: A contextual bandit algorithm providing the three functions that are used below (*init*, *choose* and *update*).

(a) Classical notations: one distribution.

```
$init(T)$ or $init()$        ▷ initializes the internal state of $A$; the horizon is known, or not.
for each $t \in \{1..T\}$ do
    Draw $(x_t, \vec{r}_t) \sim \mathcal{D}$
    $a_t \leftarrow choose(x_t)$
    $update\,(x_t, a_t, \vec{r}_t[a_t])$        ▷ updates the internal state of $A$
end for
```

(b) Alternative notations: separate distributions.

```
$init(T)$ or $init()$
for each $t \in \{1..T\}$ do
    Draw $x_t \sim \mathcal{C}$
    $a_t \leftarrow choose(x_t)$
    Draw $r_t \sim \mathcal{R}_{x_t,a_t}$        ▷ $r_t$ is a scalar, not a vector
    $update\,(x_t, a_t, r_t)$
end for
```

Notations to measure performance

We described a theoretical algorithm A as being a function that maps a history of interactions to a policy. This allows to be mathematically rigorous and is very convenient when writing algorithms. To lighten the notations here we introduce an additional notation:

$$A_t \overset{\text{def}}{=} A(h_t),$$

to represent the current policy of A at time t (h_t is the history at time t).

In our setting which is the most popular one, an algorithm is said optimal with respect to $\mathcal{B}$ when maximizing the expectation of the sum of rewards after T interactions with $\mathcal{D}$:

$$G_A(T, \mathcal{D}) \overset{\text{def}}{=} \mathop{\mathbb{E}}_{\mathcal{D}} \sum_{t=1}^{T} \vec{r}_t[A(x_t)].$$

For convenience, we define the per-trial payoff as the average click rate after T steps:

$$g_A(T, \mathcal{D}) \overset{\text{def}}{=} \frac{G_A(T)}{T}.$$

Note that we may also be interested in evaluating static algorithms, (*i.e.* that always outputs the same policy π). In the context of recommendation, this may mean that one designed a policy using a content-based approach or expert knowledge and wishes to evaluate it. In this case there is no need to model the learning process using the function A. Thus we will use π instead of A when referring to a static bandit algorithm. This will also allow us to make a very clear distinction between when we address the problem of real bandit algorithms and when we deal with static policies. Note that for a policy π, we have that:

$$\forall T, g_\pi(T, \mathcal{D}) = g_\pi(1, \mathcal{D}) \overset{\text{def}}{=} g_\pi(\mathcal{D}).$$

Note also that from this point on, we will simplify the notations by systematically dropping $\mathcal{D}$.

$g_A(T)$ **(resp.** g_π**) is thus the quantity we wish to estimate as the measure of performance of a bandit algorithm (resp. policy).**

In this document we will study and design methods to evaluate this quantity. An estimator of $g_A(T)$ will by denoted by $\hat{g}_A(T)$ (without the T if A is static). We may sometimes also specify the dataset used to compute this estimate if need be ($\hat{g}_A(T, S)$ or $\hat{g}_\pi(S)$ for a static policy). In chapter 3 in particular, we may need to distinguish between different estimators of $g_A(T)$. In this case we will use a subscript specifying the name of the estimator to distinguish them apart:

$$\hat{g}_A^{(method)}(T).$$

LinUCB: the main state-of-the-art contextual bandit algorithm

Finally the contextual bandit problem is theoretically more challenging than the non-contextual bandit problem since the kind of performance bound that can be derived is highly dependent on the nature of $\mathcal{X}$. However one very popular contextual bandit algorithm is LinUCB [7] which is designed to work under some very specific assumptions on $\mathcal{X}$. It assumes that $\mathcal{X} = \mathbb{R}^d$ (no assumption is made on the distribution of the features) and that the reward of an action in some context is given by a linear combination of the features plus some *Gaussian* noise and an action dependent constant. The implementation is given in algorithm 5. Note that it computes a full linear regression at each time step. One may argue that such an approach is not really *online* but this is not the point of this review (we come back to this issue in chapter 2). The implementation given by Li *et al.* [7] has only one parameter which does not allow to tune the regularization of the regression. As this may turn out to be necessary, we added a regularization parameter λ in our implementation.

We can also mention one of the first works on the topic: the epoch-greedy algorithm [23] that is similar to ε-greedy with an ε decreasing over time. In this work Langford *et al.* tackle the exploration *versus* exploitation dilemma by considering epochs of fixed length. Each epoch is divided into n steps of random uniform exploration and $E - n$ exploitation steps, n being chosen according to some criteria that decreases over time. Notice that the logged data from exploration is unbiased: it is not skewed since logged uniformly. This data is then fed into a black box that outputs a hypothesis that predicts rewards given actions and contexts (thus the paper do not deal with modeling reward distributions). This hypothesis is then used during exploitation. This approach has the obvious advantage of being very general: it can be applied to any kind of data as long as a black-box predictor is provided. The problem is that Epoch-greedy does not use the exploitation data because it would bias the predictions. Unfortunately there is not much we can do about that without making assumptions on that data, which is exactly what LinUCB is about. Nonetheless it is always interesting to keep in mind the existence of generic methods that can handle everything.

Remark on the possible actions to perform Typically in Contextual Bandits we consider that we have a limited number of possible actions (items to recommend). Indeed as we have to learn from scratch in a finite number of time steps, the problem would be impossible with millions of items although content-based approaches can be used to simplify the problem (chapter 3 will also exhibit that the problem with a big pool of actions worsens when it comes to evaluating an algorithm offline). However we do not necessarily have to consider that each item, or cluster of items computed in a content-based fashion is an action. A lot of recommendation strategies are now available and the designers of the recommender system may wonder which one to use (User-based CF, Item-based CF, probabilistic tools such as users who visited this page bought that, most popular item within the category and so on). Actually a contextual bandit approach could be used to learn efficiently which complex and preprocessed strategy to

Algorithm 5 LinUCB (Li *et al.* [7]).

```
parameter $\alpha$: The parameter to tune exploration.
parameter $\lambda$: The parameter to tune regularization.

function INIT
    $\forall i \in \{1, K\}, M_i \leftarrow \lambda I_d, v_i \leftarrow 0_{d\times 1}$
                    ▷ $I_d$ = identity matrix, $0_{d\times 1}$ = zero vector of length $d$.
end function
function CHOOSE($x \in \mathcal{X}, \{1, K\}$)
    return $\text{argmax}_{i\in\{1,K\}} \left(M_i^{-1}v_i\right)^T x + \alpha\sqrt{x^T M_i^{-1} x}$
                    ▷ Ties are broken arbitrarily.
end function
procedure UPDATE($x \in \mathcal{X}, a \in \{1, K\}, r$)
    $M_a \leftarrow M_a + xx^T$
    $v_a \leftarrow v_a + rx$
end procedure
```

use depending on some contextual information made of user information, the web-page he is on, the time of the day... This would be a way to considerably reduce the dimensionality of the online learning problem but not only. In the big companies of the Internet, big research teams keep coming up with new recommendation strategies. Using an intelligent algorithm able to deal with a dynamic pool of actions and to perform them contextually would be a good way to limit the impact of a bad new strategy added to the system as the algorithm would quickly stop using it in favor of the old one. Furthermore this would be a way to learn new things about how to ideally recommend content to a user depending on what he is doing.

1.5 Evaluation of recommender systems

This thesis is about contextual bandits and their evaluation using data of past events. From a recommendation point of view, this means that we focus on online learning recommender systems and their evaluation using data. If it was important to review the classical approaches to recommendation in order to motivate what we call dynamic recommendation, the evaluation of classical recommender systems is quite different from the one of the systems we want to evaluate. Indeed, when evaluating a classical recommender system, one wants to assess the quality of a static function that maps users to items. On the contrary for dynamic recommendation one wants to evaluate the ability of the system to make decisions so as to learn as fast as possible, that is the efficiency of its explore *vs.* exploit strategy. In this context of quickly shifting pool of items, evaluating a static function on past data would probably be of little interest. Thus as the classical evaluation methods are not suited for online recommendation we will not spend a lot of time on them. Moreover the main state-of-the-art method for dynamic recommendation that we describe and analyze in chapter 3 and the methods we designed during this thesis work (chapters 3 as well and 5 and the appendices A and B) are very different from what is usually done in recommendation. This is why in this section we only make a shallow review of the methodologies and metrics in use in classical recommendation. Nevertheless this review has its importance in this document for two major reasons. The first one is thoroughness. It would feel like something is missing without a part about recommender system evaluation. Some founding work in the recommendation field has been done on the subject such as the article by Herlocker *et al.* [8] that we discuss below. The second reason is that one of our purposes is to design evaluation methods for a special kind of recommender system. It is interesting to have in mind the methods that are usually used to understand why they are not well suited for online learning algorithms.

Before dealing with the actual methodologies, we also tackle topics that are interesting to consider in order to evaluate any kind of recommender system or even any kind of research. We first consider what is generally meant when talking about evaluating a recommender system, a complete piece of software implementing a possibly complex algorithm. This will allow to specify what *we* mean in this thesis work and what we do not consider. Then we quickly review the general guidelines from the literature to validate some research by experiments, which is exactly what we are after. Indeed what we want is to evaluate an online learning algorithm implemented within a recommender system using experiments based on real data. In general this type of research is called *empirical research*.

1.5.1 System evaluation

Before reviewing any evaluation methodology, what does evaluating a recommender system even mean? A recommender system is actually a complete piece of software that runs on the web, generally within a larger web application. As such, it can be evaluated in many ways. One may wonder how it can deal with load peaks. It could also be interesting to look for security breaches such as the sensitivity to the well-known SQL injections. One of the most important concerns in software evaluation is to assess whether or not the system works as expected and does not fail in extreme cases of utilization. A lot of work has been done on the subject and we can cite one of the reference books by Myers *et al.* [59]. Besides raw software testing, researchers have even considered metrics to measure much more specific attributes such as the maintainability of a piece of software [60]. All these considerations are not specific to recommendation. Moreover software testing is usually considered to be a part of the software engineering process. In this work we will only try to evaluate recommender systems that are assumed to be secure and to work as expected.

Nevertheless, even on well-tested and secure applications, attacks from the outside are possible and one may want to evaluate a system's sensitivity to such attacks. They generally target vulnerabilities of the new intelligent systems on the web that are based on statistics instead of only expert knowledge. A very famous such attack is called *Google bombing* [61]: for instance in 2004, entering the search query "miserable failure" into Google displayed the official White House Biography of George Bush as number one result. The nature of *PageRank* - the algorithm behind Google's search engine - actually makes this very easy. Indeed it was reported that out of over 800 links pointing toward the President's biography, only 32 contained the phrase miserable failure. Recommender systems are known to be sensitive to similar attacks. Most of modern recommendation is based on similarity between users or items based on explicit ratings. The algorithms we described in appendix C were built under the assumption that all the users are benevolent and that their only purpose when interacting with the recommender system is to find interesting content to enhance their use of the application. The typical attacks violates this assumption and aim at either influencing the system into recommending items more often (push attack) or less often (nuke attack). Chapter 9 of *Recommender systems: an introduction* [24] provides an overview of those different attacks as well as some countermeasures that can be taken. These attacks are based on profile injection, that is creating new profiles whose ratings are designed to influence the system. As most recommender systems use nearest neighbors at some point, most attacks require to inject profiles that are similar to existing users. This could be seen as a barrier making the system resilient to attacks. Yet it was shown that it is in fact very easy to use knowledge available to all so as to influence the system using various injection techniques [62]. It is even possible to fool attack-aware recommender systems by only boosting recommendations that would seem plausible for an expert by using market segmentation and again, no more information that the genre of the content which is available online [63]. An evaluation of the effectiveness of those various techniques as well as guidelines to prevent them are provided by Mobasher *et al.* [64]. Classical approaches to counter these attacks are based on supervised learning similar to what is done for spam detection for instance [65]. Yet unsupervised learning techniques can also be applied to detect changes in the distribution of the rating of some items [66]. A more original approach is based on trust relationships within social networks [67]. In this latter work, Massa and Avesani [67] describe a complete and robust recommender system that gives more weight to the ratings of trusted friends in order to both improve prediction accuracy and lower the sensitivity to injection attacks.

If collaborative filtering systems based on explicit ratings can be influenced using various injection techniques, it is interesting to wonder if the same goes for systems based on implicit ratings such as click through rate, time spent viewing an article, navigation chains *etc.* If it appears that injecting users is more difficult (yet not impossible) since it is necessary to simulate their behavior in order to allow them to be seen as similar to others, other methods can be envisioned. This question is discussed at length by Bhaumik *et al.* [68] using among others the example of the clickstream-based recommender system entitled "users who viewed this item also viewed these items" used at Amazon.com. In a nutshell it is not difficult to design clicking robot whose purpose is to associate a page it is trying to push with many other pages. The dynamic recommender systems that we consider in this thesis are based on implicit ratings. Furthermore since they need to be updated online, they are based on very simple statistical models that are very easy to trick. Building online mechanisms to prevent Contextual Bandit algorithms from being manipulated would be very interesting. It would also be very important to make the dynamic recommender system work as expected. A nice point is that detecting deviations in a very simple model is less difficult than in more complex ones such as CF. So if it was done for CF, it is possible for contextual bandits. Nonetheless this is not the purpose of this thesis work. Yet to the best of our knowledge this has never been studied and it would be very interesting to do so in some future work.

To conclude on those various things that can be evaluated to measure how good a recommender system is, it is important to note that they are all important. A good recommender system's implementation should be correctly tested for bugs, breaches, load resistance and so on. Furthermore the algorithm should be designed with the awareness that malevolent people will try to use it at their advantage. As we already mentioned it is also possible to consider the flexibility as an evaluation criterion for online learning algorithms (ability to handle delayed or lost feedback for instance). Yet this is not what we mean in this document when talking about evaluating a recommender system. We will only talk about whether or not it serves its original purpose: does it provide interesting content? Nonetheless there is one additional thing that we will keep in mind that is never optional when releasing a system on the web. We do not want the recommender system, which is generally a side feature of a bigger application to cause additional latency. More specifically, it is commonly admitted that the maximum acceptable latency on the web is 100ms. If we take into account the unavoidable delay due to distant communications, the amount of computational time available is even half this number. Indeed introducing more latency just to perform recommendations would probably cause more harm than good and drive users away. In our context of dynamic recommendation, we generally consider that calling successively the *choose* and *update* functions should never take more than 100ms, hence the need for simple model. A recommender system that needs more time should not be evaluated as bad but as unusable. See the challenge described in chapter 2 for a original time-aware evaluation method.

Remark The 100ms or 50ms barrier should not only be considered as a simple threshold when evaluating a recommender system. Indeed, the less computational resources it requires the better since it will make the system less expensive to run and allow it to go through load peaks more smoothly.

1.5.2 Empirical research

Even knowing precisely what we want to evaluate, that is the qualitative performance of a recommender system, the process is not straightforward. Actually the same thing goes for any piece of empirical research, that is research that is justified by other means than theory. Empirical research has been intensely scrutinized over the years from various perspectives such as philosophy or statistics. *Measurement, design, and analysis: An integrated approach* [69], written in 1991 is one of the most cited book on the subject and gathers good principles, guidelines as well as complete procedures to evaluate research. Chapter 8 of *Recommender Systems Handbook* [3] and chapter 7 of *Recommender Systems: An Introduction* [24] summarize a few of those general guidelines and apply them to recommendation. In the reference book on experimental research we mentioned, an experiment can be defined as follows:

> "In *experimental research design*, an experiment is a study in which at least one variable is manipulated and units are randomly assigned to the different levels or categories of the manipulated variables." (Pedhazur and Schmelkin [69])

There are two kinds of variables: dependent and independent ones. Some variables are independent by nature such as income, gender, personality and are assumed to remain static during all the experiment. Others are independent because they are controlled by design *e.g.* the parameters of a system under evaluation. Dependent variables are supposed to be influenced by the independent variables. The purpose of an experiment is thus to measure that influence by randomly assigning values to one or several selected variables.

Pedhazur and Schmelkin [69] also define *quasi experimental design* in which the subjects of an experiment are not assigned randomly to their treatment, sometimes because it is too costly or impossible to set up. Note that this introduces uncontrollable bias in the measurements. For instance in the context of recommendation, the users (the subjects of the experiment) actually choose whether or not they want to use the recommender system (their treatment). In this context it is not possible to measure without bias the actual impact of the system on the sales for instance. Consequently the results of experiments designed this way are not indisputable and should be interpreted carefully. Note that this thesis work does not require quasi experimental design. Pedhazur and Schmelkin [69] also introduce *non-experimental designs.* Such designs include all the research validated by asking the users about perceived utility of the system, satisfaction and so on via questionnaires. This is out of the scope of this review but it is well studied in areas such as psychology or social sciences. The interested reader is referred to two major books from the literature [70, 71].

Experimental design is a complex subject. We will not make an elaborate presentation of how to do it correctly here as it would be too long and slightly off topic but the interested reader is referred to Pedhazur and Schmelkin [69]. Yet a few criteria that a good piece of empirical research should fulfill are worth mentioning:

- **Reproducibility**: it may be the most important one. The research procedure should be systematically and thoroughly documented so that it can be verified and repeated. Such a process allows others to easily correct or improve the results and brings immediate trust.

- **Validity**: once an experiment has been designed, it is always important to wonder whether or not it actually measures what we are interested in. For instance if we measure that in general people have more chance to die when they are in a hospital than at home, can we conclude that hospitals do not provide an adequate treatment? The answer is most certainly *no* for there is an experimental flaw. The fact of being in a hospital or not is strongly correlated with one's health condition. To evaluate a recommender system, one may measure the average revenue generated by consumers interacting with it compared to the average revenue entailed by others. Here the experimental flaw, if there is one, is far less obvious. Nevertheless there is one. It is not possible to know the reason why people decide to use or not the recommender system. It is very likely that people decide to use the recommender system precisely because they want to buy something, which brings an uncontrollable bias in the experiment. Questions of the same range arise very often when evaluating recommendation such as: is it valid to consider that clicks of users on a recommended item to access its detailed description somehow measure their opinion about this item?

- **Reliability**: simply refers for instance to whether or not the results could be due to errors of measurement within the data.

- **Sensitivity awareness**: requires that it might be possible to observe different evaluations in similar conditions. One may want to measure the sensitivity of the results of the research when some parameters evolve. Yet the most important consideration is *statistical significance* of the results. Many statistical tools exist [69] such as variance (ANOVA in particular), tests of hypothesis *etc.* Moreover, once we can assess of the statistical significance of some research, it is equally important to wonder about its impact on real world scenarios. For example, what impact does a 1% improvement of the rating prediction, although statistically significant, have on the quality of the recommendations?

1.5.3 Methodology

In the literature, three major ways of evaluating recommender systems are reported: offline/data-driven evaluation, online experiments and user studies. We will focus on the first one here as it is what this thesis work is about although in the context of dynamic recommendation. The latter mainly consists in inviting a representative set of users to a lab, letting them interact with the system in the way that is of interest to us and questioning them afterward. Again this kind of approach is not within the scope of this review so the reader is referred to other works for more details [70, 71].

Online experiments are not the subject of this work either and are only accessible to the owners of the web application being evaluated. Nonetheless one widespread practice is interesting to talk about. We mentioned in the previous section that only trying to evaluate a recommender system by putting it online is naturally biased because each user chooses by himself to use it or not. A common method in marketing to overcome this issue is called *A/B testing* [72]. It is also relatively well known in web oriented data mining. When one has to decide between several versions of a recommender system for instance, the idea is to assign each visiting user to a randomly chosen version of the system. To measure the very impact of the recommender system on some external variables such as sale volume or the session length it is possible to randomly decide if the user simply gets to have access to the recommender system or not. This procedure turns a quasi experimental design in which there is some uncontrollable bias in an experimental design, free from this bias. Note however that each independent variable that needs to be tuned requires a different A/B testing design.

Offline evaluation based on past data is what we aim at in this thesis in order to, among other things, reduce engineering costs and business risks implied by recurrent online tests of new versions of a recommender system. Furthermore since it is the only possible evaluation method for most researchers, it is also the most commonly used. For instance out of 12 empirical research papers about recommendation in ACM Transactions on Information Systems (TOIS) between 2004 and 2008, 9 were validated by offline evaluation techniques, 2 via online experiments and 1 by a qualitative user study (see table 7.3 in *Recommendation: an introduction* [24]). It is important to keep in mind that when experimenting offline with datasets the subjects (logged user interactions) do not need to be assigned randomly to different treatments (prediction algorithms for instance). It is enough to reuse the same data for each variant of the experiment to compare the results. Indeed although sequentially assigning real users to different treatments would lead to biased measurements (since user might remember their previous actions, have found what they are looking for or simply be bored), doing so offline is not a problem as everything obviously remains static and independent.

Nevertheless offline evaluation suffers from one major drawback. It is not possible to know more than what is inside the data. For example in the context of matrices of ratings, it means that it is not possible to know for sure if a given user would like a recommended item if the data does not contain the rating. As a consequence it is very complicated to measure offline the quality of a recommender system as it is done online. This is why in the literature, the results of online and offline experiments are never really compared. So what does an offline experiment consist in and how is it useful? Actually they mostly consist in prediction tasks. For instance the prediction task defined in the Netflix prize (see app. C.3)that we already mentioned is accepted by the community to be meaningful as far as the quality of a recommender system is concerned. As a reminder, this task simply consisted in predicting some ratings that were randomly hidden from the dataset using the remaining ones.

In practice this is generally how things are done to ensure that the results will *generalize* to the real application. The dataset is split randomly into a training set and a test set. The training set is naturally used to *train* a model. Then this model is *tested* by measuring its prediction accuracy on the test set using some metric, root mean square error (RMSE) in the Netflix Prize. To gain reliability and to make sure that the measurements are not biased by some user profiles, it is possible to repeat the entire process several times. This procedure called *N-fold cross validation.* It consists in dividing the dataset in N parts of size $\frac{1}{N}$. Each "part" is used to test a model that is trained on the remaining $\frac{N-1}{N}$ records of the dataset. As a consequence each user is used once and only once to test the prediction model. This procedure, similarly to *bootstrapping* in statistics [14], allows to get a confidence interval on the prediction accuracy of the model in generalization, that is on another dataset from the same application. See Kohavi *et al.* [73] for more properties of N-fold cross validation.

On one hand this type of data-driven evaluation has one major advantage. Indeed the community is lucky enough to have access to numerous public evaluation benchmarks that come from major web applications. Many of them are movie rating datasets (Netflix, MovieLens, EachMovie...) but we can also mention Jester, a dataset of joke ratings and BX, a dataset of book ratings. Most of the work on recommendation is evaluated on those publicly available dataset making the results both repeatable and comparable. Moreover it is well accepted in the community that good prediction accuracy generally leads to good recommendations. On the other hand this simplicity also comes with a few drawbacks. First it is common knowledge that even when using test sets or cross validation, it is possible to overfit the data. Indeed with public datasets, people typically have an iterative approach in order to improve their prediction algorithms. Even if some part of the dataset is always kept apart for testing, this sequential process is in fact a manual learning process that naturally leads to overfitting the dataset. This problem is not only present in recommendation but in supervised learning in general. Two references on the subject by Salzberg [74] and Babyak [75] are very interesting to read. The first one gives precise guidelines to limit this phenomenon with a focus on classification whereas the second one is more general and much less technical, being meant for a broader audience. The challenge we organized and that we talk about in chapter 4 is also a perfect example of the bad generalization properties of an algorithm designed via sequential improvements of the evaluation on a specific dataset. We mentioned the second drawback of the simplicity of this evaluation method in section 1.3.1 (it is presented with more details in appendix C.3.2). It is crucial to keep in mind that this method only evaluates rating prediction which is only a part of the recommendation process. Moreover it tells nothing about how the recommender system would actually perform in terms of user satisfaction or sale volume which are the true relevant metrics. Even though everyone in the community is aware of that, the fact that it is very easy to set up made it very popular. So popular that the other parts of the recommendation process are often left aside. Furthermore one may also presume that most of the recommendation algorithms are biased towards good performance on the few most popular public datasets.

Finally, even though the supervised rating prediction method we described is the most popular there has been work to try to evaluate offline recommender systems as a whole. Those approaches are based on user models that are generally relatively complex and may be learned from datasets of real user interactions [76]. User modeling is a complicated task but there is a vast amount of research on the subject (see Fischer [77]) for instance). However the results of such evaluations should not be trusted too much as a model is never perfect and an algorithm solely optimized based on such evaluations is not likely to generalize so well to the real world. Nevertheless those simulated online evaluations can be very useful to have an idea of the impact of a lift in prediction accuracy on the quality of the recommendations or for other profiling purposes. Note that evaluation considerations set aside, user modeling can also be a powerful tool to improve recommendations [78].

1.5.4 Metrics

So far we have only discussed one offline evaluation metric for recommender systems: the root mean square error (RMSE) on predicted ratings. This historical metric is still the most commonly used along with mean absolute error (MAE) (see table 7.3 in *Recommendation: an introduction* [24]) also applied to prediction accuracy. Although it is accepted by the community to be meaningful when evaluating a recommender system, it has weaknesses. A paper by Herlocker *et al.*, published in 2004 [8], is the authority in the field of the evaluation of CF-based recommender system. This article points out the strengths and weaknesses of recommendation evaluation through rating prediction accuracy metrics. It also provides many reasons why other metrics should be used in complement or sometimes even in replacement. Finally other metrics are proposed to measure different qualities of a recommender system. Another very interesting survey about recommender system evaluation was published more recently (2009) by Gunawardana and Shani [79].

The main strength of rating prediction is probably its simplicity and generality. Despite that, it is a good indicator of a recommender system's performance, which can even be demonstrated analytically [80]. Yet it has numerous weaknesses [8, 81]. The main one is that, as an indirect method, it tells us nothing about the metrics we are interested in such as the CTR or the sale volume. This topic is very interesting as far as the context of our work is concerned but because this is not directly related either, we make a much more thorough review of these weaknesses and provide references for further reading in appendix D. In this appendix we also describe alternative metrics that can be used in complement or even in replacement of rating prediction based on Herlocker *et al.* [8]. For instance we mention rating prediction metrics that allow to give more weight to ratings that are more important to predict well [82]. We also talk about classification metrics that reflect the user experience better than RMSE or raking metrics that allow to consider more than one recommendation at the same time. Note however that all these metrics remain indirect methods of measurement.

1.5.5 Dynamic recommendation evaluation

Rating prediction tasks as well as the other evaluation methodologies that we mentioned and are described in appendix D make the implicit yet rather obvious assumption that the user behavior does not change over time. They also make the implicit assumption that the pool of recommendable items does not change, or at least not too much. We motivated in-depth in section 1.4.1 that this is not always true. In some applications that we called *dynamic recommendation* the pool of items is constantly changing. The way to tackle this issue in the literature is named *content-based* filtering. Evaluating such a recommender system can be done in a similar way as what we just described by replacing the item identifiers by sets of features. Again in section 1.4.1 we exhibited that such an approach only works if one can assume that the preference of any user about an item can be fully explained by its features. It may be almost true for the *Music Genome Project* where songs were labeled with great care by experts. However in applications such as news recommendation, new items come up really frequently. Moreover if some features could be easily built with the right piece of Machine Learning (topic, location *etc.*) it seems very difficult and very expensive to derive features describing the full complexity of why a user might or might not be interested in a piece of news, clothing or any recommendable item. For example the appeal of the featuring image or the appeal of the summary to different users are important considerations that would be extremely hard to characterize *a priori*.

This is why online learning approaches to recommendation have been introduced in the past few years [6, 7, 23, 83]. In this context the classical evaluation methods can only be used to measure the quality of the *model* an online learning algorithm is trying to learn. By doing this we miss to evaluate a crucial component of the algorithm which is its ability to learn efficiently and handle the exploration *vs.* exploitation dilemma, hence the need for new methods. This is in great part what this thesis work is about. The literature on the subject in quite poor, hence the significance of this work. Indeed the problem is not trivial: a dataset is composed of triplets (context/user, action/item, reward/rating). Therefore simulating a learning process is not easy since when facing a context, we are only aware of what happened when the logged action was performed. What to do if the learning algorithm we evaluate decides to perform another action? Yet when taking the problem into perspective, two basic types of solutions come to mind.

1. **Indirect methods:** Since evaluating exactly what we want is hard, with the example of classical recommendation in mind, one may straightforwardly think to engineer a method that measures something different, more accessible but significant in some way with respect to our goal. In chapter 2 we present one such indirect method. Note that this is exactly what rating prediction tasks are to classical recommendation.

2. **Direct methods:** Obviously it is always interesting to design methods that directly measure what we are after, that is the averaged cumulated reward. Yet the very existence of indirect methods proves that this is not always easy.

There already exists one of the latter methods for contextual bandits. The first paper about it was published in 2008 by Langford *et al.* [10] although with a slightly different formulation as the one we will consider. Indeed they consider a setting with non learning algorithms (stationary policies) for which they provide a convergence result. This method was really made popular by an article published by Li *et al.* in 2011 [11] mainly because of their unbiasedness result for any learning algorithm. A review of this method is provided at the very beginning of chapter 3. In fact it would be fair to consider that the work by Langford *et al.* [10] and the one by Li *et al.* [11] are the foundations of the two main sets of contributions presented in this documents. Indeed, the first set is a deeper theoretical and then empirical analysis of this method that we refer to as *replay*. The second set deals with shortcomings of this method that we exhibited and adds extra features such as the possibility to output confidence intervals in addition to raw estimations in order to reinforce reliability.

Before concluding this literature review, let us mention one last detail. In this work we consider that dynamic recommendation can be formalized as a contextual bandit problem. In a contextual bandit problem the goal is to maximize a sum of outcomes or rewards. This means that a good evaluation procedure should output something that reflects this quantity. This also means that each recommendation has to be associated with a real valued outcome. In general the explicit ratings that we mainly considered so far are not systematic, that is they are not made after each recommendation. This is why in dynamic recommendation we need to use implicit ratings such as clicks on the recommended item. We can go further and consider for instance the time spent viewing the detailed description after clicking on the item, the percentage of that page that was scrolled, a score based on mouse movement *etc.* Note that much more possibilities to design implicit ratings are described in the literature [84] to provide more informative rewards to a recommender system. In this work we will mostly consider clicks. Nonetheless we will keep in mind that other kinds of rewards are possible. Most of our work is valid for real valued rewards in general anyway. Finally this formalization makes an assumption that could be discussed for a long time. It assumes that the average reward that the recommender system gets is a good metric to evaluate the quality of its recommendations. Some may argue that this is not completely true since a user may have a more complicated behavior than *I like so I click*. Yet we will assume that this is true for two reasons. First it is well accepted in the research community and also within commercial applications. Indeed, in the industry the click through rate is very often considered as a major indicator of performance. Second because we do believe than even if it is not optimal, it is close enough to justify this work. Indeed, as long as a real valued reward reflecting the goal we are interested in can be designed, contextual bandits remain a valid formalization for our problem. We already mentioned a few possible reward designs but note that there are a lot of promising solutions. The reader interested in a discussion on different metrics that are more suited than the CTR to evaluate the relevance of the recommended content is referred to Zheng *et al.* [85] or Claypool *et al.* [84].

1.6 Conclusion

Of all the different approaches to recommendation, collaborative filtering is the best researched one and the most used in practice. This can be explained by the fact that CF is both efficient and generic. Indeed one only needs users, items and ratings to implement a CF-based recommender system. If ratings are not present in every applications, implicit ones can be computed using user interactions with the system. The simple yet effective idea behind CF is to use similarities between users and/or items to predict unknown ratings. Nonetheless it faces two main limitations.

1. CF requires a substantial amount of data to work correctly and it is not capable of dealing efficiently with new items and users.

2. Generic approaches are never optimal by themselves. In the particular case of recommendation, it is extremely rare to only have access to user identifiers, item identifiers and ratings. CF is a generic approach that only uses that. On the web plethora of additional information is available (user profiles, item descriptions, social networks *etc.*). It is a shame not to use all this additional data.

In some applications such as movie recommendation, CF already achieves a lot in spite of its genericity. Indeed, movies do not expire. People do not change their mind about a movie they see and they generally see a lot of them. Consequently a lot of accurate data can be fed into a collaborative filtering based system. In other applications such as news recommendation, that we refer to as dynamic recommendation, things are very different. User tastes evolve a lot and items are only available for a few hours. In the classical literature about recommendation, such applications are faced with an approach called content-based filtering. These kinds of approaches are based on pieces of machine learning that predict user preferences using information about the items. Yet part of this chapter was about motivating the need for different approaches. Indeed very often the information directly available (from the manufacturer, using text mining *etc.*) is not enough to design an accurate system. Manual labeling is then the only option to keep using classical tools. The problem is that it can be very expensive: 5 years by professional musicians for the music genome project which is the basis of Pandora, a personalized radio station. It is even more problematic for what we call dynamic recommendation, that is when items come and go such as in news recommendation or online private auctions. Indeed a constant labeling effort would be necessary.

Another alternative to manual labeling is to learn online the characteristics of each item. In addition to avoiding the labeling effort, we also avoid being fooled by mislabeled or incompletely labeled items. To study this problem formally, we use the contextual bandit framework which was well studied in the literature. Nevertheless most of the work focuses on a sub-problem - the multi-armed bandit problem - that does not allow to do personalized recommendations, which is annoying when talking about recommendation. The more general problem allows personalization via what is called contextual information. Yet it is more tricky to analyze theoretically as assumptions are necessary on the information that is used. This is why the literature on that subject is much less dense although it is much more interesting in practice. This is the starting point of this thesis work whose primary aim was to study new approaches to that problem. Our first contribution (besides the motivation of online learning in the context of recommendation) is about that and is detailed in the next chapter.

Nevertheless the main topic of interest of this work progressively shifted to data-driven evaluation methods. This is why this chapter also reviewed the methods to evaluate recommendation offline. This review had two major purposes. First it is important to have in mind what it means to evaluate some piece of research and in particular a recommender system. Second the review clearly exhibits that the approaches to evaluate the classical recommendation algorithms are particularly ill-suited to evaluate online learning algorithms, hence the need for new ones. Note that such a new method was designed in 2008 [10] and made popular in 2011 [11]. This method that we call *the replay method* was not reviewed in this chapter. A brief review will be done in a following chapter along with a much deeper theoretical analysis than what was done in the literature. These two articles are the starting points of the two main set of contributions of this thesis work.

1. The first set is an analysis of the *replay method* that fills the gaps left in the literature and an empirical study in which a great care is put to ensure reproducibility, validity, reliability and a great sensitivity awareness. Furthermore, the scale of one of the experiments, a challenge we organize, is unprecedented in the context of both contextual bandits and dynamic recommendation.

2. The second set consists in two new evaluation methods meant to overcome a problem that is empirically highlighted in the challenge we organize: the *time acceleration phenomenon.* The first method lays strong theoretical foundations based on bootstrapping theory. The second is much more practical and allows to control risk with a novel approach: *entangled validation.*

Chapter 2

The Exploration *versus* Exploitation challenge 2 (ICML 2011): the winning algorithm and a true ”online” alternative to LinUCB

Abstract of the chapter *In this chapter we describe the evaluation method used in the Exploration vs. Exploitation challenge 2 organized by the University College of London jointly with ICML 2011. To win this challenge we modified a state-of-the-art algorithm: Ad Predictor which we also provided with additional possibilities. This algorithm makes assumptions that are quite similar to the ones made by LinUCB, the most well-known contextual bandit algorithm but is made to use discrete features (although we show how to relax this constraint). After deriving a new algorithm, LAP, able to deal with the classical contextual bandit problem, we show that this new algorithm exhibits a performance that is comparable with that of the state-of-the-art solution LinUCB while being much more computationally efficient: linear versus cubic time complexity. We also derive a generalized version of this algorithm, GAP, which is able to deal with a much wider range of types of data that can be met in real applications than LinUCB.*

Keywords Recommendation - Challenge - Bayesian - Exploration - Exploitation - Online approximation - Time complexity.

Contents

2.1 Introduction

This chapter is the story of how we won the *Exploration vs. Exploitation challenge 2*, organized by the University College of London (UCL) and whose results where presented in a workshop held at the International Conference in Machine Learning (ICML) 2011. After having read the introduction to this thesis, the reader may wonder why this chapter comes first or even comes at all. This first work we present is also the first work we did chronologically. Yet, this is hardly reason enough. This challenge can in fact be considered as some huge experiment to evaluate online recommendation. Even though we did not take part in the design of the evaluation method which we describe further in this document, our participation to such a challenge and its uncanny outcome made us realize how tricky it was to evaluate online learning algorithms in the context of recommendation. In fact this work is the real starting point of this thesis since wondering how we could improve this particular challenge or the evaluation of dynamic recommendation in general came as the most natural question after realizing how we had won this challenge. This question was even raised at the end of an article of ours describing our approach to win this challenge published in JMLR W&CP [5]. Alongside with this interrogation, we were already proposing a new evaluation method based on classification. This method will however only be discussed in appendix B, and so for two reasons. First this chapter only deals with learning algorithms for both the contextual bandit problem and the precise task defined by the organizers of this challenge. Second we do believe this new method to be interesting but not to solve the issues that shall be highlighted in this chapter and in the chapters that follows. Nonetheless we apply that method as a diagnostic tool in chapter 4, demonstrating its usefulness.

The Exploration *vs.* Exploitation challenge 2 considers the problem of predicting clicks of users (that are not logged in so we may simply refer to them as visitors) on items presented on a website, based on data of past interactions. The challenge relies on data provided by Adobe. The dataset represents a website activity regarding the interactions of the visitors with some recommended items. The items correspond to articles displayed in a small box on the visited web page with a summary to attract the user as he or she visits the website. The aim is to propose interesting content to the users; their interest for a given article is only asserted by a click, which then provides the user with the full content of the article. Once the article is displayed, the user is simply assumed to be satisfied. So the goal is to assert whether or not we are able to make users read the articles regardless of their actual interest in the full content. This is in fact exactly the idea that is behind performance metrics such as the CTR in classical recommendation or even in advertisement that we introduced in section C. We actually simply assume the summary to be representative enough of what is inside the article to make the metric relevant of user satisfaction.

To use the terms we introduced in chapter 1 the task proposed in this challenge is an *indirect* method of evaluation of a dynamic recommender system. Indeed a different online learning task, that can be easily evaluated using the data is designed. Consequently the interest of participating to this challenge is twofold. First it is an exciting opportunity to design an online algorithm which very close to the contextual bandit algorithms we want to study and to be evaluated wit data from a real application. As we mentioned it in section 1.5.5, this is a rare opportunity when working on bandits. This is actually the main reason why we entered this competition but there is another interest in this challenge, especially considering the topic of this thesis. Participating in this challenge is a chance to study an indirect method of evaluation for dynamic recommendation and in particular the inherent bias of such methods.

As mentioned earlier this chapter narrates how we won the challenge we just described but that is not all. We do start by talking about the challenge and how we managed to win it. More precisely, section 2.2 models the task mathematically with notations corresponding to the ones introduced in chapter 1. Section 2.3 is about the main ideas of the approach we used, inspired by a work by Herbrich *et al.* [86] and Graepel *et al.* [6]. Finally section 2.4 goes into the details of a lot of very interesting tweaks we came up with along the way to improve our performance.

In a nutshell, our approach is based on an additive model updated at each time step in a Bayesian way. Most of the improvements that we were able to make on the original approach were obtained thanks to our continuous work on feature construction, and, more surprisingly, by a continuous simplification of the model. Indeed, we ended up making our predictions ignoring completely the displayed item. This may be surprising but we explain in this chapter how this was made possible by the evaluation procedure. More importantly for a so-called "Exploration *vs.* Exploitation" challenge, all our attempts to take into account any underlying dynamics or to explore with care also led to a decrease of the performance.

After talking about our algorithm we discuss the evaluation method. In section 2.5 we mainly try to explain the reasons why simplicity was the best option in this challenge . Furthermore we emphasize that it may not be the case in real recommendation systems. In this at least, the evaluation method did not serve its purpose very well. The bias toward simpler algorithms is not a great concern in itself though. The main concern is actually that simplicity to the extreme was so efficient, that we ended up ignoring the items which is preposterous in the context of recommendation but possible with this indirect evaluation method.

We then forget a little bit about the challenge and set our focus on our algorithms only. Indeed, although the task of the challenge ended up being disappointing as far as the evaluation of recommendation is concerned, the algorithm we designed did beat many participants and performed really well on a real online learning task based on real data. Consequently it is really worth studying it a little further. Therefore since we are interested in contextual bandit algorithms, we propose two contextual bandit algorithms based on our approach: GAP and LAP. We exhibit how both algorithms improve the state of the art. LAP is a special case of GAP and can only be used with real-valued, linear features exactly as the main state-of-the-art algorithm: LinUCB [7]. LinUCB is very computationally intensive but we show that LAP is faster by several orders of magnitudes (linear *versus* cubic complexity). We also justify using experiments than despite this complexity gain, LAP exhibits performances that are very close to LinUCB's and sometimes even slightly better. Finally the strength of GAP is the fact that contrarily to LinUCB and LAP, it can use a combination of a much wider variety of features:

linear, discrete but also any function of an original feature without any foreknowledge of that function.

2.2 The task to accomplish in the challenge

The task proposed in the Exploration *vs.* Exploitation 2 is an indirect method of evaluation for dynamic recommender systems. To the best of our knowledge none existed before and this particular one had never been tested before, hence the appeal of this challenge. This task is very simple but before that, let us say a word about the data.

2.2.1 Dataset and notations

As one may expect, the dataset S is a chronological sequence of interactions between users and recommended items. Each interaction is logged as a triplet (user, item, click), the "click" being a 0/1-valued implicit rating, or (context, action, reward) to use the contextual bandit terminology. We refer to a couple (user, item) as a *display*, a term introduced by Graepel *et al.* [6]. Here, the word context has a lot of sense because a user is characterized by a feature vector (described in what follows) but we do not have access to a unique identifier which is typical of dynamic recommendation. Furthermore, nothing ensures that users described by the exact vector of features (context) are the same person. Reciprocally, one particular user may be described by different features depending on various contextual elements such as the web page he or she is on, his or her location, the time of the day *etc.* On the contrary, an item is simply characterized by an ID. Finally the reward is binary and reflects whether or not the user clicked on the presented item.

Let us now define the necessary notations to describe both the challenge and our approach to win it.

A user (or context) is a point in a space $\mathcal{C}_c \times \mathcal{C}_D$ where:

- $\mathcal{C}_c$ is compact subspace of $\mathbb{R}^{99}$ that we may also refer to as the continuous features,
- $\mathcal{C}_D$ is a discrete space of dimension 20, referred to as the discrete features.

Notice the use of the letter $\mathcal{C}$, similarly to what was introduced in section 1.4.3 for the context set. Here we divide it into two subspaces for we need that distinction in what follows. To keep consistency with the rest of this thesis work, a user (or context) will be denoted by the letter x. An item, whose recommendation is an action in contextual bandit terminology, will be denoted by the letter a. Yet due to the evaluation method we will mainly considers displays, denoted d, that gather a user and an item.

Contrarily to the users, the items are not characterized by anything but an identifier. The item space is thus denoted by $\{1..., K\}$, where K is the number of items in the dataset. In the challenge, $K = 6$. Finally a reward lies in $\{0, 1\}$ and is denoted by the letter r.

2.2.2 The evaluation method

The evaluation method is illustrated on figure 2.1. In a nutshell, the idea is to divide the dataset sequentially into batches b of L consecutive interactions and to hide the rewards. A batch is therefore composed of L displays (user-item couples: (x, a)). The detailed procedure to build them is the first part of algorithm 6. To avoid any confusion note that $L = K = 6$ but the equality is not a requirement at all. Indeed even with $K = L$, nothing ensures the items within a batch to be all distinct from one another.

The batches are then proposed one at a time and in chronological order to a player. Here we consider the behavior of a player to be automated within an algorithm A. At iteration t,

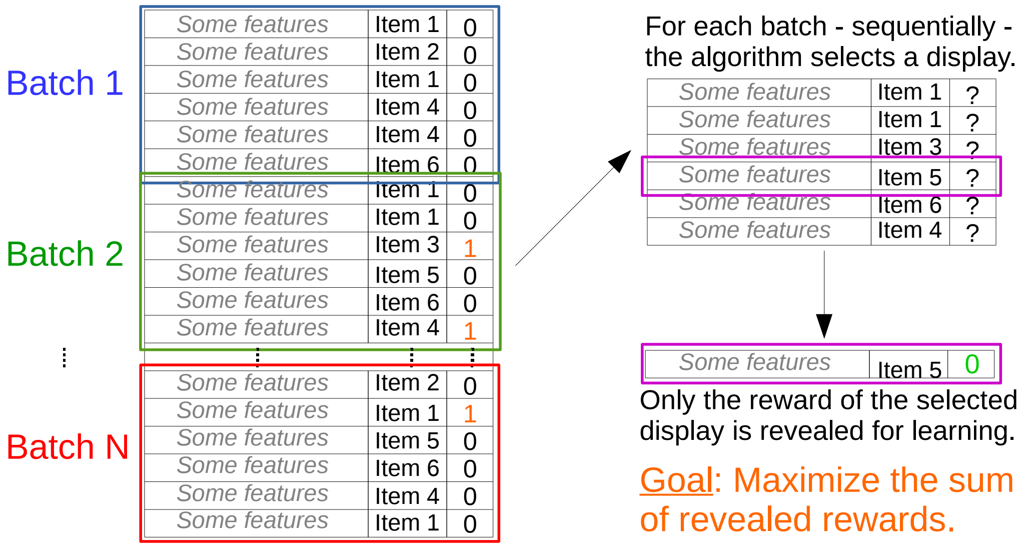

Batch 1
Batch 2
Batch N
Some features
Item 1
For each batch - sequentially - the algorithm selects a display.
Only the reward of the selected display is revealed for learning.
Goal: Maximize the sum of revealed rewards.

Figure 2.1: Graphical explanation of the evaluation method of the challenge.

the evaluation method proposes a batch b_t to $\mathcal{A}$. $\mathcal{A}$ then chooses the display inside b_t which it considers the most likely to be associated with a reward of 1 (a click). The reward of this element (and only this one) is then revealed to $\mathcal{A}$ and added to the current score. The goal is to maximize the sum of the revealed rewards, that is exactly the same goal as in a contextual bandit problem. To do so, the player is given no prior knowledge. Rather he or she is expected to learn online how to recognize clicks when facing a user to whom an item is displayed (a display). Note that as a consequence, since the player can only choose between logged displays, the problem is no longer partially labeled which the whole point of this method. The organizers of the challenge (Glowacka, Dorard and Shawe-Taylor) expected this task to make the player focus on finding correlations between user features and the displayed items rather than simply identifying the best item, which a bandit algorithm would do in priority in a bandit problem. In other words their *a priori* idea was that restricting the ability of the player to simply make non personalized recommendations would force him or her to learn faster how to make personalized recommendations which is exactly what we are after in the end.

Since the task is different, an algorithm, to compete in this challenge has to implement functions that are not exactly the same ones we defined for contextual bandit algorithms in the previous chapter. Such an algorithm is however very similar in the sense that it has to define three procedures.

1. *init* is called before the evaluation process.

2. *chooseDisplay* selects a display within a batch of L displays. In contrast, the *choose* function of a contextual bandit algorithm chooses an action or an item in a list of available actions, given a user/context, hence the different denomination in use here.

3. *update* lets the model being updated by a display and its corresponding reward (a full interaction as it is logged in the dataset S). This procedure is similar to the *update* procedure we defined for the contextual bandit algorithms. We thus kept the same denomination.

To implement an approach to the challenge in this chapter, we will simply provide the pseudo-code of these three procedures. To make things perfectly clear, we also provide the pseudo-code of the evaluation procedure based on these three procedures (algorithm 6).

Algorithm 6 Evaluation of an algorithm in both phases of the challenge. Note that such an algorithm, although it may look similar, is not a contextual bandit algorithm as defined in section 1.4.3. The *update* function is the same but the *choose* function is different, hence the different name.

parameter $\mathcal{A}$: an algorithm implementing the 3 necessary procedures.
parameter S: a dataset of T triplets (x, a, r) ordered chronologically with T a multiple of L, the size of a batch.

```
/* Construction of the batches. */
$\forall t \in \{1, 2, ..., T\}, d_t = (x_t, a_t)$
for each $t \in \{1,\ L+1,\ 2L+1,\ 3L+1,\ \dots\ T-L,\ T\}$ do
    $i \leftarrow (t-1)\ div\ L$                    ▷ $div$ is the Euclidean division.
    $b_i \leftarrow \{d_t,\ d_{t+1},\ d_{t+2},\ \dots\ d_{t+L-1}\}$
    $reward(d_t) \leftarrow r_t;\ reward(d_{t+1}) \leftarrow r_{t+1};\ \dots\ reward(d_{t+L-1}) \leftarrow r_{t+L-1}$
end for

/* Evaluation of $\mathcal{A}$. */
$init()$
$score \leftarrow 0$
for each $i \in \{1,\ \dots\ T/L\}$ do
    $d \leftarrow chooseDisplay(b)$
    $r \leftarrow reward(d)$                    ▷ Only one reward from the batch is revealed.
    $update(d, r)$
    $score \leftarrow score + r$
end for
return $score$
```

2.2.3 Imposed constraints

Besides the somewhat groundbreaking evaluation method, other aspects of this challenge make it all the more interesting. Firstly it is organized in two phases. During the first and main phase, that lasted around three months, the evaluation of the algorithms was made on a remote server in order to prevent any offline preprocessing. Although it may seem a bit extreme, the idea is to push the participants to design innovative and flexible online learning approaches.

Another interesting aspect of this challenge is the limitation of the computational resources. At all time the participants only have access to 1GB of memory. Even more constraining, each batch has to be processed (chooseDisplay and update procedures) in less than 100ms on a 2GHz core. Any iteration taking more than 100ms is interrupted and considered as a random choice. In this case the model is not allowed to be updated. If an update is ongoing when the evaluation procedure decides to stop the iteration, the model is reset to its previous state. Note that we realized during the course of the challenge that due to the use of a bad way of measuring time, any answer taking more than 10 ms in average had a non negligible chance to be interrupted often enough to bias the evaluation. We did not investigate this issue in great details but we identified two main causes: Java's regular calls to its garbage collector and the fact that pure clock time is taken into account and not the actual processing time. Therefore any operation of the OS of the remote server (that are obviously not uniformly distributed among the iterations of an evaluation) decreased the time available for the algorithm at a given iteration.

Regardless of the technical issues it involves, this constraint is very interesting because it is a real incentive for the participants to design models that can really be updated online. Indeed it is impossible to store all the data inside the memory. Moreover, heavy computations, such as the computation from scratch of the parameters of the model during an iteration are totally impossible. As an example, it would be unthinkable to use an approach like *LinUCB* that inverts a matrix so as to compute a regression that maps contexts to rewards at each time step. This the reason why in section 1.4.3 we argued that *LinUCB* is not really an online approach. Rather we may consider it as an offline approach applied to an online problem. The organizers of this challenge wanted to prevent such approaches.

Finally, although the evaluation data is not revealed to the users there is a severe risk of overfitting due to the repetition of submissions. See the Netflix prize for an example [9,87]. To measure it, the submitted algorithms are only evaluated on the first part of the data during phase 1, that is $3 \cdot 10^6$ lines or $5 \cdot 10^5$ batches. At the end of this first phase of repeated submissions, the algorithms of the participants are evaluated on the rest of the data. This is the second phase of the challenge. Note that the entire dataset consists in 18.10^6 logged interactions. Note also that at the end of the second phase, the data corresponding to the first phase is made public. The participants are allowed to use it to tune their algorithm before being run against the rest of the dataset.

2.2.4 General considerations

The task of the challenge has already been defined and described enough to make this chapter broadly understandable. Yet to fully understand our research process, the expectations of such a challenge and the results, some additional details are worth being mentioned. Note interestingly that all these considerations are also valid for the challenge we organized (chapter 4), although the evaluation procedure is different.

Remote server evaluation The fact that the data is kept hidden is not as harmless as it seems. First names and purposes of the attributes are unknown. This makes it impossible to rely on any expert knowledge or even intuition of any sort. We only know that all the attributes are somehow characterizing the visitor (or his or her visit in general) except one: the id of the item. Second for debugging purposes, we were initially given 60 lines of the dataset. Yet the participants were warned that those 60 interactions should not be assumed to be representative of the whole dataset in terms of attribute values and their distribution. In particular, the number of successful displays is grossly over-represented in these 60 lines with regards to the whole dataset. Thus, it is clear that one should neither train one's algorithm with these 60 interactions, nor even learn precise information about attribute values with them. Not knowing the distribution of the features is quite common. Yet not being aware of the range of the continuous attributes nor the domain of definition of the discrete ones is rather annoying. One thing we noticed thanks to the small amount of data released before the challenge was that the features were clearly not on the same scale. Also, the values of the continuous features were not uniformly distributed. This is true on both a linear and a logarithmic scale. Rather even on this very small sample of data, the values of these features appeared to be gathered in clusters, although not very clearly delimited.

Missing values There are missing values (denoted NA) in the data for both continuous and discrete features. Note that we realized when the dataset was released that some attributes are in fact never available.

Non identified users As we already mentioned it, users are characterized by a set of features but they are not identified by a unique ID. Consequently we have no way to know if we see a user for the first time or not. Furthermore even though it could be thought reasonable to assume that two users represented by the same set of attributes are the same person (although we do not know for sure), it would be completely unreasonable to believe that two users with different attributes are necessarily distinct. Indeed we do not know what the attributes represent and some of them may vary *i.e.* if they are time dependent, location dependent or even if some additional information is discovered about the user between two visits. Nevertheless, with that many features it seems advisable not to consider users individually but to generalize thanks to all this information. Therefore not having access to such IDs is not such a big deal. It only prevents us from changing a recommendation that once was unfruitful if we see a user more than once.

Chance Our approach, as well as many others is stochastic. Moreover the order of the displays within a batch is not deterministic. Therefore given an algorithm, the score it produces is a random variable. This has advantages and shortcomings. One problem is that when a modification is made, we do not know if the score variation is due to that modification or simply to chance. A careless approach may cause to ignore positive modifications and to keep useless or even negative ones. Useless tweaks are not harmless because they increase the complexity of the algorithm in both number of parameters to tune and time. The advantage is that by submitting several time an algorithm, we can have a clearer idea of whether or not a score improvement is significant or just an artifact. Note that during phase 1, only the best score is taken into account. Thus if we compare an algorithm with excellent average performance to an algorithm that exhibits slightly lower performance in average but has a higher variance (more specifically a higher upper bound of performance), we should keep the latter for phase 1 and run it several time until reaching a nice peak of performance. On the contrary, it would be best to keep the first one for phase two given that only one evaluation is performed.

Fast algorithms It is only possible to submit one algorithm at a time. The score is given right after it is computed along with a notification that a new submission is possible. This severely limits the possibilities of hyper-parameter optimization. This has another impact on the challenge. The goal of the resource limitations is to force the participants to design fast, online approaches. Here we have another incentive. The faster the algorithm is, the more often we can submit a new version and thus the more we have the possibility to improve it. To have an idea, the first phase lasted around three months, from the beginning of march 2011 to the end of may 2011. The time to perform an evaluation ranged from less than an hour to more than a whole day. The approach we present here, contrarily to others we tried and to some that were used by other participants, was among the fastest, giving us a significant advantage. Note however that this advantage was slightly alleviated by the fact that the time to evaluate an algorithm also depended on the server load. As only a few cores where available, the simultaneous submissions of several computationally intensive algorithms could make a participant wait a day for a response to a submission of a fast algorithm. To some extent, it leveled the possibilities of submissions of the participants but not completely.

2.3 Our approach

We started by investigating in parallel a few different approaches based on different ideas. In particular we tried a kind of online random forest with upper confidence bounds at its leaves that yielded interesting results. Yet we only present here the approach that was the most promising and allowed us to win the challenge. This approach is based on the adPredictor™ algorithm, designed by Graepel *et al.* [6] to predict click rates in online advertisement. Surprisingly this solution is inspired from the TrueSkill™ algorithm by Herbrich *et al.* [86] used to rank online players on the Xbox gaming network. The main idea is to consider the probability of click of a user on an item as the sum of some contributors having different levels of uncertainty and which are updated in a Bayesian way. More specifically if the reader is familiar with the TrueSkill algorithm or more generally with player ranking algorithms, the idea of adPredictor is at the same time very interesting and quite surprising: a display is actually considered as a team of players. This team - whose player are in fact discrete values taken by the different features - always plays against the same opponent (an opponent of strength 0). The strength of the players is inferred by considering that a click is a win and a non-click a loss. A team (a display) with a strength greater than 0 is likely to win as it is stronger that the opponent. The display is thus likely to yield a click. On the contrary a negative strength indicates a low probability of click. The rest of this sections gives the full detail of the algorithm.

2.3.1 The click model

The model for the click probability of a visitor on an item is made of F *discrete* features built on the feature space $\mathcal{C}_c \times \mathcal{C}_D \times \{1, ...K\}$ of dimension 120. Let us denote this engineered discrete feature space by $\mathcal{C}_E$ of dimension F. *A priori* the item space and $\mathcal{C}_D$ are left unchanged for they are already discrete. $\mathcal{C}_c$ needs additional processing to be made discrete. Therefore nothing prevents us from building more than one discrete feature per continuous feature. One may even envision computing the square, the logarithm or some other function of a continuous feature. Several distinct discretizations are also possible and can be used conjointly. The detailed construction of these features will be discussed mainly in section 2.4.2. Before that, let us precise that for *each possible value of each feature*, we estimate a Gaussian distribution $\mathcal{N}(m, \sigma^2)$ (that is to say a mean and a variance). We call these estimated distributions *Gaussian weights*. To complete the analogy with player rankings, these weights are in fact the strengths of the players composing our team with the respective uncertainty we have on their estimations. A player is a specific value that a feature can take and a display is a team of F players.

For a given display d, each discrete feature of $\mathcal{C}_E$ takes a value. Note that we consider the missing values (NA) simply as a value the feature can take. The particular values taken by each feature of a given display d are said *active* for d. The Gaussian weights associated to these values are called d's *contributors* and reflect the contribution of each feature to the click probability of d. This contribution is more or less uncertain depending on σ^2, the variance of the Gaussian weight. Note that these various contributions add up to predict d's click probability: the model is additive. We note $active(d)$ the *active contributors* of a display d. Given a *contributor* c, m_c (resp. σ_c) is the mean (resp. the standard deviation) of the associated Gaussian weight. For a given display d, the contributors are thus the respective strengths of the selected players playing for the team. Note that while the players can change from one display to another (if a feature simply takes a different value), the opposing team is always the same. This opposing team simply has a strength of 0. Thus a contributor with a positive value contributes toward predicting a victory, and thus a click whereas a negative contributor helps predicting a defeat (the user does not click).

2.3.2 Making the decision on a batch

We have provided intuition and notations on the linear, additive model we use. Let us now formalize how we use this model to compete in this challenge. When a batch b is received, a score $Score(d)$ is computed for each display $d \in b$. This score is a *Gaussian distribution* and is computed as follows:

$$Score(d) = \sum_{a \in active(d)} \mathcal{N}\left(m_a, \alpha.\sigma_a^2\right) = \mathcal{N}\left(\sum_{a \in active(d)} m_a \quad , \quad \alpha. \sum_{a \in active(d)} \sigma_a^2\right),$$

where α is an additional parameter that did not exist in *AdPredictor* but that we introduced to tune exploration more finely during the challenge. Then the actual score for a display is a *sample* from $Score(d)$. Note that to do so we use the distribution on the right hand side of the equation above. Instead we could sample from each weight - which are Gaussian distributions as well - and then sum up the samples. Yet this less computationally efficient because sampling and multiplying is more expensive than adding.

We obviously select the display with the highest sampled score. That way the chosen display is not necessarily the one with the highest expected score. This is how the algorithm handles the exploration *vs.* exploitation dilemma. The basic idea is that the more the uncertainty (*i.e.* the standard deviation of the contributors) shrinks, the less likely it is to select a display supposed to be suboptimal as we are confident that it really is suboptimal. Moreover a display on which we have a lot of uncertainty compared to the others will have more chance to beat the one with the highest expected score making the exploration process very efficient compared to uniform exploration policy such as an ε-greedy strategy. This is a very old approach to the exploration *vs.* exploitation problem. Indeed a very similar approach called Thompson sampling was designed as early as in the 30s [45] (see also section 1.4.2 for more details and algorithm 2 for a non-contextual implementation with Bernoulli rewards). For theoretical guarantees on that kind of exploration strategy, the interested reader should refer to Kaufmann *et al.* [55].

2.3.3 Online learning

The model is as simple as it gets. The way it is updated is at the same time not much more complicated and very clever. After a display d has been chosen, the algorithm is notified whether d led to a click or not, by way of the binary reward. The active contributors of d are updated in a positive way if a click occurred, and in a negative way otherwise. This comes close to the update of the weights of a Perceptron [88], but in the present case, the weights are not scalar but Gaussian, and this leads to smarter updates as the uncertainty on each individual weight is taken into account. The update equations are actually derived from tools of graphical model theory. We make the choice of not providing details on the way they were derived. Indeed, we would take a lot of time describing a work which is not ours without deepening our understanding of the algorithm. Instead, because the resulting equations are very elegant and intuitive, we only provide intuition that allows to fully grasp the mechanisms ruling the updates. Let us still provide a few pointers for the reader interested in more justifications on the approach. The basic idea is to lay down the contributors into a factor graph (each contributor is a node of the graph) and to connect them all to a score node. To this score node is connected another node that transforms it to a probability of click via a Probit function. Then because exact inference is generally computationally intensive, even in this simple case, an approximation is used to compute the update equations: the approximate message passing algorithm. For more details on message passing within factor graphs, the reader is referred to Kschischang *et al.* [89]. For details on the way the algorithm is applied to our case, the reader may refer to Herbrich *et al.* [86] (full version of TrueSkill with two opposing teams) or to Graepel *et al.* [6] in which the exact version we use (the opposing team is a constant: 0) is described.

Graepel *et al.* [6] simply justify the equations by showing how the approximate inference method builds them from the factor graph. These equations are surprisingly intuitive and elegant if we take the time to rewrite them which is exactly what we do here. To define these equations intuitively, we define three variables. First, the click variable $y \in \{-1, 1\}$, that takes the value 1 if there is a click, -1 otherwise. Then we introduce Σ that quantifies the uncertainty on the estimated score and Λ that quantifies its correctness:

$$\Sigma^2 = \beta^2 + \sum_{a \in active(d)} \sigma_a^2, \qquad \Lambda = \frac{y \cdot \sum\limits_{a \in active(d)} m_a}{\Sigma},$$

where β is a real valued parameter.

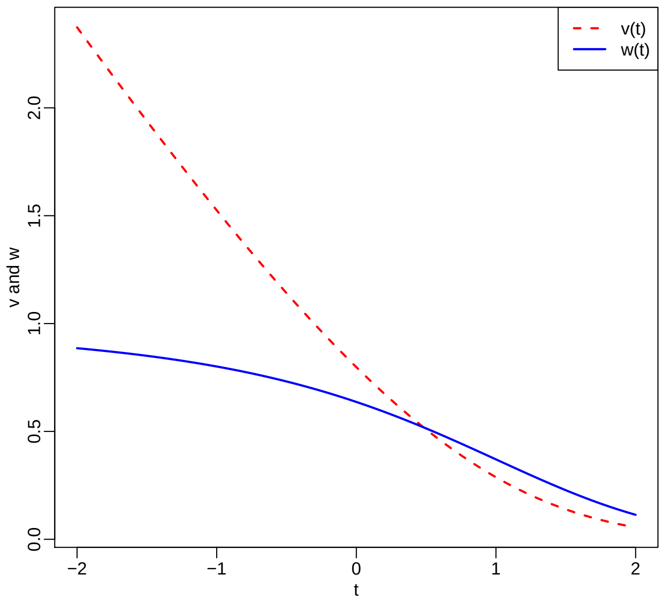
It is thanks to these quantities that we make smart updates. Σ^2 is defined as the variance of the score plus a parameter. This parameter β is used to artificially increase the uncertainty we have on the score when performing the updates (but not when exploring). The correctness Λ is defined as the estimated score (computed before knowing the reward) times a $\{-1, 1\}$ reward and then divided by a measure of uncertainty. The idea is very simple. A positive score predicts a click whereas a negative one predicts that there will not be a click. As such, after the multiplication by y, the numerator is positive if the prediction was correct and negative if the prediction was wrong, hence the term correctness for Λ. The second trick is the division by the uncertainty ,a positive number. One way to look at it is that the more uncertain the prediction was, the less it can be considered as really meaningful. In other words, a correct (resp. incorrect) prediction is considered even more correct (resp. incorrect) if we were sure of it *a priori* (low uncertainty). On the contrary, an uncertain prediction cannot be considered as either very wrong or very right for we knew that it was likely to be wrong anyway.

Using these quantities, the update rules for the parameters of the active contributors are as follows:

$$m_a \leftarrow m_a + y \cdot \frac{\sigma_a}{\Sigma} \cdot v(\Lambda), \qquad \sigma_a^2 \leftarrow \sigma_a^2 \cdot \left(1 - \frac{\sigma_a^2}{\Sigma^2} \cdot w(\Lambda)\right),$$

where v and w are real valued functions. Note that only the active contributors are updated. Many different functions can be used for v and w. With respect to our tests, as long as the shape is similar, the results are quite robust to variations on v and w. Because it does not change much we simply use the same functions as Herbrich *et al.* [6] (see Fig. 2.2). Note that to speed up our algorithm, we precomputed a table of values for v and w that was preloaded within the *init* function. Also because having a very good approximation of the true functions is not determinant with regard to performance, values that were not preloaded were computed by a simple weighted mean. More precisely, for a value X between two loaded values X_i and X_{i+1}, $v(X)$ is given by:

$$v(X) = (X - X_i)\, v(X_{i+1}) + (X_{i+1} - X)\, v(X_i), \qquad \text{where } X_i < X < X_{i+1}.$$



$$v(t) = \frac{d\Phi(t)}{dt} \cdot \frac{1}{\Phi(t)}$$

$$w(t) = -\frac{dv(t)}{dt}$$

with $\Phi(t)$ the cumulative distribution function of $\mathcal{N}(0,1)$

Figure 2.2: The functions v and w. They have this form due to the Probit regression function used by Herbrich *et al.* [86]. Yet any function that respects the intuition we describe in section 2.3.3 would work as well.

The main idea of the update equations is that the update is small if the outcome is not surprising (*i.e.* the prediction was correct). This can easily be seen from the shape of v and w (see Fig. 2.2). Indeed when Λ is negative (the prediction is wrong), $v(\Lambda)$ is large and so is the correction made to m whereas if Λ is positive (the prediction is correct), then almost no update is performed. The same thing is true for the multiplicative update of σ as when Λ grows, $w(\Lambda)$ gets closer to 1. To see why this is important, let us consider a bad contributor c active along with a lot of good contributors in a display. If we observe a click which is not surprising given the number of good contributors, should we change a lot our estimation of c? Obviously the answer is no since it is very likely that the click has resulted from the contributions of the good ones and not from a bad estimation of c. This does not mean either that the estimation that c is bad is correct but simply that this click is not very meaningful with respect to that estimation. As a consequence this click should not change what we previously thought too much. We could provide other example such a good predictor that loses within a bad team, or even a certain predictor within an uncertain team. Anyway the fact that these cases are addressed following what our intuition would suggest is what makes the update equations really smart. In comparison the Perceptron for instance just performs a positive update for a click and a negative update otherwise, whose magnitude is the same for all the contributors.

This update strategy offers some similarities with TD-learning [1]: the more we are surprised by the consequences of a decision, the larger the correction. This strategy usually leads to fast convergence rates of learning.

Another interesting idea is the use of the uncertainty. It makes the learning of the model much more robust than the learning strategy of a simple Perceptron for example. Indeed, let us consider the case in which we are sure that a display is bad (low uncertainty on the contributors), and in which unexpectedly, we get a click. This is very surprising, but since the update also depends on the uncertainty, we will not make a big update because of an event that is most certainly noise.

2.4 Our performance along the course of the challenge

The goal of the challenge is to score as much as possible on the 5.10^5 first batches (3.10^6 displays). The final score is the only output we get when an algorithm is submitted; so it is the only criterion we will use to compare different approaches in this section. This section describes how we improved our program along the course of the challenge. For a pseudo-code implementation of our final algorithm, see algorithm 7.

2.4.1 Early results

A purely uniform random strategy scores slightly lower than 1200 on 5.10^5 batches; a strategy that always chooses the same option scores equally. If we model the number of clicks with a Poisson law of parameter $\lambda = 1200$, a significant difference (with risk 5%) is 57 points. This order of magnitude should be kept in mind when trying to compare two different scores although intelligent strategies tend to be much less variate.

The algorithm presented in section 3 needs discrete features so during the first trials, we just ignored the continuous ones. Moreover as the main goal of the challenge was to do recommendation, it is quite natural to try to catch the preferences of the visitors in the features. So we simply built 20 features, each one of them being the Cartesian product of a discrete feature (a dimension of $\mathcal{C}_D$) with the item space. As an example, let us consider $K = 2$ and a feature of cardinality 2. A feature based on the Cartesian product of this feature with the item space can take the 6 following discrete values:

$$(1, v_1) \;\; (2, v_1) \;\; (1, v_2) \;\; (2, v_2) \;\; (1, NA) \;\; (2, NA).$$

It is thus associated to 6 distinct contributors (Gaussian weights) that can be activated in turn.

As the example shows, we handled the missing values by just assigning them to a specific contributor. Note that another strategy would be to simply not select any contributor for a feature with a missing value. Yet this strategy that we tried several times with different approaches always deteriorated the performance. This probably means that missing values are informative in some sense.

In what follows we will consider various lists of features on which we use our algorithm. Here, the list of 20 features we use can be written as follows:

$$\left\{ f_d \overset{dim}{\subset} \mathcal{C}_D | f_d \times \{1, \ ... \ K\} \right\},$$

where $\overset{dim}{\subset}$ is an operator for spaces that we will use a couple of times in this chapter. It means than the left operand is a dimension of the right operand. In other words it means that the left operand is one particular feature contained in the right operand: a feature space.

Note that running the algorithm on this list of features coming from Cartesian products is equivalent to having 6 separate models, one for each item, using only $\mathcal{C}_D$ as a feature space. Indeed, two contributors of two different features associated with different items can never be active together. This is actually the approach we took in the beginning but we changed a bit later to be able to handle both features resulting from a Cartesian product with $\{1, ...K\}$ and features independent from the item without having to create duplicates.

Quite surprisingly the first attempts scored the same as a random policy. Our first improvement occurred when we set the value of the parameter β to 100 instead of 1 which was the value in use in previous experiments [6,86]. We recall that β^2 is added to the uncertainty of the score when performing the updates (see the definition of Σ, the uncertainty in section 2.3.3). Its role is to soften the updates by telling the model that the predictions are very uncertain. In the beginning of the evaluation, it causes the model to converge at a slower pace, which avoids converging to non optimal weights in case of noisy observations (which are typical on the web and with that many features). When the model is confident about its weights (low values for the standard deviations), a bigger value for β allows to keep performing small adjustments. With $\beta = 100$ our score was approximately 1500 (25% more than random). Then our score went up to 1610 (34% more than random) after optimizing β a little better and setting the prior on the variance of all the contributors to 10. A zero prior on the mean seems to work fine. It was a good start as we performed significantly better than the random strategy using only 21 out of the 120 features at our disposal. Notice that although 21 of the original features were used, our list of engineered features was only of length 20 given that the item space was only used to compute Cartesian products.

2.4.2 Discretization

Cartesian products

From this point, the most natural idea to get a better performance out of this model was to include the continuous features. Since we need discrete ones, we began with a discretization of the continuous values in 5 buckets. Cutting values were fixed using the first seen values using an EM algorithm [90] to cluster the 64 first values in 5 distinct buckets, each associated to a contributor. As with the discrete features, the non attributed (NA) values (or missing values) had their own contributor for each feature.

We note $\mathcal{C}_c^*$ the discretized version of $\mathcal{C}_c$. Using this, we tried to run the algorithm with different versions of the model.

- Still with the idea of trying to identify the personal taste of each user, we first tried to do a Cartesian product of each feature with the item space. With the following feature list:

$$\left\{ f_c \overset{dim}{\subset} \mathcal{C}_c^* | f_c \times \{1..K\} \right\}, \left\{ f_d \overset{dim}{\subset} \mathcal{C}_D | f_d \times \{1..K\} \right\},$$

our algorithm scored around 1550. It was very disappointing since it is worse than with only the discrete features only.

- As adding the continuous features crossed with the item space made the performance decrease, we tried to add them without any Cartesian product. With the following list of features:

$$\mathcal{C}_c^*, \left\{ f_d \overset{dim}{\subset} \mathcal{C}_D | f_d \times \{1..K\} \right\},$$

our algorithm scored around 1900 (58% more than random).

- As removing the Cartesian product with $\{1..K\}$ for $\mathcal{C}_c$ drastically improved the performance, we also tried $\mathcal{C}_c^*, \mathcal{C}_D$ (no Cartesian product) but our algorithm only scored around 1600.

We tried to combine (with a Cartesian product) only a few features from $\mathcal{C}_c$ with the item space but no matter which ones we picked, we found no contribution scoring more than 1900. We also tried to identify online which features from C_d were actually correlated with the option using an ANOVA and then to dynamically combine them with the item. It did not work either. According to these results, it seems clear that the features from $\mathcal{C}_D$ are characterizing the user preferences whereas the features from $\mathcal{C}_C$ are characterizing the general behavior of the user when facing a recommended item.

A better discretization

We were trying different approaches in parallel to the one we present in this chapter and we had noticed that using only one cutting value per feature was pretty efficient (so two buckets per feature). Nevertheless trying to find the good cutting value using only the 60 given lines was pretty hopeless. That is why we decided to build several 2-buckets discretizations for each continuous feature and then choose the best one online.

More formally, for each feature $f_i \overset{dim}{\subset} \mathcal{C}_c$ we have a set S_i of possible cutting values each of which corresponds to a discretization in two buckets. The possible cutting values are chosen before running the algorithm looking at the 60 lines of data we have been given. Note that for each f_i, we took $|S_i| \approx 20$ in order to cover f_i quite exhaustively, going even out of the observed range in the sample given its limited size. For each cutting value v_{ij} of S_i we have 2 Gaussian weights $w_{ij}^{(1)}$ and $w_{ij}^{(2)}$ (one for values lower than v_{ij} and one for greater values).

In principle, for a given display, each feature has one *active* Gaussian weight (see section 2.3.1). To compute the score of this display, we sum these *active* Gaussian weights (see section 2.3.2). Here for each feature $f_i \overset{dim}{\subset} \mathcal{C}_c$ we have several active weights, one per discretization/-cutting value. So for each $f_i \overset{dim}{\subset} \mathcal{C}_c$ we only consider as *active* the one weight associated to the cutting value which maximizes the following criterion:

$$I\left(\sigma_{ij}^{(1)} \leq T(t) \;\; AND \;\; \sigma_{ij}^{(2)} \leq T(t)\right) \times d(w_{ij}^{(1)}, w_{ij}^{(2)})$$

where:

- $\sigma_{ij}^{(k)}$ is the standard deviation of the weight $w_{ij}^{(k)}$.
- I is the indicator function. It returns one if its input is true, 0 otherwise.

- $d(w_1, w_2)$ is the probability that the Gaussian weight with the greatest mean between w_1 and w_2 is actually greater than the other. It is a metric of how far apart the two weights are. We recall that computing this probability is trivial from the probability density of the normal distribution. Indeed if we consider two Gaussian weights $w_1 \sim \mathcal{N}(m_1, \sigma_1^2)$ and $w_2 \sim \mathcal{N}(m_2, \sigma_2^2)$ which are in fact nothing more than normally distributed random variables, we have that:

$$d(w_1, w_2) = \mathbb{P}(X > 0) = 1 - \mathbb{P}(X \leq 0), X \sim \mathcal{N}(|m_1 - m_2|, \sigma_1^2 + \sigma_2^2).$$

- $T(t)$ is a threshold function depending on time. The one we used is just a linearly decreasing function of time $T(t) = \gamma \cdot t$ with a minimum value (time is actually the batch index).

Even with these multiple discretization, using the criterion, we only use one contributor per feature to compute the score prediction. Note also that this is not compulsory. We tried to use them all in order to drag more information out of them. Yet it made the performance decrease, probably because it reduced the accuracy of the predictions. During the update however, we update the *active* weights of all the discretizations as in section 2.3.3. Otherwise we would not be able to compute the criterion.

To sum up, for each feature $f_i \overset{dim}{\subset} \mathcal{C}_c$ instead of building only one complex discretization we build $|S_i|$ discretizations with 2 discrete values (so with only on cutting value). We then choose the cutting value with the two most separated weights if they are both accurately enough estimated. This approach slightly outperformed the EM based discretization mentioned above (see Sec. 2.4.2). Indeed, it scored around 1940 (62% more than random).

Note that while running, the algorithm could add new values to one of the sets of possible cutting values S_i if it observed too much values outside of the minimum and maximum observed in the 60 given lines for feature f_i. However when we were able to run the algorithm by ourselves after the end of phase 1, we noticed that it was never the case, hence the good results we observed with the preselected cutting values.

A simpler discretization

To check whether we were learning interesting things with our two previous online discretization techniques (EM and sets of cutting values) we tried to build an offline baseline. To do so, we looked at the density plot of the continuous features for the 60 lines of the dataset given during phase 1 (see Fig. 2.3) and built coarse discretizations based on what we saw. We can divide the 99 continuous features into four groups.

1. **The group of similarly distributed features** where all the features looked almost exactly like the one on the right of Fig. 2.3. They represented one third of all the continuous features and we only built a unique handmade discretization for all of them. We simply made two buckets for the two peaks, two buckets for possible out of range values with the rest of the data and one for missing values.

2. **The group of zeros** where the features are equal to 0 roughly 80% of the time. They represent half of all the continuous features and we only built a unique discretization for all of them. This discretization simply consisted in a bucket for the value 0, a bucket for negative values, and then an exponential series of cutting values (0.01, 0.1, 1, 10...). Note that all the values we saw in the 60 lines for these features were integers between 1 and 50. Yet as contributors that are never active are harmless, we played safe and added a lot of them.

3. **The group of missing values** where all the values in the 60 lines were missing. We just arbitrarily took the discretization of the group of zeros. Note however that we found out during phase 2 that these features were actually NA in all data so what we did about them does not matter much. The were 6 such features.

4. **The group of outliers** where all the features are very different from one another and from the other groups. We simply made one bucket per peak on the density plot. The two leftmost plots of Fig. 2.3 are the densities of two of these twelve features.

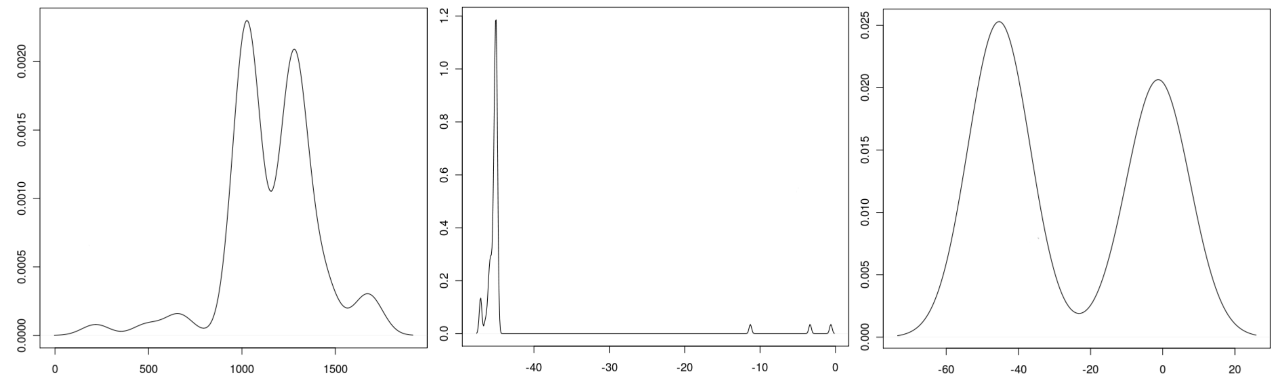


Figure 2.3: The density plot of 3 continuous features for the 60 lines of the dataset given during phase 1. The two on the right are on a logarithmic scale.

With this simple method of discretization, our algorithm scored 1950 (63% more than random) which was already more than the previous approach. We then focused on the two largest groups of features and after merging and splitting a few intervals, we were able to bring our score up to 2000 (67% more than random) and sometimes a bit more because of the variance of the algorithm. The details are purposefully omitted for they mostly consist in trial and error, repeating the submissions a few times not to be fooled too much by the variance of the evaluations. This has remained our best score for a while. At this point, we more or less tied with two other competitors and another approach of ours.

2.4.3 Dynamics

The batches were served in chronological order during the challenge to let us identify the dynamics of the system. When observing the update equation for σ one may notice that whatever happens, σ always decreases making the model unable to handle the fact that some weights may vary over time. Note that this is a other side of the coin of the smart updates. Indeed with its naive and constant updates, the Perceptron can naturally handle shifts in the values of the weights it is learning. Anyway, none of the approaches we tried here allowed to improve our performance. We try to explain why in section 2.5. Yet the reader might be interested in these methods to apply them on different learning problems. Consequently, we describe them in appendix E.2.

2.4.4 Exploration and stochasticity

Throughout the challenge we had been sampling from the score computed as described in section 2.3.2 with $\alpha = 1$. Actually this parameter did not even exist in the literature [6, 86]. As none of our approaches seemed to do better than 2000 we were starting to explore different paths. Just to see what happened we submitted an algorithm which was not taking into account the standard deviations in the prediction phase (it was actually summing up the means of the weights, nothing more). Surprisingly this approach scored 2130 (130 points more than with exploration and 78% more than random). Then, we tried to optimize the value of α but it seemed that the best value was 0 (no exploration at all). A value up to 0.05 was not making any noticeable difference though.

We obviously tried a decreasing exploration approach as $\varepsilon_n - greedy$ but it did not improve the performance. The only one that seemed to slightly improve the algorithm (in terms of performance and variance) is the following: for the first 5000 batches (1% of the total) instead of choosing the display with the best mean score (or sampled score), we choose the display d such that $Score(d)$ has the highest *variance*. During phase 2, by running the two approaches 100 times, we were able to check whether the performance was really improved or not. Here are the results: using $Score(d)$ for the 5000 first batches: $mean = 2130$ $variance = 1086$, while using $var(d)$ for the 5000 first batches: $mean = 2140$ $variance = 587$. This is while trying to tweak exploration in various ways that our best score (2170) was achieved, probably due to a lucky run taking advantage of the variance of the evaluation process.

Note that as we already mentioned it, we certainly did so with a suboptimal approach that is more variate than the others. This is a perverse effect of such challenges with repeated evaluations which is much less known and predicted by organizers than overfitting for instance. If an evaluation method (or an evaluated algorithm) is stochastic, as long as the number of submissions that can be made is big enough, more variate approaches are better than stable ones. Indeed we can take advantage of a very favorable run even though we do not perform very well on average. In practice, people evaluating systems want to minimize the risk of error in their evaluation. Thus they are looking after approaches with very little variance or with high lower bounds that is exactly the opposite of the ones that are the more likely to win such a challenge (high variance, high upper bounds). Unfortunately, not much can be done by the organizers to tackle this issue without significantly increasing the amount of computational resources needed to run the challenge. If it is an option however, it might be enough to run each submission several times and to output a more risk-aware statistic.

2.4.5 Further score improvements

For phase 2 we got access to the data of phase 1. We thus acquired the possibility to run a lot more simulations to optimize the algorithm. Trying to optimize α, β and the prior on the weights led to minor improvements. For instance one small improvement was made when differentiating the prior on the variance of the discrete features and the continuous ones. See section E.3.2. Furthermore to be able to present the results during the workshop, we did again a few experiments. In one of them, we tried to use $\mathcal{C}_c^*, \mathcal{C}_D$ as a model which is the raw feature space with our discretization over $\mathcal{C}_C$ and without any intervention of the item space. We ran the algorithm 100 times with these features and it achieved a mean score of 2215 and a variance of 190. The best score we got during this experiment is 2240 (85% more than random) which is much better than our previous approach. This is the only major improvement we made during phase 2. This approach is the one that won the second phase of the challenge. In summary, the simplest version of the model ended up having the best performance and being the most stable. Note that adding the items as a 120th feature in that list of feature did not change anything. This was to be expected since we had been warned at the beginning of the challenge that the 6 items had similar click through rates.

Note that we had tried this approach before and it had scored badly on the server of the challenge. We can only try to guess why: a bug on the server side, the wrong file selected while sending the submission, an error in the code *etc.* Anyway, we could have avoided that kind of thing by submitting each algorithm more than once all the time and not only at the very end of the challenge when we were trying to tune parameters. Yet we would probably have lost a lot of time as one submission usually returned after several hours and it may have slowed down our progress to the point of not allowing us to win. This is however an interesting idea to think about in case of a future challenge. The simplest thing would be to run the algorithm 5 or 10 times in parallel at each submissions. However if computational resources are limited, it becomes an issue. To address this problem, more data could be given to the challengers so that they can test their algorithm by themselves before submitting it. Restrictions on the number of submissions per day could even be imposed in that case to easily allow a sharing of resources between challengers. In any case, mistakes can happen and we should be careful when drawing definite conclusions based on a single submission.

The pseudo code of the final implementation of the algorithm we described in this section is given in algorithm 7. Note that additional experiments on the role of the parameters (β and the priors) and the possibility to use Ad predictor to estimate the CTR of a display are available in appendix E.3.

Algorithm 7 Our version of Ad predictor [6] which won the challenge.

```
$T$: the number of initial pure exploration steps
$\alpha$ : The parameter to tune exploration
$\beta$ : A kind of learning rate: the larger it is, the slower the learning is
$m, \sigma \in \mathbb{R}^d$ : the prior vectors for the weights

function CHOOSEDISPLAY($Batch$)
    for each $Display\ d_i \in Batch$ do
        $s[i] \leftarrow 0$
        for each $a \in active(d_i)$ do
            if $\#iterations \geq T$ then
                draw $x$ from $\mathcal{N}(m_a, \alpha\sigma_a^2)$
            else
                $x \leftarrow \sigma_a^2$
            end if
            $s[i] \leftarrow s[i] + x$
        end for
    end for
    $maxIndex \leftarrow \text{argmax}_i s[i]$
    return $d_{maxIndex}$
end function
procedure UPDATE($display, click$)
    if click then
        $y \leftarrow 1$
    else
        $y \leftarrow -1$
    end if
    $\Sigma^2 \leftarrow \beta^2 + \sum_{a \in active(d)} \sigma_a^2$
    $\Lambda \leftarrow \frac{y \cdot \sum_{a \in active(d)} m_a}{\Sigma}$
    for each $a \in active(display)$ do
        $m_a \leftarrow m_a + y \cdot \frac{\sigma_a}{\Sigma} \cdot v(\Lambda)$
        $\sigma_a^2 \leftarrow \sigma_a^2 \cdot \left(1 - \frac{\sigma_a^2}{\Sigma^2} \cdot w(\Lambda)\right)$
    end for
end procedure
```

2.5 Understanding the results

The purpose of the challenge was to design an algorithm capable of doing three main things:

- identify the preferences of a visitor to recommend something to him/her,
- balance exploitation and exploration,
- adapt to the dynamics of the system (some items may for example become less popular for some visitors or the other way around).

However, in the challenge we had to choose between 6 couples $(visitor, item)$ and what we did is basically visitor selection. Indeed our best approach only used $\mathcal{C}_C$ and $\mathcal{C}_D$ - the user features - to make its decisions. Moreover as presented in the previous section the approach which performs the best neither explores, nor adapts.

2.5.1 Why did visitor selection work so well?

We identified the general behavior of visitors in front of a recommended item without needing to explore at all and without paying attention to a possible evolution in their tastes. When it came to choosing a batch, using this knowledge without considering the displayed item was the best option. As surprising as it may seem, we can try to explain why that works so well with both intuitive arguments and hard evidence.

Intuition Trying to find the click probability of a visitor in general is a lot easier than trying to find it for each item. Indeed each time a visitor is seen, the model can be updated whereas to find the preferences, no information is obtained for non displayed items. When working on the logs of a big french company we had also noticed that some variables are very important as far as clicks are concerned. For instance some pages have higher click probabilities than others and the time of the day matters (visitors do not click much at night). $\mathcal{C}_c \times \mathcal{C}_D$ might have contained that kind of features making our task even easier. Moreover half of the visitors in the company's logs never click. Identifying all of them in the challenge already doubles the click probability (which is basically the maximum score we achieved).

Intuitively the general behavior of someone on the Internet is very unlikely to change dramatically as opposed to his preferences. That is why trying to find some kind of dynamics was hopeless. So having to identify something pretty easy to characterize and very stable, the reason why almost no exploration was needed is rather clear.

Feature based explanation In a more pragmatic way we can explain why we performed so well without using the items (or any other Cartesian product) by looking at the click frequency plot of some features. Looking at Fig. 2.4, we can clearly identify patterns both for the discrete and the continuous features. The one in the middle of Fig. 2.4 is particularly informative. We tried to run a very simple UCB strategy as described by Auer *et al.* [13] with one arm per value of this feature (ties, that is when the feature takes the same value in two displays of the same batch were broken at random) and scored 1450. It means that using only one feature from $\mathcal{C}_D$ allows us to perform 21% better than a purely random uniform strategy!

2.5.2 Crossed effects

Does this mean that there are not any crossed effects between features in this dataset? The answer is very likely to be no. We performed an analysis of variance (ANOVA) and found a lot of crossed effects between the item space $\{1..K\}$ and some other features. We also found crossed effects between features of $\mathcal{C}_D$ and $\mathcal{C}_C^*$. During phase 1 we had tried to learn them

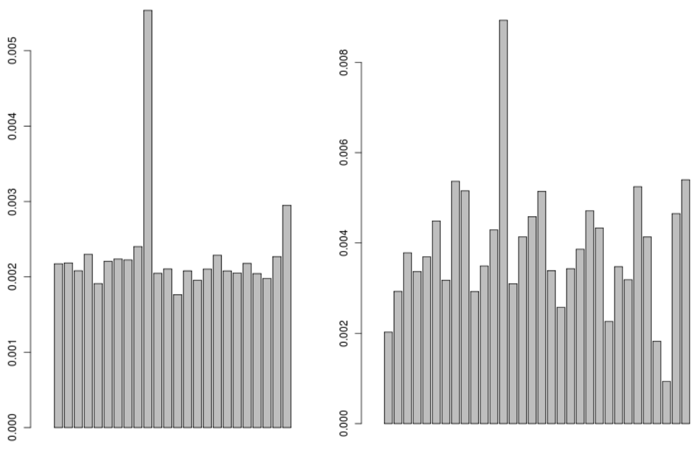


Figure 2.4: Click frequency against the value of 3 different features. The two features on the left are discrete and the values appearing less than 5000 times in the dataset have been removed. The one on the right is continuous and plotted on a log scale. For that latter feature, please note that the high frequencies on the right only represents 0.5% of the values.

online and during phase 2, we knew about the Cartesian products. However we still have not been able to exploit them. We can only try to give an intuition to explain that. The general behavior of visitors seems to be something very stable and very well characterized by the set of features we have been given. Trying to enhance the model with things like preferences which are much more unstable and much harder to identify is very tricky. In our experiments we have even noticed that it adds disturbance to the model making it less efficient in learning and using the behavior of visitors.

2.6 Comparison with state-of-the-art algorithms: LAP, an *online* alternative to LinUCB

The algorithm we used to win this challenge uses discrete features and proved to be quite efficient with the task defined here. When we are dealing with continuous features the algorithm has to rely on another tool to compute a discretization. However what could be considered a shortcoming of the algorithm proved not to be that important as a very coarse hand-made discretization allowed us to perform very well.

Nevertheless it would be very interesting to be able to compare fairly this algorithm on basic examples with the main state-of-the-art algorithm for dynamic recommendation/contextual bandits: LinUCB introduced by Li *et al.* [7]. The approach we used to discretize the data in the challenge would appear unfair compared to an algorithm (LinUCB) which would have to learn from scratch. Indeed it would be intellectually disturbing to compute a discretization to fit a model we designed ourselves before comparing it to another algorithm to which we would not give the slightest prior information.

At some point in the challenge, we came up with a version of Ad Predictor which is able to learn from (linear) continuous features without having to discretize them and that we call *Linear Ad Predictor* or more concisely *LAP*. We did not present it in the section presenting our successive approaches (sec. 2.4) as it did not help us improve our performance, on the contrary. This section starts with presenting this algorithm and then compares it qualitatively with *LinUCB* and *PerceptronUCB*, another algorithm we design. *PerceptronUCB* can be considered to be a simpler version of *LAP* only built on state-of-the-art techniques (the Perceptron and UCB). It is mainly meant to be a baseline for comparison. We also introduce an hybrid version of *LAP* which allows us to use heterogeneous features of various sorts.

2.6.1 Linear Ad Predictor (LAP)

First note that the algorithm we describe in this section is a contextual bandit algorithm exactly as defined in the previous chapter. The idea to allow this algorithm to deal with continuous features is very simple and relies on the same ideas as LinUCB or as the Perceptron algorithm. We assume that the click probability of a given display d is given by a linear combination of the values of the continuous features describing d.

In other words, for a given feature, instead of trying to infer one weight per possible discrete value, we only infer one single weight. The contribution of this feature will simply change proportionally to its value. Thus the score of a given display is given by:

$$Score(d) = \mathcal{N}\left(x^T w, \sum_{\forall i} \sigma_i^2\right),$$

where x is the vector of features corresponding to d and w is the vector of weights that the algorithm is trying to infer (one weight per feature). Note that we still denote by m_i the i^{th} element of w so as to maintain unified notations.

The inference process is a simple adaptation of the update equations for the discrete case. So with the same notation as in section 2.3.3 (y represents the click, Σ the uncertainty and Λ the surprise), we perform the following updates for all features i.

$$m_i \leftarrow m_i + y \cdot \frac{\sigma_i}{\Sigma} \cdot v(\Lambda) \cdot x_i, \qquad \sigma_i^2 \leftarrow \sigma_i^2 \cdot \left(1 - \frac{\sigma_i^2}{\Sigma^2} \cdot w(\Lambda)\right).$$

We do not explain the equations further as the ideas are the same as for the Perceptron (at least for the mean since the Perceptron does not estimate the uncertainty). However note that the update equation for the variance is the same as in the discrete case as we assume that the variance engendered by a feature does not depends on its value. Furthermore, as opposed to the discrete case and if we ignore the features with missing values that this algorithm cannot really handle, all the σ_i are updated the same way at each time step and are thus all equal. Indeed there is only one contributor (thus always active) per feature. This is why for this algorithm we can drop the index and compute σ (a scalar) only once per time step for all the features at the same time, which reduces the time complexity. σ acts as a decreasing coefficient for the exploration in a very similar fashion as ε for ε_n-greedy like strategies. The main difference is that the decrease rate is not set in advance. Rather the algorithm adapts to the data it meets. Note that for the real contextual bandit setting that we consider here, we need one tuple (w, σ) (w is a vector and σ a scalar) per possible item.

This very simple algorithm is the one we will compare to *LinUCB* as it uses the same kind of features and cannot be considered to be "cheating" by using preprocessed data . Algorithm 8 is the pseudo code of this contextual bandit algorithm. In the next section we extend this algorithm to cases that are more interesting in practice.

2.6.2 Hybrid approach: Generalized Ad Predictor (GAP)

We just presented a way to allow *Ad Predictor* - an algorithm meant for discrete features - to use continuous, linear features instead. The two methods are obviously not exclusive from one another. We can actually derive a very flexible algorithm that can use both types of features at the same time and combine them. Even more interestingly, we can define hybrid features to handle very efficiently and without assumption any function of the features. We call the highly flexible algorithm that can handle all these different features Generalized Ad Predictor (GAP).

Let us be more specific about those hybrid features. The classical way to handle a non-linear function of a feature f is to compute additional features such as f^2, f^3, $\frac{1}{f}$, e^f *etc.*and to treat them as linear features. Obviously such features can be added to GAP. The downside of this

Algorithm 8 Linear Ad Predictor (LAP).

```
parameter $\alpha$ : The parameter to tune exploration
parameter $\beta$ : A kind of learning rate: the larger it is, the slower the learning is
parameter $m \in \mathbb{R}^d, \sigma \in \mathbb{R}$ : the priors for the weights

function CHOOSE($x \in \mathcal{X}, \mathcal{A}$)
    for each $i \in \mathcal{A}$ do
        $score[i] \sim \mathcal{N}\left(w_i^T x, \alpha d \sigma_i^2\right)$
    end for
    return $\text{argmax}_{i \in \mathcal{A}}\, score[i]$
end function
procedure UPDATE($xin\mathcal{X}, ain\mathcal{A}, reward \in \{-1, 1\}$)
    $y \leftarrow reward$
    $\Sigma^2 \leftarrow \beta^2 + d\sigma_a^2$
    $\Lambda \leftarrow \frac{y \cdot sumOfElements(w_a)}{\Sigma}$
    for each $i \in \{1, d\}$ do
        $m_{a,i} = m_{a,i} + y \cdot \frac{\sigma}{\Sigma} \cdot v(\Lambda) \cdot x_i$
    end for
    $\sigma^2 = \sigma^2 \cdot \left(1 - \frac{\sigma^2}{\Sigma^2} \cdot w(\Lambda)\right)$
end procedure
```

approach is that it requires to know in advance the kind of function we want to learn. Yet there exists another method - easy to plug into GAP - that does not require any assumption and which consists in considering that features are linear by part, hence the term hybrid. More precisely a feature is *discretized* (say uniformly) and each part is considered as a *linear* feature independent from the others. This way any function can be approximated. Choosing the granularity of the discretization is a trade-off between accuracy and learning speed. Indeed, only one part is active and thus updated per display. We do not provide experiments with GAP nor did we use it in the challenge. Therefore its detailed presentation is in appendix E.1.

2.6.3 Intuitive comparison between LinUCB and LAP

The whole idea of this section is to compare *Ad predictor* and more particularly *LAP* with the state-of-the-art methods. Before doing so it is interesting to understand exactly what we are actually trying to compare. In fact *LAP* and *LinUCB* are trying to do exactly the same thing. Given some contextual information x, from a numerical context set $\mathcal{X} = \mathbb{R}^F$ both algorithms are trying for each possible action a to infer the best weight vector w_a such that:

$$\mathbb{E}\left[\vec{r}[a]\right] = x^T w_a$$

Note that *LinUCB* adds a regularization term but that does not change the purpose. So in other word they are computing K linear regressions that maps $\mathcal{X}$ to a reward expectation using the triplets (x, a, r) that have been encountered.

What is the difference then? At each time step *LinUCB* actually computes the exact solution of the (regularized) linear regression. This can be done easily by inverting a $F \times F$ matrix (see algorithm 5) but can be quite computationally expensive if F is bigger than a few tens. On the contrary, *LAP* operates in a true online fashion. The records are only used when they are received to update the model and can be forgotten afterward making the algorithm very efficient.

Note that we did not even consider solving an entire regression problem at each time step of this challenge because of the very tight time constraints. Even an approximate solution using gradient descent would have been too costly. Another team of participants of the challenge tried to do it anyway but they could only run the gradient descent on a summary of the data. See Salperwyck *et al.* [91] for more details.

That is what leads us to the conclusion that *LinUCB* should perform better than *LAP* because they both assume the same thing but *LinUCB* is given all the time it needs to fit exactly the data to the model whereas *LAP* tries to do it approximately on the fly in a true online fashion. One may even argue that *LinUCB* is not really an online algorithm and that it is like comparing an online method to its exact offline counterpart.

2.6.4 Perceptron-UCB

As we just explained it, comparing *LAP* with *LinUCB* is quite unfair and we expect *LinUCB* to win all the time since it computes an optimal regression from scratch at each time step. This is the reason why we want to compare *LAP* with another state-of-the-art algorithm which would also really learn online from the flow of data. This way we would be able to assess the quality of the update equations that we advertised to be smart and intuitive in section 2.3.3. Our problem is that there is not really such an algorithm in the literature. However there are algorithms used for online classification/regression that are relatively close to what we want to do. These algorithms are used to learn online a regression/classification function when receiving a flow of labeled examples (x, y), y being a class or a real value and x a feature vector. The most well-known among these algorithms is one we already mentioned as *LAP* looks similar to it: the Perceptron [88]. Moreover it also assumes that $\mathbb{E}[r] = x^T w$ with w a real vector that the algorithm is trying to infer.

What we want to do here is to design a simple online learning algorithm based on state-of-the-art methods and that can deal with the contextual bandits problem successfully. However the *Perceptron* is meant to learn from a flow of data but obviously does not define an exploration/exploitation strategy to determine which action to choose. We can overcome this issue very easily using the literature. Here we just need to have one *Perceptron* per possible action. To handle the exploration strategy, we can use another well known method: UCB [13]. The resulting decision rule is pretty simple. Given a context x after t interactions, we select the action a maximizing:

$$x^T w_a + \sqrt{\frac{\alpha log(t)}{n_a}},$$

where n_a is the number of times a was selected and t the total number of iterations. Similarly to what is done with UCB, all actions have to be performed once before starting using the indices to avoid dividing by zero.

One may argue that this exploration strategy is not contextual but the same goes for *LAP* as σ, the value ruling the exploration strategy, has the same value for each Gaussian weight. Indeed it is not easy to design a cheap exploration strategy. For example *LinUCB* has to compute a matrix inversion to manage it. Moreover this exploration strategy was proved to achieve the optimal regret, up to a constant when the rewards and features are normally distributed [7] which is the case in our generative model. This is one additional reason why *LinUCB* should beat *LAP*.

The update rule is the classical update rule of the *Perceptron*. When receiving a record (x, a, r), all the values $w_{a,i}$ of the vector w_a are updated as follows:

$$w_{a,i} = w_{a,i} - \gamma \left(x^T w_a - r\right) x_i \ ,$$

where γ is the learning rate. This update equation is ruled by the same ideas as *Ad Predictor* but is much coarser. In fact we let all the weights be corrected toward correcting the error with the same magnitude. The pseudo-code is given in algorithm 9. This *Perceptron-UCB* will be the contextual baseline in our comparison. We will loosely refer to it as the state-of-the-art baseline. It is obviously an overstatement given that we designed this algorithm ourselves. Yet it is just the concatenation of state-of-the-art learning equations (the Perceptron) and a state-of-the-art decision rule (UCB). As such it is a really nice baseline of comparison for *LAP* which is an algorithm that simply does exactly what *Perceptron-UCB* does but in a very slightly more elaborate way. One may still argue that *LinUCB* should be considered as the state-of-the-art baseline. Yet let us recall one more time that *LinUCB* does not learn online and uses a lot of computational resources compared to our two online algorithms: *LAP* and *Perceptron-UCB*. As such the comparison is not fair but can be used to quantify the loss of performance due to online approximations. This is exactly what we do in what follows.

Algorithm 9 Perceptron-UCB, a contextual bandit algorithm that learns like a Perceptron [88] and handles the exploration *versus* exploitation dilemma like UCB [13].

```
parameter $\alpha$ : The parameter to tune exploration
parameter $\gamma$ : The learning rate: the smaller it is, the slower the learning is

procedure INIT
    $n \leftarrow 0$
    for each $a \in \mathcal{A}$ do
        $n_a \leftarrow 0$
        $w_a \leftarrow 0_d$
    end for
end procedure
function CHOOSE($x \in \mathcal{X}, \mathcal{A}$)
    if $\exists a, n_a = 0$ then
        return $a$
    end if
    return $\text{argmax}_{a \in \mathcal{A}} \left( x^T w_a + \sqrt{\frac{\alpha r_a}{log(n_a)}} \right)$
end function                                              ▷ ties are broken arbitrarily
procedure UPDATE($x \in \mathcal{X}, a \in \mathcal{A}, reward \in \mathbb{R}$)
    $w_a \leftarrow w_a - \gamma \left( x^T w_a - r \right) x$
    $n \leftarrow n + 1$
    $n_a \leftarrow n_a + 1$
end procedure
```

2.6.5 Empirical comparison

This empirical comparison is about trying to verify that as *LinUCB* can be considered as a exact version of the two online algorithms *LAP* and *Perceptron-UCB*, it outperforms them. Furthermore quantifying the cost of the approximation made to be able to learn online via a simple update would be very interesting to quantify. Another interesting concern is the verification of whether our online algorithm *LAP* is better than *Perceptron-UCB* which was built using classical state-of-the-art techniques.

As *LinUCB* is an exact version of the other two, it will obviously be better but also more greedy in terms of computational resources. Since it is sometimes interesting to be able to speed up things at a cost of a loss of efficiency, we also got ourselves interested in the time the three algorithms take to run the experiment in order to quantify the gain in terms of speed and its cost in terms of performance.

In their article about *LinUCB*, Li *et al.* [7] propose a different way to overcome *LinUCB*'s greed in resources. The idea is very basic: since updating the model is what is really expensive,

Table 2.1: Theoretical time complexity of the five algorithms compared in sec. 2.6.5. F denotes the number of linear features of the context space we use and K the number of actions. We recall that *Eco-LinUCB* is a version of *LinUCB* that only call the update procedure every T_{up} time steps.

	UCB	*Perceptron-UCB*	*LAP*	*LinUCB*	*Eco-LinUCB*
choose	$O(K)$	$O(K.F)$	$O(K.F)$	$O(K.F^2)$	$O(K.F^2)$
update	$O(1)$	$O(F)$	$O(F)$	$O(F^3)$	$O\left(\frac{K.F^2}{T_{up}}\right)$

they propose to only do it only once every T_{up} time steps. To be completely fair we included this approach, that we refer to as *Eco-LinUCB*, in our experiment. Of course *LAP* and to a slightly lesser extent *Perceptron-UCB* are supposed to be much smarter and thus much more efficient ways to reduce the time complexity of *LinUCB*. We want to verify that here as well. Finally we also add *UCB* in the experiment to have a standard baseline for the computational times we measure. Note however that since the toy model we introduce is contextual, *UCB* is expected to perform very poorly in comparison with the other studied algorithms.

For the record, the detail of the theoretical time complexity of all these algorithms is given in table 2.1.

Model description

The generative model we designed for this experiment is pretty standard and includes some intuition we may have about recommendation in general. It is a linear model with some Gaussian noise and Bernoulli rewards that was built as follows:

- a fixed action set of size $K = 10$.
- The context space $\mathcal{X}$ has F dimensions (with $F = 15$ or 50 here). A given context is a real valued vector x of dimension F such that $x = c + n$ with the informative part c sampled from $\mathcal{N}(0, 1)$ and the noise n sampled from $\mathcal{N}(0, \frac{1}{2})$.
- Each action a has two hidden components: a real number p_a representing its overall quality and a real valued vector w_a of length F which characterizes its affinity with $\mathcal{X}$.
- The probability of success of an action a performed in a context x is given linearly by $p_a + w_a^T x$.
- Finally there are two kinds of actions:
  - $\frac{4}{10}$ of "universally" good actions that tends to be successful no matter what. As far as recommendation is concerned, these actions are items that are interesting to everyone. For such actions, p_a is high (sampled from $\mathcal{U}(0.4, 0.5)$) and $w_a = 0_F$.
  - $\frac{6}{10}$ of highly specific actions that are efficient in some specific contexts and for which p_a is low (sampled from $\mathcal{U}(0.1, 0.2)$) and w_a is composed of mostly zeros and a number sampled from $\mathcal{U}(\{1, q_{max}\})$ of relevant weights sampled from $\mathcal{N}(0, \frac{1}{5})$ (q_{max} being a parameter of our experiment. The bigger it is, the more personalized recommendation yields big performance lifts).

Remark *LinUCB*, *LAP* and *PerceptronUCB* are all trying to infer a vector w for each action. With this model one needs to add an intercept term (*i.e.*, a feature constantly equal to 1) to the context. This way the vector to infer for each action so that $x^T w$ is equal to the reward expectation is in fact:

$$(p_a, w_a[1], w_a[2] \dots w_a[F]) \,.$$

Intuitively, for a news recommendation application, this model considers that some news are interesting to all such as "Obama is re-elected" and that others are not very popular but may be of high interest to a specific part of the audience that we hope to identify ("New Linux distribution released").

We will use this synthetic model several times in this thesis work and may refer to this section when need be. Note that it has two parameters that concern the features of the context space. F is simply the number of features. q_{max} is the maximum number of informative features for a "specific" action, that is that does not perform well in every context. q_{max} characterizes the level to which it is possible to personalize the recommendations. The greater it is, the more likely personalization will be possible and profitable.

Experimental protocol

We compare 5 approaches in this experiment. The first one is *UCB* (or more precisely *UCB2* by Auer *et al.* [13]) in order to have a non-contextual baseline. The next three are the one we presented in the previous subsections.

We set T_{up} to a value that allows *Eco-LinUCB* to use an amount of time of the same order of magnitude as the online algorithms: $T = 50$ when $F = 15$ an $T = 500$ when $F = 50$. Note that these values may seem huge but they are in fact quite generous as we will see in the next section that *Eco-LinUCB* still takes significantly more time than the three online algorithms.

When running experiments to compare algorithms, there is always one issue that comes into play: the tuning of the parameters. It is quite common in the literature to find a new algorithm with highly tuned parameters compared with a state-of-the-art method for which no tuning effort is made. Herlocker *et al.* [8] mentions this phenomenon for recommendation but this is very likely to happen similarly in a wide array of domains. Here we tried to be fair and give the same amount of parameter tuning to each approach using the following experimental protocol:

1. We reduce by hand the number of parameters of each algorithm to one by keeping the most significant one and setting the others to standard values.

2. 6 sub-problems are considered with F, the number of features, set to 15 or 50 and q_{max}, the maximum number of informative features per item set to 2, 5 or 10. We set the horizon (number of choose/update cycles) to be $T = 5,000$.

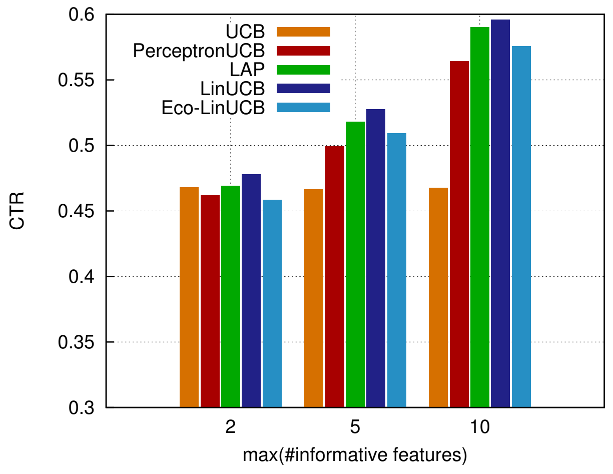
3. We select 8 possible values for the parameter of each algorithm.

4. For each sub-problem P and each algorithm A, we run A 1,000 times on P for each parameter and keep only the results corresponding to the parameter with the best average performance.

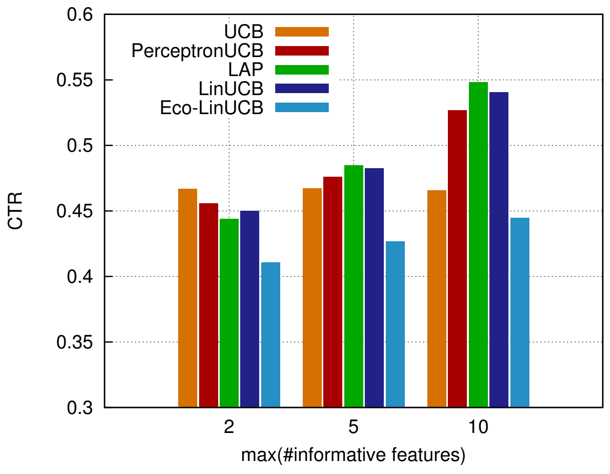
The first step of this protocol was done using the common knowledge or the experience we have about these algorithms. More specifically, here is how we set the parameters.

- **UCB** has only one parameter: α.

- **Perceptron-UCB**: We set the learning rate $\gamma = 10^{-3}$ which is quite standard and kept α, the exploration parameter, for the tuning phase.



(a) $F = 15$ features.



(b) $F = 50$ features.

Figure 2.5: Results obtained by the five algorithms on the six problems considered in section 2.6.5 (higher is better).

- **LAP**: From the experience acquired during the challenge, $\mathcal{N}(0, 10^3 CTR)$ seemed to be a reasonable value for the priors on the Gaussian weights (with CTR being an order of magnitude of the CTR of the problem). This is why we set σ_{prior} to 300 and m_{prior} to 0_F. Furthermore α multiplies $F.\sigma$ in the algorithm so to get a consistent algorithm no matter what the value of F is, we set $\alpha = (F.\sigma_{prior})^{-1}$. We kept β for the tuning process as it is the key parameter of the algorithm, playing a role both in the exploration and learning processes.

- **LinUCB**: We set $\lambda = 1$ as it is what is done everywhere and especially in the paper featuring it [7] and kept α, the exploration parameter, to be tuned.

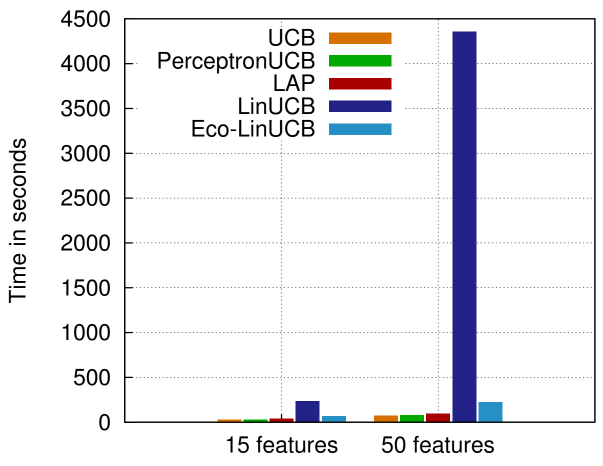
The range of values for the parameters is $\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3\}$ for α in the UCB-like algorithms and $\{1, 3, 10, 30, 100, 300, 1000, 3000\}$ for β in *LAP*.

With this protocol, none of the algorithms (even our own) is assured to reach its maximum of performance but at least they all have the same chance at it. Even if it could be improved a lot, we really encourage that kind of approach when comparing algorithms as it is very simple to implement and really brings more confidence in the results. Note that the performance of an algorithm is very often a more or less quadratic function of its parameter (or at least a function with a clear peak - examples are provided in section 4.10 for non-contextual bandit algorithms played on real data). Therefore the approach could be significantly improved by using a more or less robust gradient ascend instead of a coarse list of possible values.

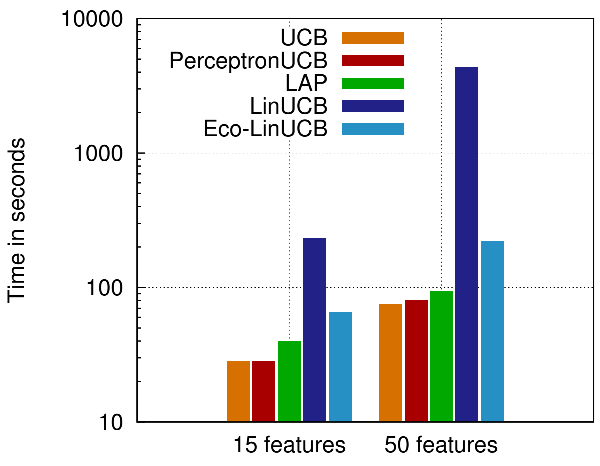
Results and conclusions

As it can be seen on figure 2.5, when $F = 15$ the results are almost exactly what we expected. *LAP* is better than its state-of-the-art version *Perceptron-UCB* but is outperformed by *LinUCB* which computes the exact linear regression at each time step. It is interesting to note than even with a small number of features it is more interesting to use *LAP* to save computational resources than what was proposed by Li *et al.* [7]: *Eco-LinUCB*. As we can see it on figure 2.6 *LinUCB* takes approximately 10 times more time than the others. Yet in spite of the huge speed-up entailed by the use of *LAP* instead of *LinUCB*, the performance drop is not that big.

When increasing the number of features to 50, the results are more unexpected and thus more interesting. Let us start with what is crystal clear: *Eco-LinUCB* is no longer a viable option to save computational resources as it is even clearly outperformed by *UCB*, the



(a) Regular scale.



(b) Log scale.

Figure 2.6: Time taken by the five algorithms considered in section 2.6.5 to run 1,000 times. The left-hand figure is on a regular scale to display the differences whereas the right-hand one is on a log scale for more precision.

non-contextual approach while still consuming a substantial amount of additional resources compared to *LAP* (we did not manage to reduce that consumption much further). There is not even the slightest trade-off going on here. *LinUCB* is simply very computationally demanding but quite efficient in terms of learning. When we reduce the number of updates, it simply becomes a very bad algorithm that still requires quite a lot of resources. Indeed in practice *LinUCB* is two orders of magnitude slower than its truly online counterparts (Perceptron-UCB and LAP).

With respect to performance comparison, the sub-problem with $q_{MAX} = 2$ is not really relevant. Indeed, since very few weights are active in the model, the problem is almost non-contextual which is why *UCB* achieves the best performance. Indeed, there is not much personalization to learn from the model and UCB does not even try do so, hence its success in this experiment. For the other two problems, the results are almost the same as with $F = 15$ except for one little detail: *LAP* slightly outperforms *LinUCB*. This is very surprising especially when we think that *LinUCB* takes more than 50 times more time to run than *LAP* which makes *LAP* better regarding both time *and* performance. *LAP*'s "victory" in this situation is not intellectually satisfying as *LinUCB* is supposed to be some kind of batch version of *LAP* and *Perceptron-UCB*.

We tried to find intuitive reasons for these results and one can be that this problem is much more noisy than the previous one and a Bayesian approach (such as *LAP*) is usually a good fit in this situation. Yet this is not convincing enough. The reader may remember that we added a regularization parameter λ to *LinUCB*, which is useful when the data is noisy. Section 4.4.1 shows this need for λ empirically on real data. We did not tune this parameter here because since we limited the tuning to one parameter, we kept the only one described in the literature. Moreover tuning the exploration parameter carefully is always crucial when taking about bandits. It is very likely that we would be able to find a value of λ for which *LinUCB* beats *LAP*. Nevertheless note that in our experiment *LAP* may not have achieved its very best performance either since it was given the same tuning opportunities. It was simply able to adapt better to a more noisy problem.

To be sure of the reason why *LAP* beat *LinUCB*, we plotted the average CTR of both algorithms over time and indeed Figure 2.7 clearly shows what actually happened. In the results from figure 2.5 that we already commented, *LinUCB* outperforms *LAP*. Yet it is very interesting to see on this new plot (fig. 2.7) that *LAP* takes a much better start and presents a

Table 2.2: Best parameter for each algorithm considered in section 2.6.5.

	UCB	*Perceptron-UCB*	*LAP*	*LinUCB*
$F = 15, MAX = 2$	$\alpha = 0.1$	$\alpha = 0.03$	$\beta = 10$	$\alpha = 0.1$
$F = 15, MAX = 5$	$\alpha = 0.1$	$\alpha = 0.1$	$\beta = 10$	$\alpha = 0.1$
$F = 15, MAX = 10$	$\alpha = 0.1$	$\alpha = 1$	$\beta = 10$	$\alpha = 0.3$
$F = 50, MAX = 2$	$\alpha = 0.1$	$\alpha = 0.001$	$\beta = 300$	$\alpha = 0.001$
$F = 50, MAX = 5$	$\alpha = 0.1$	$\alpha = 0.03$	$\beta = 30$	$\alpha = 0.003$
$F = 50, MAX = 10$	$\alpha = 0.1$	$\alpha = 0.3$	$\beta = 30$	$\alpha = 0.1$

typical logarithmic learning curve. On the contrary *LinUCB*'s curve oscillates at the beginning. This is typical of an algorithm overfitting the data and therefore of a lack of regularization. This confirms that a better choice of λ would have solved the problem. However it is very interesting to notice that these little oscillations are completely absent for *LAP*. In fact this makes a lot of sense as its model is updated and improved regularly in a truly online fashion. Moreover the magnitude of the update is limited by the amount of uncertainty, limiting the risks of early overfitting. On the contrary, *LinUCB* is much more sensitive to overfitting since it uses an exact optimization algorithm which is symptomatically very hard to tweak when only a little bit of data is available. So in fact when $F = 50$, as the model is more complex than for $F = 15$, *LinUCB* is outperformed as it suffers from this overfitting phenomenon at the beginning and does not have enough time to "catch up" in only $5,000$ time steps.

Finally, just for the record, the parameters that were automatically selected are given in table 2.2. They are not crucial but it is quite interesting to notice how they vary for each algorithm with the different parameters of the generative model.

Remark Also it would seem that there exists another way to speed up LinUCB that is not mentioned by Li *et al.* [7] than making it truly online (LAP, Perceptron-UCB) or than delaying the inverse computation as they proposed (Eco-LinUCB). The idea is to use the Sherman-Morrison formula [92, 93] (1949):

$$\left(M + uv^T\right)^{-1} = M^{-1} - \frac{M^{-1}uv^TM^{-1}}{1 + v^TM^{-1}u} \ .$$

It would yield an update in $O(F^2)$ instead of $O(F^3)$ (and $O(F)$ for LAP) without altering the performance. Studying the speed-up compared to the original formulation of LinUCB but mainly measuring the extent of LAP's speed-up compared to this new version would be very interesting. This is left for future work.

2.6.6 Discussion

To conclude this section, let us summarize the contribution we made. The evaluation protocol introduced by the challenge was too far away from the actual contextual bandit problem to let us draw any conclusion on the efficiency of *Ad predictor* compared to state-of-the-art methods. This is why we derived a similar but simpler algorithm, *LAP* which takes the same input as *LinUCB*, the main state-of-the-art contextual bandit algorithm. We then argued that *LinUCB* is *not* what is usually called an online learning algorithm as it computes the exact solution to the problem at each time step which takes quite some time. On the contrary, *LAP* behaves as an online learning algorithm would, that is it updates its model. As a consequence, it runs much faster which is confirmed by our set of experiments. This is the reason why we also compared *LAP* with another genuinely online learning algorithm that we also designed ourselves but only by gathering state-of-the-art methods: *Perceptron-UCB*.

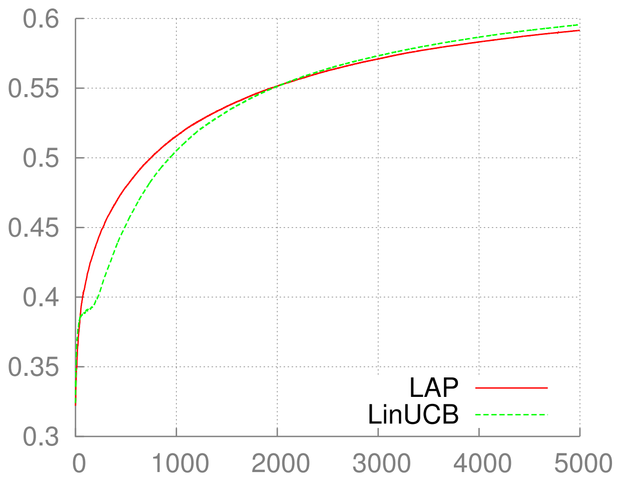


Figure 2.7: Average CTR of *LAP* and *LinUCB* over time for $F = 15$ and $q_{MAX} = 10$. Note than even though *LinUCB* ends up beating *LAP* as it was shown on figure 2.5, LAP clearly makes a better start. This explains *LAP*'s better performance when $F = 50$ as *LinUCB* does not have enough time to recover from this terrible start.

The empirical results show that *LAP* definitely improves the state of the art in terms of contextual bandit algorithms as it can be considered online in the strict sense. Indeed it always beats *Perceptron-UCB*, our state-of-the-art baseline. Moreover, *LAP* achieves performance that are very close to *LinUCB*'s. So close that we could almost say that this algorithm reaches the best of both worlds: speed and performance. It is important to note that a perfectly tuned *LinUCB* (that is to say with a carefully tweaked λ) would still achieve a better performance almost regardless of the setting. Yet *LAP* seems to be insensitive to overfitting, even with very little data, without being tuned which is an incredibly useful property. On the contrary, *LinUCB* can really perform extremely poorly with short horizons if not enough care is put into tuning regularization. Furthermore in any case, *LAP* is a perfect way to slightly decrease the performance in order to reduce the time complexity of the exact method - *LinUCB* - contrarily to the approach proposed in the literature: *Eco-LinUCB*. Indeed *Eco-LinUCB* does not even reach the performance of a non-contextual algorithm while still being much more computationally demanding than *LAP* or *PerceptronUCB*. Note however that an exact method exists to speed up LinUCB (which we did not try), based on the Sherman-Morrison formula, but the time complexity gain is less important.

To emphasize the point that *LAP* or *PerceptronUCB* are an important contribution even though their performance tends to be slightly lower than *LinUCB*'s, it is important to note that the time constraints of the challenge were based on real life applications and that it would have been impossible to even consider running a complete linear regression at each time step as what is done by *LinUCB*. Furthermore, our experiments considered at most a model with 50 numerical features to exhibit how heavy was the computations made by LinUCB. The dataset of the challenge has 100 numerical features and 20 categorical features. Categorical features cannot be used by LinUCB but we can easily turn each of them in several binary features. Their cardinality is between 3 and 25. Using binary code, these discrete features could each be transformed into 2 to 5 binary features. Yet this would involve dependence between these features which linear algorithms such as LinUCB or LAP cannot detect. To capture the full complexity of the discrete features into a linear model we may have to build one binary feature per possible value of each discrete feature. Therefore LinUCB, that already struggles with less than a hundred features, would have to handle something like 500 features in this case. An other example is the Yahoo! R6B dataset [12] that we study further in this document which has 136 numerical features, which is still much more than the 50 features we considered here.

2.7 Conclusion

We presented a Bayesian approach to dynamic recommendation. This approach led us to win the Exploration *versus* Exploitation challenge 2 (organized by UCL and ICML-2011). Based on a dataset provided by Adobe we have shown that, on this particular dataset, the click probability is much more related to the user than to the recommended item. Moreover this behavior is very stable over time. In fact this means that not all users are equal, and that some of them are much more willing to interact with recommended content whereas others simply ignore that content. This idea of identifying the best users may seem useless in the context of classical recommendation. Indeed, when a user is on a web page, we want to make a recommendation whoever he or she may be, even if that recommendation has a very low probability of being clicked. Nonetheless learning efficiently to identify "good" users might be crucial in applications very close to recommendation. In particular, in advertisement display people generally have more constraints than in recommendation. For instance an ad campaign typically has a click budget and a lifetime. Therefore displaying the best one at all time is not necessarily the optimal strategy. This issue is for instance tackled with a mixture of bandit algorithms and linear programming proposed by Girgin *et al.* [83]. It is clear that information on the raw click probability of the users would be very valuable in such a planning approach. Also in modern advertisement architecture, an advertiser needs to bid on users before being able to display some content to them. An accurate model of the "quality" of users is thus very important in that case so as to minimize the money lost in the auction process. See Chen *et al.* [94] for an example of what is called *Real Time Bidding* (RTB).

Let us not forget however that Ad Predictor was a contextual bandit algorithm before being turned into a user selection technique for the purpose of winning the challenge. Based on our experience during this challenge, we then proposed a generalization (GAP) of this algorithm which is able to deal with a big variety of datasets (discrete features, linear features, non-linear features that are approximated by linear-by-part functions, missing values). Such an adaptability is an interesting contribution in itself for the main state-of-the-art contextual bandit algorithm *LinUCB* does not possess it. Furthermore this algorithm is highly customizable thanks to its Bayesian nature. Indeed the priors can be used to insert any expert knowledge one may have into the model. Note that this many parameters could be considered a problem in theory. Yet we showed using experiments that even with little tuning, a good performance could be achieved. Another strength of such priors are their convincing power. Indeed, marketing teams who were historically in charge of recommendations, ad display or any other activity may be reluctant to simply replace there expert rules by a learning system they know nothing about. Similarly convincing the direction is always a challenge when proposing a new approach to a problem that involves money. With this algorithm, one can build features with the marketing team which would probably be very meaningful and let them set the priors of the corresponding Gaussian weights. This way the system would perform their expert decisions, at least at the beginning. Then, one could argue that, once online, the system would only drift from that expert model only if it finds better decisions to take. Possible changes can be monitored and understood by the marketing team and the direction. As a conclusion this algorithm can be trusted and understood by all, making interactions between the people involved in the system much easier. It significantly reinforces its usefulness compared to an algorithm that would require blind trust from both marketing experts and the direction. Such an algorithm would meet much more resistance and may even never be put online regardless of its potential.

The adaptability and transparency we just mentioned are great features for an algorithm. Yet they only allow structural comparison with other algorithms. As we may also be interested in performance comparisons, we also defined a simpler version of this algorithm, *LAP*, able to deal with continuous features the same say as *LinUCB* does. When *LinUCB* computes an exact solution to the optimization problem implied by the Exploration/Exploitation dilemma, *LAP* only performs an *online* approximation. We showed in this chapter that this tremendous gain of time did not decrease the performance very much. *LAP* is also better than the equivalent approximation methods that can be derived straightforwardly from state-of-the-art techniques (*PerceptronUCB*). In addition note that *PerceptronUCB*, that we designed and tested, possess an advantage that *LinUCB* and *LAP* do not have. Although its performance is not as good as *LAP*'s on a static problem, it is natively able to handle changes in the reward distributions.

Besides the contributions it allowed with respect to learning algorithms, the evaluation method of this challenge was also novel. It was based on an indirect evaluation method, that is based on a task close to dynamic recommendation but easier to evaluate which is its whole interest. Yet considering that we won by implementing an algorithm doing no exploration and more annoyingly no use whatsoever of the item we were supposed to recommend it is quite obvious to conclude that the evaluation protocol suffers from a huge bias. Although we discuss in the conclusion of this document (page 205) how some ideas could be recycled, this bias makes the method of no practical use as it is now. In fact this conclusion is the starting point of the main piece of research of this thesis whose topic is precisely contextual bandit *evaluation*. Indeed at the beginning of this work, we were interested in designing bandit algorithms that are not only good theoretically but that are both exhibiting nice performance on real life problems and able to handle dynamic environment (changing pool of items, evolution of user tastes, CTR decay *etc.*). This challenge left us with disturbing questions. For instance how are we supposed to demonstrate that we manage to design a practical algorithm without a proper evaluation method based on real data? Also designing meaningful indirect tasks for online learning seems too challenging. Is it thus possible to design direct methods for dynamic recommendation ?

As a matter of fact, we started working on these questions very quickly. At the end of the article in which we published our winning approach to the challenge [5], we were talking about a new approach based on classification supported by interesting theoretical guarantees. Yet, although the task we introduced was much closer to recommendation, it can still be considered as slightly indirect. We however describe this method and provide practical applications in appendix B. We also use it in chapter 4 as a diagnostic tool. Interestingly however, around the time of the challenge (2011) Li *et al.* [11] published an article popularizing a new *direct* evaluation method by advertising its *unbiasedness*. Note that this method is inspired from a work by Langford *et al.* [10] in 2008. A direct unbiased method is exactly what we were after. Nonetheless given its very young age, the literature studying it both theoretically and empirically is quite limited. Indeed from the theoretical point of view, even the very notion of unbiasedness remains rather vague. Also, as often in applied mathematics, the theoretical framework makes assumptions that are not true in practice. Finally some experiments that seem promising are described but they are not very thorough and are not reproducible, being based on private data. This is why we decided to study this method deeper. We first provide a theoretical analysis of many of its statistical properties in chapter 3. We then take advantage of our victory in the challenge to organize a new one based on that method so as to study its empirical properties and find its possible limitations (chapter 4.

Chapter 3

The replay method: an "unbiased", offline, data-driven evaluation method of contextual bandit algorithms

Abstract of the chapter *This chapter is about the so called unbiased evaluation method introduced by Langford et al. [10] and analyzed deeper by Li et al. [11] in the precise context that we are interested in. Here, we start with briefly recalling the main results from these two papers. However their analysis leaves the notion of unbiasedness quite obscure. This is why in this chapter we make things clear by proving that the method is actually slightly biased in the main theoretical framework that they consider but that this bias can be lifted at the cost of a small loss of accuracy. We also show that the previous analysis of the expected error was quite coarse and find a way to significantly sharpen the analysis. Finally we exhibit some limitations of analyzing the accuracy of such a method with concentration inequalities before overcoming them with a variance analysis.*

Keywords Offline Evaluation - Contextual Bandits - Concentration Inequalities - Bias - Variance - Replaying past data.

Contents

3.1 Introduction

This chapter is a presentation followed by an in-depth analysis of what we call the *replay method*. This method is an answer to what we were wondering at the end of the previous chapter: is it possible to alleviate the inherent bias of an indirect method, or best to derive a direct method evaluating a learning contextual bandit algorithm with real data? This method is a direct method as it allows to evaluate a contextual bandit algorithm against a contextual bandit problem. It was first introduced to the best of our knowledge by Langford *et al.* in 2008 [10]. In a following work from the same group at Yahoo! research, Li *et al.* [11] analyze the method further. Li *et al.* [11] claim in the title of their article that the *replay method* is "unbiased" which is most likely what popularized it. Another very nice feature of that method is its incredible simplicity and thus its accessibility to as many people as possible. Yet despite its simplicity, the replay method is quite revolutionary. Indeed if it meets all its promises it will significantly broaden the community of researchers having the possibility to work on application-driven contextual bandit algorithms, which should greatly benefit research and knowledge in this field. As a side effect, recommender systems performing dynamic recommendation should get much better very fast, taking advantage of the incoming amount of new research.

Nevertheless, although very promising and interesting theoretical results are presented in the body of the paper, Li *et al.* [11] leave the very notion of unbiasedness pretty vague. Also very few experiments are provided. Furthermore these experiments were made on private data, making them non-repeatable, non-extendable and as a consequence not very reliable. To provide the bandit community with a full knowledge of both the theoretical and empirical properties of this revolutionary tool, we conduct in this work a thorough study of the replay method that highlights its strength but also its limitations. This way people will know what they are doing when they evaluate empirically their new bandit algorithms or build a dynamic recommender system. Another goal of this complete analysis is to promote further this exciting tool and to encourage the design of application-oriented bandit algorithms. Indeed, by lack of means to evaluate them, such algorithms are extremely rare within the literature. In this chapter we focus on the theoretical analysis of the method in the framework introduced by Li *et al.* [11]. The empirical study is left for chapter 4. Together these two chapters are the first of the two main sets of contributions of this thesis work that we introduced at the end of chapter 1. Both are about the evaluation of contextual bandit algorithms based on real data.

In this chapter, after stating the very promising results from the two aforementioned founding papers [10, 11], we investigate further the unbiasedness of the replay method. Without going into the details in this introductory part, we show that unbiasedness, however true in one specific unrealistic framework, can be considered to be quite an overstatement. We then conduct an analysis of this method's bias in the realistic framework introduced by Li *et al.* [11]. As odd as it may seem, this bias analysis is absent from the literature. We pursue this analysis by introducing a truly unbiased variant of the replay method. This unbiasedness is obtain by a slight and very intuitive reduction of accuracy.

This bias analysis is followed by an improvement of the state-of-the-art concentration bound and a discussion on the limited range the result in general. In particular it does not communicate the right intuition with regard to the accuracy reduction we just mentioned. As a consequence we refine the analysis of the deviation of replay from the value it estimates (*i.e.* its error) with a different approach: a variance analysis.

The analysis of replay's statistical properties is the main contribution of this chapter. After that, we tackle a few practical considerations. In particular the last significant contribution of this chapter is a study of an extension of replay that aims at relaxing assumptions that are made on the data. Indeed, throughout this chapter, the assumption that the data is logged by a random uniform policy is made. Ways to relax that assumption have already been studied in the literature. Without going into the detail, we show here that relaxing it is possible in some cases and in particular in our simplified theoretical framework. Yet the contribution we make is to prove that this relaxation is much more difficult to make or even impossible in the general case than what it looks like. We derive new theoretical results justifying that. We conclude this chapter by reviewing a few tricks that should be kept in mind when some assumptions of the analysis are not met in practice.

3.2 Basic idea

The main point of this chapter is to fully understand the strengths and weaknesses of the replay method so before anything else let us begin from the beginning and describe how this method works. The *replay method* is not very complicated and its implementation does not need a lot of explanations. It takes as input an algorithm to evaluate and a dataset of interactions with users of an application. An interaction is simply a triplet (context, recommended content or action, reward) that we denote (x, a, r). Obviously given an interaction we have no knowledge of what would have happened had another action been performed. We say that the data is partially labeled since for each interaction, only the label (reward) corresponding to the performed action is known. The others are unknown. This is without doubt problematic when it comes to evaluating an algorithm that will necessarily want to perform non labeled actions. With *replay*, the basic idea to overcome the fact that only one Kth of the data is labeled is to only use one Kth of it by way of a simple rejection mechanism. In a few words, we scan the dataset in a chronological order (hence the replay terminology) and then for each record (x, a, r) we:

- Ask the evaluated algorithm for a recommendation in context x.
- Check whether it is the same as the one in the dataset (a).
- If so we reveal the outcome (that is the reward r which can be an explicit rating as in Netflix or a click/no click as in news or video recommendation). The algorithm can then update its reward model.
- Otherwise nothing happens and the record *is not* taken into account at all for the evaluation. The learning algorithm obviously cannot learn from this record.

The evaluated CTR is naturally given by the sum of *revealed* rewards divided by the total number of revealed outcomes. This method is as simple as that, which is its first strength.

Algorithm 10 is the detailed implementation of this method using the notations given in section 1.4. A few additional notations that we will use in the remainder of the chapter are also defined and justified there. From there on we will dissect this method by studying among other things the data it can use, the assumptions it requires to be made, its limitations, its bias, its variance *etc.* Indeed, in spite of its genuine simplicity, the replay method has surprisingly nice properties given that a few assumptions are met. Note as a reminder that the notations used to measure the performance of an algorithm and its estimation are defined in section 1.4.3.

Algorithm 10 *Replay method* (introduced by Langford *et al.* [10], analyzed by Li *et al.* [11]) on a dataset S acquired via uniformly random interactions with a contextual bandit distribution $\mathcal{D}$.
Remark: In order to be mathematically rigorous, we use a history of events h_t which is the list of triplets (x, a, r) from S that the method did not reject before time t. Therefore an algorithm is a function that maps a history of events to a policy, which is a function mapping a context to an action. Note that this way of writing the algorithm that Li *et al.* use as well is very mathematical. The goal is to avoid hiding information in A. Indeed such an object with an internal state would be misleading and difficult to handle within theoretical analysis. However a more "software oriented" implementation (see figure 4.3) would use an object representing the algorithm which would be asked to *choose* actions given contexts and to *update* its model (internal state) based on the outcome. Comments in the pseudo-code are given to picture where those functions would be called on A.

```
Input: A contextual bandit algorithm $A$ and a dataset $S$ of $T$ triplets $(x, a, r)$
Output: An estimate of $g_A$
  $h_1 \leftarrow \emptyset$
  $\widehat{G}_A \leftarrow 0$
  $V \leftarrow 0$                                  ▷ Counts the number of evaluations.
  for each $t \in \{1 .. T\}$ do
      $\pi \leftarrow A(h_t)$
      if $\pi(x_t) = a_t$ then                       ▷ $\approx$ choosefunction
          $h_{t+1} \leftarrow h_t + \{(x_t, a_t, r_t)\}$      ▷ $\approx$ updateprocedure
          $\widehat{G}_A \leftarrow \widehat{G}_A + r$
          $V \leftarrow V + 1$
      else
          $h_{t+1} \leftarrow h_t$                       ▷ We do nothing, the history does not change.
      end if
  end for
  return $\frac{\widehat{G}_A}{V}$
```

Remark Before going any further, let us briefly comment on the name we gave to this method. The reason why we called it *replay method* is very simple and does not come from intense thinking. In the paper featuring it, its author did not name it but often referred to it as "a replay methodology" because the data was acquired against the real world and then meant to be reused or replayed in a different way to try to evaluate different policies. However it is important to note that the *replay method* is not the only way to "replay" a dataset. The indirect method proposed in the Exploration *vs.* Exploitation challenge 2 presented in chapter 2 is one particular example. The variant of replay we introduce in this chapter is another one. We could also find many different ways to replay a dataset with various purposes that could be something else than evaluating a policy. Nevertheless, when we refer to *THE replay method* in this document, we will always mean the method implemented by algorithm 10 and introduced by Langford *et al.* [10].

3.3 On the data that can be used

Before diving into the heart of the subject, let us provide a few definitions. It is obvious that to be able to evaluate contextual bandit algorithms on real data, we need to acquire some data from the application we are interested in. This can only be done by interacting with this application by way of a policy or a contextual bandit algorithm. Let us recall that what we call a policy does not change over time, that is at any time step it always chooses the same action in a given context. We may also call them static policies to emphasize the fact that they do not evolve. A policy can be either deterministic or stochastic. A deterministic policy is a simple function that maps contexts to actions. Given a context a stochastic policy can perform several actions but each one of them has a fixed probability of being performed. On the contrary a contextual bandit algorithm can change its behavior at each time step. More formally, we view it as a function that maps the history (the list of past interactions) to a static policy (possibly stochastic). In this work, the policy that interacted with the application in order to acquire the dataset is referred to as the logging policy and will be denoted by π_{LOG}. We recall that a contextual bandit game $\mathcal{B}$ is defined by a horizon T (optionally) and a distribution $\mathcal{D}$ that generates couples of the form $(x, \vec{r})$, where x is some contextual information on which no assumption is made and $\vec{r}$ a vector of K real-valued rewards (one per action). No assumption is made on the reward distributions either. At times we may restrict the reward distributions to Bernoulli distributions for simplicity but it is never a necessary conditions for things to work both in practice and theory. Finally a dataset S is a simple ordered list of T interactions $(x_t, a_t, \vec{r}_t[a_t])$, where a_t is the choice made by π_{LOG} at time t ($\pi_{LOG}(x_t) = a_t$) and $\vec{r}_t[a_t]$) is the reward corresponding to that action. Indeed we do not get to see the entire vector hence the need for that rejection mechanism in the algorithm (see algorithm 10).

For more thorough notations and vocabulary on contextual bandits and their evaluation on real data, the reader is referred to section 1.4.3.

3.3.1 Necessary conditions

We will make several assumptions in this chapter to be able to analyze the replay method. The assumption on the data or more precisely on the logging policy is the most important as it is not only an assumption we make in order to make proofs but also a requirement for the method to actually work in practice. This assumption is the fact that the logging policy selects at each time step an action uniformly at random from $\{1..K\}$. We will make this assumption throughout this chapter. This assumption may seem quite extreme but it can be considerably weakened. Yet we keep the random uniform assumption for simplicity of exposition and because doing otherwise does not deepen the understanding of the method. On the contrary, it makes the notations and the theoretical results significantly heavier and more obscure. Nonetheless let us mention that for the replay method to work it is in fact only necessary that it respects the three following conditions.

- The logging policy does not actually need to be stochastic nor static (in that case it is not a policy as we defined it but a general algorithm). Yet the first condition is that each possible item has to be recommended at least a fair amount of time. Indeed if an item is not recommended when the dataset is acquired, it is impossible to use that dataset to evaluate without bias an algorithm recommending this item. Moreover if an item is only recommended a very little fraction of the time, it will introduce a lot of variance in the evaluation process.

- The logging policy is independent from the context. This is needed for two reasons. First if the logging policy depends on the context then it is possible to design an algorithm which would learn the behavior of the logging policy and take advantage of it. For example it would be possible to avoid being evaluated when a context with low probability of click (*i.e.* a user that never clicks) is proposed by choosing an action that the logging policy would not choose. One may think that it is not a big deal as one trying to really improve one's recommender system would not cheat during the evaluation process as it would only lead to bad surprises. Yet an optimization algorithm could typically do so without us noticing it. The second reason why the logging policy should be independent from any contextual information is that combined with the first condition (all the item are recommended at least a fraction of the time) we mentioned, it ensures that all the space $\mathcal{X} \times \{1, K\}$ is explored and thus that any recommendation algorithm can be evaluated. If it is not the case, some items will never be recommended to some specific areas of the context space $\mathcal{X}$ in the dataset so a recommendation algorithm trying do it will not be fairly evaluated.

- The last condition for the evaluation policy to work is obviously that the logging policy is known.

These conditions are the one presented in the literature [10]. The first two ones can be replaced by another condition. The logging policy needs to be stochastic and at each time step, all actions need to have a strictly positive probability of being performed. Obviously this policy still needs to be known. This case is reviewed in section 3.6.1.

3.3.2 Impossibility result

To justify why we need the aforementioned assumptions about the data, let us mention an impossibility result that was proved by Langford *et al.* [10] in the case where the second condition is not met (*i.e.* π_{LOG} is context dependent). This result states that it is impossible to assert that a given statistic about a given policy can be computed using a dataset, even if we know that the logging policy plays all the actions in equal proportion.

In other words, to illustrate the importance of the second condition mentioned above, it is easy to design a very simple example in which all the items are recommended in equal proportion by the logging policy but for which the resulting dataset does not allow the evaluation of some recommendation policies.

Such an example can be constructed as follows. Let $\mathcal{D}$ be a contextual bandits distribution with two arms (or actions) and a context set $\mathcal{X}$ composed of only two distinct elements 0 and 1. For all t, $\vec{r_t} = (0, 0)$ - which means that both actions yield a reward of 0 - if the context is 0 and $\vec{r_t} = (0, 1)$ - which means that action 0 yields a reward of 0 whereas action 1 yields a reward of 1 - if the context is 1.

Provided that the contexts are equiprobable, a logging policy defined by $\pi_{LOG}(i) = i$ performs both actions in equal proportions. A dataset logged by this policy would look like that:

context	0	0	1	0	0	1	1
action	0	0	1	0	0	1	1
reward	0	0	1	0	0	1	1

...

Let another policy be defined as $\pi(i) = 1 - i$. It is easy to see that π would have a CTR of 0 on $\mathcal{D}$. However it would be impossible to use a dataset logged by π_{LOG} to have the slightest idea of any characteristic of π's CTR. Indeed it only contains context-action couples of the form $(0, 0)$ or $(1, 1)$ whereas π performs action 1 in context 0 and action 0 in context 1. It is then possible to formulate the following theorem which is a more intuitive way of writing the first theorem from Langford *et al.* [10] using our notations:

Theorem 1. *There exist a policy π, a policy π_{LOG}, a contextual bandit distribution $\mathcal{D}$ with two contexts and only two actions that π_{LOG} performs in equal proportions such that it is impossible to get any statistic on π using any dataset S logged by π_{LOG} on $\mathcal{D}$.*

Proof. A proof was provided by constructing the aforementioned example. □

Finally, seeing this example one may now presume that in fact the condition that the logging policy is independent from the context could be replaced by a simple extension of the condition on the allocation of the actions. Indeed, one may think that it is enough for the logging policy to recommend a sufficient number of times each item to each context. Of course this is true if the context space is discrete. However the context space is often continuous or its cardinality is much larger than any reasonable dataset. For example in the Yahoo! R6B dataset [12], $\mathcal{X} = \{0,1\}^{136}$ so $|\mathcal{X}| = 2^{136} = 8.10^{40}$ while the dataset contains a few tens of millions of interactions ($< 10^8$).

So given that a logging policy explores all the pool of actions, to make sure that $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}, P(a|x) > 0$ and thus that *any recommendation policy is evaluable* (whatever it may mean), it has to meet one of the two following conditions.

1. The logging policy is independent from any contextual information.

2. The logging policy is stochastic (possibly contextual) and has a non-zero probability to recommend each item given any possible context.

This fact is independent from the evaluation method used. In general, if one of the two conditions are not respected, it is possible to find policies for which no statistic at all are computable (theorem 1).

The main reference we use in this chapter, that is the article by Li *et al.* [11], considers a randomly uniform logging policy, which is the ideal scenario in the general case. This is what we will do here as well. Moreover the dataset that we use in this thesis work to run experiments, the Yahoo! R6B meets this requirement. Yet we will briefly talk about how to relax this assumption in section 3.6.1 without altering the results about bias that we are about to discuss.

3.4 Unbiasedness

In this section we discuss the most important and strongest result about the *replay* method for it applies to all contextual bandit algorithms. It was more or less sketched by Langford *et al.* in 2008 [10] but stated as such by Li *et al.* in 2011 [11]. This results is about unbiasedness. The only real condition for it to be true is that we are dealing with a single static contextual bandit problem $\mathcal{D}$. It may not be true in practice (we have already discussed the dynamic nature of news recommendation) but since it is always possible to consider that $\mathcal{D}$ is static by parts, this is a very strong result.

Here is the precise formulation of the corresponding theorem as it is formulated by Li *et al.* [11] (Theorem 1).

Theorem 2. *For any contextual bandit distribution $\mathcal{D}$, any contextual bandit* ***algorithm*** *A, any dataset S of* ***infinite*** *length containing i.i.d. interactions acquired using a uniformly random logging policy on $\mathcal{D}$ and all positive integers T we have that all the histories of interactions h_T have the same probability whether A is played against $\mathcal{D}$ or evaluated using the replay method on S:*

$$\forall h_T, \; \underset{A,\mathcal{D}}{\mathbb{P}}(h_T) \;=\; \underset{replay(A,S)}{\mathbb{P}}(h_T).$$

In other words, a list of T interactions $h_T^{(real)}$ of A with the real world (history) is a random variable conditioned on A and $\mathcal{D}$. A list of T non rejected interactions $h_T^{(replay)}$ produced by replaying A on S is also a random variable (conditioned on $\mathcal{D}$, π_{LOG} and A). These two random variables are equal in distribution:

$$h_T^{(replay)} \;\overset{d}{=}\; h_T^{(real)}.$$

Note that the first part of the theorem is more or less how it is stated by Li *et al.* [11] whereas the second part is a rewriting to emphasize the full extent of the result. Basically this theorem says that the probability of a history of interactions seen by an algorithm is the same whether it is played in the real world (on $\mathcal{D}$) or it is evaluated by the replay method on a dataset S ($replay(S)$). This may seem like nothing but it actually means that a learning algorithm behaves (recommends, learns *etc.*) exactly as in the real world. No difference can be made. As a consequence in this particular setting, replay is totally unbiased. Moreover it is important to note that contrarily to the following results, the theorem is true for any algorithm, not only static policies. This is what makes this result so strong and significant. This is also a particularly interesting result in the context of dynamic recommendation as what we want to do is to measure the ability of an algorithm to learn.

The infinite length condition is only here to make sure that the *replay method* does get to see T events. Indeed for each recommendation logged in the dataset, the probability that an algorithm chooses the action logged (and thus gets evaluated) is equal to $1/K$ as the action was logged by a random uniform policy. Therefore, whatever the size of S, as long as it is finite there is no way to tell with probability 1 that the algorithm evaluated will see T recommendations.

However using the theorem we can assert that the estimator given by the *replay method* is *asymptotically* unbiased for any evaluated algorithm. Moreover with an infinite dataset, it is possible to build an unbiased estimator of $g_A(\mathcal{D}, T)$ for all algorithm A and all horizon T. Moreover we mention two interesting property. When S is finite we have no way to tell *a priori* whether we are able to produce T replayed interactions. Yet *a posteriori*, once T interactions are simulated on a dataset S of any length, we can say that they are an unbiased sequence of T interactions. Therefore an estimator of $g_A(T)$ based on that generated sequence is unbiased. Again this can only be said *a posteriori* which is far from being a classical statistical guarantee. Also it is trivial to provide an upper bound valid with high probability on the size of the dataset needed to replay T interactions. Li *et al.* [11] proved that with probability at least $1-\delta$ a dataset such that:

$$|S| \leq 2K\left(T + ln(1/\delta)\right)$$

insures that T interactions can be replayed. As a consequence, we can say that in addition of being asymptotically unbiased, replay is unbiased with high probability, although it is not the kind of result people are generally after.

Finally note also that in the finite case, the estimator of the CTR of an algorithm given by the *replay method* on a dataset of length T is an estimator of the CTR of this algorithm played for T/K iterations in the real world: $g_A\left(\mathcal{D}, \frac{T}{K}\right)$. Good empirical results on the accuracy of this method in non asymptotic settings are provided in the literature [7, 11]. Nonetheless from a theoretical point of view, the literature does not provide any further study about the possible bias of this method in a finite time setting. This is very unfortunate given that infinite datasets do not exist. Furthermore the reader should keep in mind that web-oriented datasets, although very big, are dynamic. Therefore they can, at best, be considered as static by part and these parts definitely have a finite and even modest size. In what follows we propose to right this wrong.

3.5 Finite datasets

This section studies the theoretical properties of the replay method and gathers the main contributions of this chapter: a bias analysis, the introduction of an unbiased method, a refined concentration bound and a variance analysis.

Our first concern is the bias of replay of finite datasets. The only mention of the unbiasedness of the *replay method* for a finite dataset by Li *et al.* [11] is quite vague. They say that the output given by the replay method run on a finite dataset *”may not be an unbiased estimate of the true* *per-trial payoff of A"* and as a matter of fact it is biased. As the bias of the evaluation method used in the previous edition of the challenge (see chapter 2 or even Nicol *et al.* [5]) caused very surprising results, we decided to further investigate this issue. It seemed all the more important to us that even though the stream of users on the web may seem almost infinite, the problem is dynamic (changing pool of items, changing user behavior...). Indeed we may consider that what we actually need is to evaluate algorithms on sequences of finite-size datasets. In the literature [10, 11], a setting with a static policy is considered in order to prove convergence bounds. Indeed, nothing can be proved with that regard in the general case but we come back on that point in this section. As it was not done before, in this section we investigate the exact bias of the *replay method* in this setting and show that it can be made unbiased. Then we discuss further its convergence.

3.5.1 Bias

First let us clarify things about the bias of the estimator engendered by the use of *replay* on a finite dataset. Before anything, let us also define an additional notation: $V_t = I(\pi(x_t) = a_t)$, $\dot{I}()$ being the indicator function. In other words, V_t is the random variable indicating whether π chooses the same action as the one in the dataset at iteration t. Note that it depends on a context x_t drawn from $\mathcal{D}$ and the action picked by the uniform logging policy ($a_t \sim \mathcal{U}(\{1, K\})$).

To study the replay method with precision, we need to define a convention. A useful quantity is the number of interactions simulated by replay (*i.e.* the number of non rejected records):

$$\sum_{t=1}^{T} V_t \ .$$

For all T, this quantity has a strictly positive probability of being equal to zero. In that case, replay outputs zero by convention. Other conventions would slightly impact the results we are about to present. Yet the modifications would not have the faintest significance while making the results heavier, hence this choice of convention.

Theorem 3. *Given a contextual bandit problem $\mathcal{D}$, a static policy π, a dataset S of size T containing i.i.d. recommendations acquired using a random uniform logging policy on $\mathcal{D}$, the estimator defined as follows:*

$$\hat{g}_\pi^{(replay)}(S) = \begin{cases} \frac{\sum_{t=1}^{T} V_t . r_t}{\sum_{t=1}^{T} V_t} & when \ \sum_{t=1}^{T} V_t > 0 \\ 0 & when \ \sum_{t=1}^{T} V_t = 0 \end{cases}$$

which is outputted by the replay method is a biased estimator of $g_\pi(\mathcal{D})$. This bias, which can be quantified as follows:

$$\mathbb{E}\, \hat{g}_\pi^{(replay)} - g_\pi = -g_\pi . \left(\frac{K-1}{K}\right)^T \ ,$$

goes to zero exponentially fast as T grows.

This theorem, which is proved in appendix F.1 basically says that the *replay method* has a bias when evaluating a static policy on a finite dataset. However it is asymptotically unbiased and that bias goes to zero exponentially fast as the size of the dataset grows.

3.5.2 Replay*: a truly unbiased estimator

We showed in the previous section that even though it is small, the replay method has a theoretical bias when used on finite datasets. Yet it is straightforward to come up with an unbiased estimator which would be the result of the very same methodology with just one modification: at the very end, instead of dividing by the total number of evaluations ($\sum_{t=1}^{T} V_t$) (see algorithm 10) we divide $\widehat{G}_A$ by T/K which is the expected value of the previous quantity. That way the bias vanishes as it is stated in the following theorem. We call this slightly different method the *replay* method.*

Theorem 4. *For any contextual bandit problem $\mathcal{D}$, any* static *policy π and any dataset S of finite length T containing i.i.d. interactions (x_t, a_t, r_t) acquired using a random uniform logging policy on $\mathcal{D}$ we have that the estimator outputted by the replay* method:*

$$\frac{\sum_{t=1}^{T} V_t . r_t}{T/K}$$

is an unbiased estimator of $g_\pi(\mathcal{D})$.

Note that the bias of both methods could also be quantified for the evaluation of non-stationary algorithms. The results would be much heavier, involve assumptions on the nature of the algorithm and provide more or less the same intuition. As such we do not believe that they would be very useful in general. A significant detail however would be that the *replay-method* (resp. *replay* method*) would be a biased (resp. possibly unbiased) estimate of the CTR of the algorithm for $\frac{T}{K}$ interactions with $\mathcal{D}$ and not T, the size of the dataset.

Finally, it is common knowledge that sometimes systematically looking for theoretically unbiased estimators can lead to reducing the convergence properties of the estimations. The reader interested in finding example of this should read Richard Bellman's severe criticism of the systematic search for unbiased estimators in *Dynamic Programming* [95]. He illustrates the thought with the poem of a young Indian, Hiawatha, who is mocked by his fellow hunters for using a biased estimate of the position of the target when shooting arrows. Nevertheless contrarily to the arrows of his peers, all of Hiawatha's arrows end up in the center of the target. The next section is meant, among other things, to study the convergence properties of the replay method and its unbiased counterpart: the replay* method. Intuitively, replay should be more accurate that replay* for we divide by the actual number of evaluations. As a consequence, Bellman should find another argument to support his argumentation against the systematic research of unbiased estimators.

3.5.3 Concentration result

The real point of considering the setting with a fixed policy comes here. Indeed, even though it would be possible to derive some (possibly weaker) unbiasedness or to at least quantify that bias for general algorithms, it is actually impossible to prove any convergence results in the general case, hence our consideration of this simpler setting. This is demonstrated by the impossibility result presented in the next paragraph.

In this section we also study the actual convergence of both the *replay method* and the *replay* method.* The intuition tells us that even though the *replay method* is biased as opposed to the other one, it should converge slightly faster as we divide the sum of the evaluations by the actual number of evaluations instead of its expected value $\frac{T}{K}$ as with the *replay* method.* The theory should be able to demonstrate that fact. The previous analysis of the *replay method* is too coarse and naive to achieve that purpose so we start with sharpening it to be able to compare it with the *replay* method.* Finally we will see that when it comes to comparing the results, things are not as simple as they might have seemed.

Impossibility result

It can be easily shown that in the general case, that is without any assumption on the nature of the evaluated algorithm, it is impossible to get any convergence result. This impossibility can easily be justified using an example designed by Li *et al.* [11]:

Let us consider a contextual bandit distribution $\mathcal{D}$ with two actions and two contexts with some different CTRs depending on the various possible associations of contexts and actions. Let us consider as well a policy π such that $\pi(x) = x_1$. It means that π will always and only choose action 1 if the first context was context 1 and it will always choose action 2 if the first context was context 2. This way all the sequences of interactions will only depend on the first context and no matter how T - the size of the dataset - grows the estimated CTR will never get any closer (in probability) to the average CTR of π. Note that this is not an artifact of replay. Indeed such a policy, if evaluated against the real world, would not concentrate with the number of interactions either.

A naive previous result

What we want to do here is to bound with high probability the deviation of the estimator from the true CTR of the estimated policy. As a quick reminder, here is what the estimator looks like:

$$\hat{g}_\pi = \frac{\sum_{t=1}^T V_t.r_t}{\sum_{t=1}^T V_t}.$$

Note that in this part about concentration bounds, we consider Bernoulli distributed rewards. This is not a requirement for the method to converge in theory or in practice. We could extend the following results to real valued, bounded rewards regardless of their distributions without effort. The only reason why we consider Bernoulli rewards is that this is what Li *et al.* [11] considered in their work. This assumption allows us to compare our results with what they did but we get rid of it in the variance analysis in section 3.5.6.

In their article [11], Li *et al.* improved the work done by Langford *et al.* [10] about the replay method's convergence. The technique they used to bound the estimator is quite simple. They bound separately the numerator and the denominator in order to bound the ratio. This would be the way to go were the two quantities independent but the fact is that they are clearly not. Indeed the more we sum rewards in the numerator, the bigger the denominator becomes.

Li *et al.* [11] get a bound which looks like what follows. With probability $1-\delta$, the error of estimation of g_π made by the replay method is bounded by:

$$\frac{\sqrt{(g_\pi^2 + g_\pi)3Kln(\frac{4}{\delta})}}{\sqrt{T} - \sqrt{3Kln(\frac{4}{\delta})}} = O\left(\sqrt{\frac{g_\pi Kln(\frac{1}{\delta})}{T}}\right).$$

Note that the term $\sqrt{3Kln(4/\delta)}$ disappears because as T grows it becomes negligible in front of $\sqrt{T}$ from which it is subtracted. The reader may also have noticed the disappearance of g_π^2. This simply comes from the fact that g_π is between 0 and 1 so $g_\pi^2 \leq g_\pi$. The rest of the transformations are constant removals.

To allow comparison with our work that follows and get a better sense of the structure of the bound, we can simplify the raw bound in a more precise fashion.

$$\begin{aligned}
\frac{\sqrt{(g_\pi^2 + g_\pi)3Kln(\frac{4}{\delta})}}{\sqrt{T} - \sqrt{3Kln(\frac{4}{\delta})}} &= \sqrt{\frac{(g_\pi^2 + g_\pi)3Kln\left(\frac{4}{\delta}\right)}{T}} + \frac{3Kln\left(\frac{4}{\delta}\right)\sqrt{g_\pi^2 + g_\pi}}{T - \sqrt{3Kln\left(\frac{4}{\delta}\right)}} \\
&= \sqrt{\frac{(g_\pi^2 + g_\pi)3Kln\left(\frac{4}{\delta}\right)}{T}} + O\left(\frac{g_\pi Kln\left(\frac{1}{\delta}\right)}{T}\right).
\end{aligned}$$

The first line can easily be proved by subtracting one side from the other. The second line is a straightforward simplification. The main term of the bound appears clearly along with all the corresponding constants. It is followed by a smaller order term that we do simplify for it does not matter much. Note also that here, $g_\pi^2 + g_\pi$ appears. To get a sense of what it means in practice for the value of the bound, let us take an example. For a typical CTR of 0.1, g_π^2 is only three times smaller than g_π. Thus it cannot be considered as totally negligible.

A tighter bound

By bounding separately the ratio of two random variables that increase and decrease together, we obviously get a bound which is quite loose. To do better we can use our intuition about the replay method. This method gives low accuracy results when it gets unlucky that is when $\sum_{t=1}^{T} V_t$ is small. This is why in order to obtain a better bound we can try to lower bound this quantity and then notice that if we fix $\sum_{t=1}^{T} V_t$ to some value N, the estimator is in fact a basic estimator of the expectation of g_π when drawing N i.i.d. samples (and indeed the smaller N is, the less the results are concentrated). The bound we get, as it is detailed in the following theorem, is of the same order of magnitude as the previous one but cleared of the dependence on g_π^2 in its highest order term.

Theorem 5. *For any contextual bandit problem $\mathcal{D}$ with Bernoulli rewards, any* static *policy π and any dataset S of finite length T containing i.i.d. interactions (x_t, a_t, r_t) acquired using a random uniform logging policy on $\mathcal{D}$ and for any δ in$]0,1[$ we have that, with probability $1-\delta$, the error of the estimator outputted by the replay method:*

$$\frac{\sum_{t=1}^{T} V_t.r_t}{\sum_{t=1}^{T} V_t}$$

is bounded by:

$$\sqrt{\frac{3Kg_\pi ln(4/\delta)}{T - \sqrt{2KTln(2/\delta)}}} < \sqrt{\frac{3Kg_\pi ln\left(\frac{4}{\delta}\right)}{T}} + O\left(\left(\frac{Kg_\pi ln\left(\frac{1}{\delta}\right)}{T}\right)^{3/4}\right) .$$

To be perfectly clear, the bound we actually prove is on the left. The decomposition on the right is proved by upper bounding and thus slightly loosening the previously proved bound. The goal of the decomposition is to highlight that the higher order term of this bound is much smaller than the one of the literature. As a consequence, it is obvious that this new bound is tighter as T grows since the highest order term is much smaller. In fact it is tighter for any value of T, even small ones. We can get intuition justifying that by having a look at the raw bounds. The leading term of the new bound is definitely smaller. Yet, the quantity subtracted from T in the denominator depends on $\sqrt{T}$ in the new bound which was not the case before. Nevertheless that quantity remains very small compared to T. Besides this intuition, we do not provide analytic evidence that the new bound is tighter than the former one for small values of T. Rather we plotted the raw bounds (see fig. 3.1) to confirm that this new bound is really better than the state-of-the-art one. Note the significant difference between our bound and the state-of-the-art one proved by Li *et al.* [11]. The reader will notice a third curve which corresponds to replay*. We discuss this bound after proving it in the next section. Anyway, this third curve can also be interpreted as the highest order term from our new bound for replay. This shows how small the second order term is in comparison.

The sketch of the proof - which is provided in appendix F.3 - is quite simple and follows exactly what we said previously. We lower bound in probability $\sum_{t=1}^{T} V_t$, that we denote N, using a result derived from the Chernoff bound. We then use the actual multiplicative Chernoff bound and the previous lower bound to bound the error of the estimator.

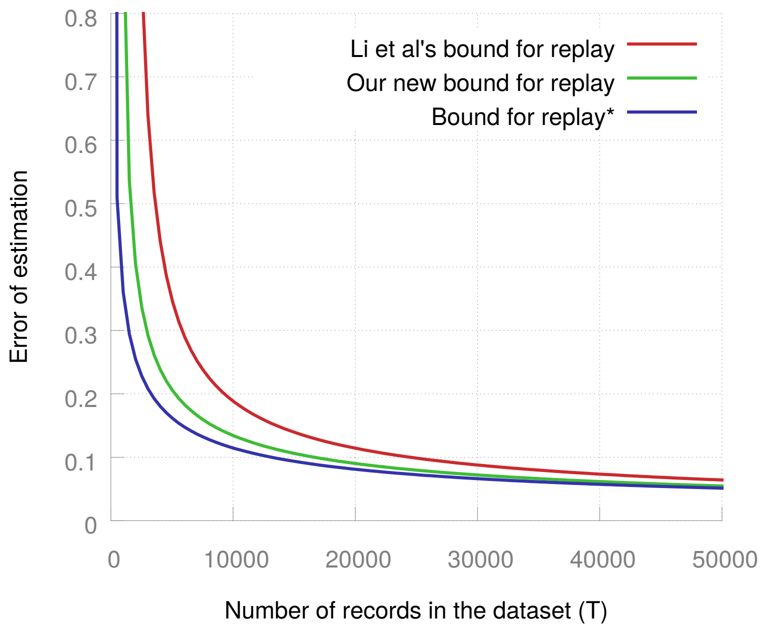


Figure 3.1: 0.95 probability bounds for the error made the replay and replay* methods (with $K = 100$ and $g_\pi = 0.1$). Lower is better. The two curves for the replay method corresponds to the one we derived and the state-of-the-art one. The formulas that are used here are the exact ones and *not* simply the highest order terms. Note that as it could have been predicted by looking at the structure of the bounds, the various parameters δ, K and g_π have little impact on the curves. Only the scale changes.

3.5.4 Comparison with the replay* method

Concentration analysis

As the estimator outputted by the unbiased method is a constant multiplied by the sum of Bernoulli random variables, it is much easier to analyze. Here is the result we obtain.

Theorem 6. *For any contextual bandit problem $\mathcal{D}$ with Bernoulli rewards, any* static *policy π and any dataset S of finite length T containing i.i.d. interactions (x_t, a_t, r_t) acquired using a uniform logging policy on $\mathcal{D}$ and for any $\delta\ in]0,1[$ we have that, with probability $1-\delta$, the error of the estimator outputted by the replay method*:*

$$\frac{K}{T}\sum_{t=1}^{T} V_t.r_t$$

is bounded by:

$$\sqrt{\frac{3Kg_\pi ln(2/\delta)}{T}}\ .$$

The proof is provided in appendix F.4. First note that the constants are slightly smaller than with the *replay method* and also that we are not talking about an order of magnitude or about a high order term as in theorem 5 but about an exact bound exactly as it can be proved. The two bounds are very close as it is shown on figure 3.1 - much closer actually than our bound for replay and the one by Li *et al.* [11] - but as opposed to what the intuition would suggest, the unbiased *replay* method* has a better concentration bound than the biased *replay method*.

Discussion

As we mentioned it in the previous paragraph, the concentration bound found for the *replay* method* is better than the one for the *replay method* which is counter intuitive. In fact this makes a lot of sense when we think about how the proofs were designed. However as we already improved the bound for the *replay method*, one may wonder if it would not be possible after all to shrink it further and then rightfully beat the unbiased method.

In fact it turns out that using the theoretical tools we used here, it is impossible. But why is that? First we do not say that it is impossible in general but only using classical concentration inequalities (The Hoeffding/Azuma inequality gives slightly looser results here). We can think about the following experiment. Let us imagine that we have complete access to $\mathcal{D}$ and use π to perform $\frac{T}{K}$ interactions with $\mathcal{D}$. This is an ideal experiment meant to quantify the cost of the uncertainty of V_t whose expectation is $\frac{T}{K}$. Because this experiment is only idealistic, we ignore the fact that $\frac{T}{K}$ might not be a integer. Let us then consider the estimator of the CTR of π resulting from this experiment. Using our notations, we would express this estimator as follows:

$$\hat{g}_\pi^{(world)}\left(\frac{T}{K}\right) = \sum_{t=1}^{T/K} \vec{r}[a_{\pi,x_t}] \ .$$

Obviously $\hat{g}_\pi^{(world)}\left(\frac{T}{K}\right)$ concentrates at least slightly faster than both replay methods as it is ridden of the randomness on the number of evaluations.

Yet, if we have a look at the bound we get using the Chernoff bound to analyze $\hat{g}_\pi^{(world)}\left(\frac{T}{K}\right)$, we see that:

$$P\left(\left|\sum_{t=1}^{T/K} \vec{r_t}[a_{\pi,x_t}] - \frac{T}{K}g_\pi\right| \geq \frac{T}{K}g_{pi}\gamma\right) \leq 2exp\left(-\frac{T\gamma^2 g_\pi}{3K}\right)$$

which is exact bound that we derive for the *replay method**.

This is the reason why we can say that, using these tools, it is not possible to show that the *replay method* is better in terms of accuracy than its unbiased counterpart.

Remark We really want to use this funny example to emphasize that, without at all trying to denigrate the approach, going after tighter concentration bounds as it is the case in numerous papers over the literature is not always the best way to improve an algorithm or an estimator's quality. The example we have at hand here is a perfect counter example in which intuition and/or empirical analysis are better advisers than concentration analysis.

The last paragraph of this section is about what happens to these concentration results with non stationary policies. In the next section, even though we do not completely solve the problem of counter intuitive results we just mentioned, we try a different analytic approach which allows to confirm that the *replay* method* is indeed not as good as the ideal "$\frac{T}{K}$ interactions with $\mathcal{D}$" experiment we defined. Moreover we visualize both graphically and analytically that the *replay method* concentrates faster than its unbiased counterpart.

3.5.5 Precision of evaluation of non stationary policies

We showed at the beginning of the section that it was impossible to prove convergence results in the general case, that is with non stationary policies (algorithms). This is the reason why we only considered static policies here. However Li *et al.* [11] exhibit good empirical performance with classical learning algorithms such as UCB [13], Thompson sampling [45] or LinUCB [7]. How is that possible? As a matter a fact, it is possible to derive similar convergence results if the policy is non-stationary but independent from any contextual information (see Langford *et al.* [10]). This setting is not very interesting as the whole point of recommendation is to be able to choose actions/items based on that context (the user and his/her session). In fact learning algorithms are not really the general case either and we could prove that the results presented here could be extended to some classes of learning algorithms at a cost of looser and more complicated bounds. Also these bounds would be specific to every single algorithm. What these algorithms have in common is that they are consistent and convergent. Indeed whatever rewards they get along a trajectory of interactions, such an algorithm will always end up having the same behavior (consistency), and so with higher probability as time passes (convergence). This is because as they always keep exploring a little bit, they will always end up finding the best action and doing it most of the time (consistency). Furthermore, the more they have explored, the more likely they are to have found the optimal policy and to stick to it (convergence). To summarize, if given a contextual bandit distribution $\mathcal{D}$ one can prove that an algorithm converges (almost) surely toward a behavior (optimal or not) at a given rate, then some convergence bounds can be proved for a replay evaluation of that algorithm.

Nevertheless we chose not to use this realistic setting for two reasons. First we wanted to be able to compare what we did with the state of the art. Second it would have made all the results and the proofs more complicated without deepening any further our understanding of the method. Indeed we would have had to add smoothness assumptions on the successive recommendations, convergence rates of the algorithms which would obviously depend on the natures of $\mathcal{D}$ on which additional assumptions would thus be necessary as well...

Remark Now one may think that all reasonable algorithms could be proved to concentrate around their real average CTR as T increases. Yet it is important to note that one may come up with a completely reasonable algorithm for which this would be wrong. An algorithm exploring first for some amount of time and then exploiting until the end of time would be an example. Such algorithms are very common in the literature when the horizon is assumed to be known [54, 96]. There is also an example of such an algorithm for contextual bandits which is intuitively called epoch greedy [23].

3.5.6 Expected squared error of a replay evaluation

We have seen that the analysis based on concentration inequalities did not allow us to make the difference we expected between the *replay* and *replay** methods. We have not been able either to quantify nor even detect the cost in terms of concentration of the *replay* method* compared to an ideal case in which $\mathcal{D}$ would be completely available. In this section we address these two issues by analyzing the variance of the estimators. Note that contrarily to the previous section, we never assume the rewards to be Bernoulli distributed since it is not necessary and that such an analysis was never undertaken in this context. The examples we take are Bernoulli distributed but all the results are valid for real valued rewards.

Theoretical analysis

Quantifying the cost of the uncertainty on the number of simulated interactions by replay* is quite easy. It can be addressed by analytically calculating the variance of both estimators. The calculations do not bring a lot to our understanding of the methods so they are in appendix. The ideal experiment "$\frac{T}{K}$ interactions with $\mathcal{D}$" obviously has a variance of:

$$\frac{K}{T} Var(\vec{r}[\pi(x)]) \ ,$$

where the variance in the result is with respect to the randomness engendered by drawing $(x, \vec{r}$ from $\mathcal{D}$ and possibly by π's stochastic choice of action given x. We omit to write it down in the equation to lighten the notations. The same thing goes for the expectations and variances in what follows.

Theorem 7. *Under the conditions stated in theorem 4, the replay* method has the following variance:*

$$\frac{K}{T}Var(\vec{r}[\pi(x)]) + \frac{K-1}{T}g_\pi^2 \ .$$

The variance of replay is also its expected squared error.*

This theorem clearly exhibits the cost in terms of variance of the uncertainty on the actual number of evaluated interactions (second term). The proof is given in appendix F.5 and simply comes from the development of the variance of the product of two independent random variables ($\vec{r}[\pi(x)]$ and V).

The *replay method* is biased so it would be unfair to use its variance to make a comparison as it would not take into account this bias. This is why we use its expected squared error (ESE). Indeed as mentioned in theorem 7, the ESE is the very definition of the variance for an unbiased estimator. However when the estimated value differ from the mean of the estimator (*i.e.* it is biased), the ESE and the variance naturally differ.

Theorem 8. *Under the conditions stated in theorem 5, the replay method has the following expected squared error:*

$$\left(\frac{K-1}{K}\right)^T \mathbb{E}\left[\vec{r}[\pi(x)]\right] + Var(\vec{r}[\pi(x)]) \sum_{i=1}^{T} \frac{1}{i}\binom{T}{i}\left(\frac{1}{K}\right)^i \left(\frac{K-1}{K}\right)^{(T-i)} \ .$$

The ESE of replay is not equal to its variance.

The proof is provided in appendix F.6 and uses the same approach of decomposition with respect to the values that V_t can take that was used to quantify replay's bias (theorem 3).

We have not been able to derive a close form formula for the variance of replay. We even suspect that this would be very difficult and possibly impossible to achieve. This result is thus quite difficult to compare with what we obtained for replay* and the ideal experiment "T/K interactions with $\mathcal{D}$". As such it does not allow at first sight us to analytically compare the *replay* and *replay* methods* nor to quantify the cost of the uncertainty on $\sum_{t=1}^{T} V_t$ on replay's ESE.

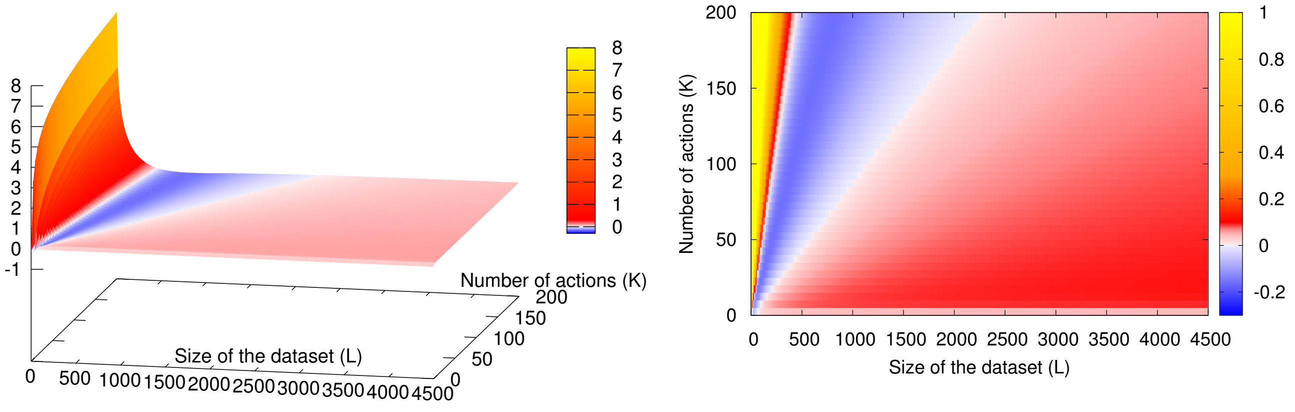
However it is possible to prove an interesting result.

Theorem 9. *Under the conditions stated in theorem 5 and asymptotically in T (only for the second equality), the ESE of replay* and replay can be expressed as follows:*

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay*)} - g_\pi\right)^2\right] &= \frac{KVar\left(\vec{r}[\pi(x)]\right)}{T} + O\left(\frac{1}{T}\right) \ , \\
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] &= \frac{KVar\left(\vec{r}[\pi(x)]\right)}{T} + O\left(\frac{1}{T^2}\right) \ .
\end{aligned}$$

The first equality is a simple rewriting of theorem 7. The second one is trickier to prove and comes from the fact that asymptotically, the binomial distribution is symmetric and has an exponentially small tail (it is the Normal distribution). A proof is provided in appendix F.7. The proof is asymptotic in T but we are convinced that the result is true way before that. In general, the normal approximation of the binomial distribution is considered accurate when $\frac{T}{K} > 5$ which is almost always true when replay is applied in the real world. We even suspect the equality to be true for all T.

A interesting way to interpret these equations is by considering the second term as the cost in term of variance of the uncertainty over $\sum V_t$. Indeed the errors are expressed in terms of the variance of the ideal experiment "$\frac{T}{K}$ interactions with $\mathcal{D}$" that does not suffer from that cost, plus the cost corresponding to the chosen method. With this analysis, we finally exhibit what



(a) 3d plot (CTR=0.1). (b) Heat map (CTR=0.1).

Figure 3.2: Log ratio of the ESE of replay* over the ESE of replay under two different views. Blue means that replay* has less variance whereas red means that replay has less variance. The CTR of the policy considered here is 0.1 which is quite typical for a good policy in a news or video recommendation application. The blue hollow and the red area as T grows larger are roughly what we expected. Note that the color scales are different in the 3d plot and the heat map for a better visualization. Note also the linear separations between red and blue areas.

we expected from the beginning but could not verify analytically with concentration bounds: the cost induced by the uncertainty on the number of evaluations is much lower for the classical *replay method* than for replay*. As a consequence, replay is more accurate than replay*.

Nevertheless there is still a little downside. Our analysis is asymptotic and we provided intuition justifying that it should be true at least when $T > 5K$. It remains unclear however which method would be better for very small values of T. This is what we study next.

Visual analysis

It is also interesting to wonder what happens when T is relatively small. Indeed it is possible to consider a time-dependent dataset to be static by part, each part being very small. One could also think of problems in which the horizon is typically small or could only be interested in this for the only sake of science. Moreover in this section we also try to quantify the difference between the two methods depending on the parameters of the problem.

In any case, the previous analysis does not give us information about that. Note however that this analysis is far from asymptotic either for it only requires the normal approximation of a binomial distribution to be accurate enough. Our intuition tells us that the *replay method* would suffer from a larger variance when T is small since a lot of very small realization of $\sum V_t$ (and even 0) would then tend to occur. Indeed when V is small the result of *replay* is very unstable whereas the systematic multiplication by $\frac{T}{K}$ induced by *replay** leads to more reasonable variations.

Figures 3.2, 3.3 and 3.4) show the results of our investigation. It is important to have in mind that these plots are not the results of experiments but simple plots using the previously calculated formulas of the ESE that are quite complicated to fully compare analytically. Note for the record that the formula for the ESE of *replay* is not in close form and thus quite heavy to compute as T increases. Indeed the results corresponding to one heat map for a given value of the CTR took 2 days to compute on a $2.5GHz$ processor with code written in python.

All the figures are commented one by one but a few features are common to all the results. As expected *replay** is better for small values of T. This is because dividing by the expected value of $\sum V_t$ ($\frac{T}{K}$) instead of the actual value of $\sum V_t$ as in *replay* tends to compress the results towards zero when $\sum V_t$ is lower than its mean. Indeed in that case $\sum V_t$ is close to zero which

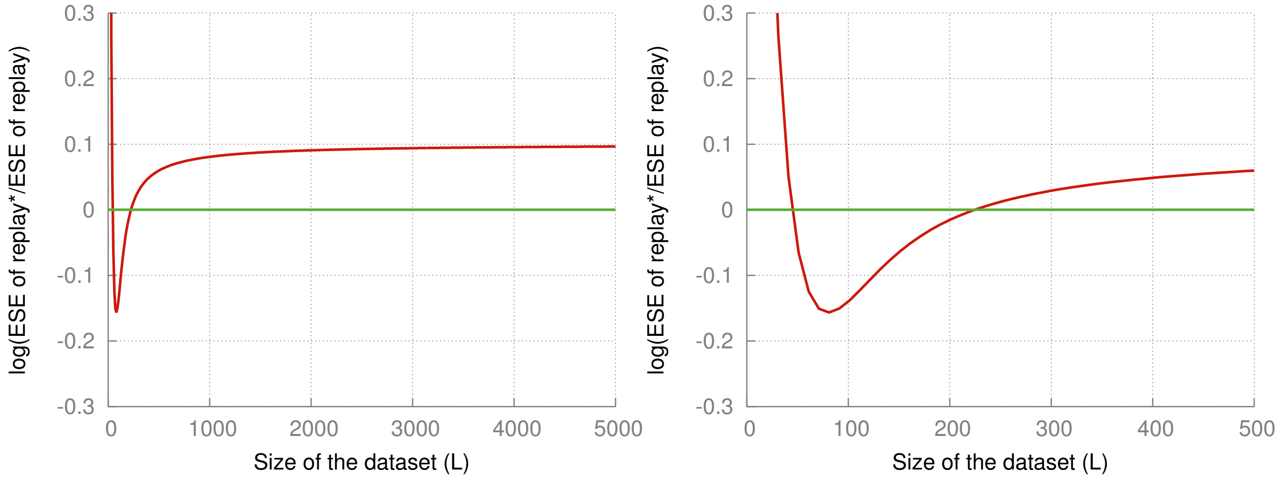


Figure 3.3: Log ratio of the ESE of replay* over the ESE of replay for K=20 and CTR=0.1. This plot uses the same data as the 3d plot from figure 3.2 but is meant to give a better view of the difference of variance depending on T for a typical value of K. The right-hand figure is simply a zoom on the down peak between 0 and 500 on the left-hand figure. A positive value y in ordinate on that plot means that *replay** has e^y more variance than *replay*. A negative value of y means that *replay* has e^{-y} more variance than *replay**. In particular at the down peak ($T \approx 80$) *replay* has 17.3% more variance than *replay** whereas as T grows larger, *replay** stabilizes at 10.5% more variance than *replay*. These relative difference are non negligible.

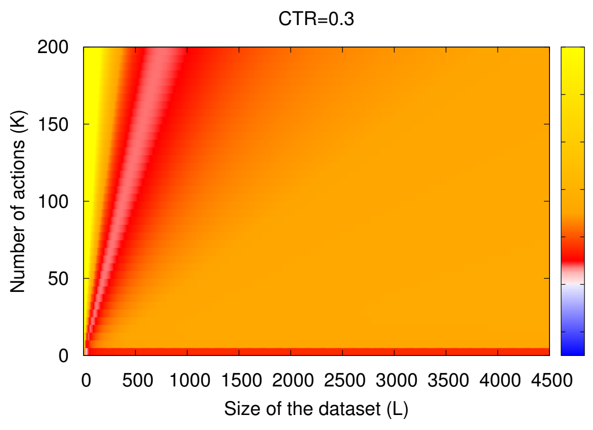
is where the variance is the highest. The aforementioned compression reduces that variance. Dividing by the actual value of $\sum V_t$ is more accurate in a sense but engenders a lot of variance when that value is close to zero.

One common feature however is more unexpected: *replay* is much better than *replay** when T is very, very small. This corresponds to the huge peak of yellow area at the very left of each plot. One may argue that this is not very important as it is quite hopeless to try to estimate a policy when T is equal to only once or twice the size of the action space (K). This is completely true, this case is extreme and is not what the methods were designed for but it is always intellectually interesting to explain the unexpected. The best way to understand this is by way of a small example.

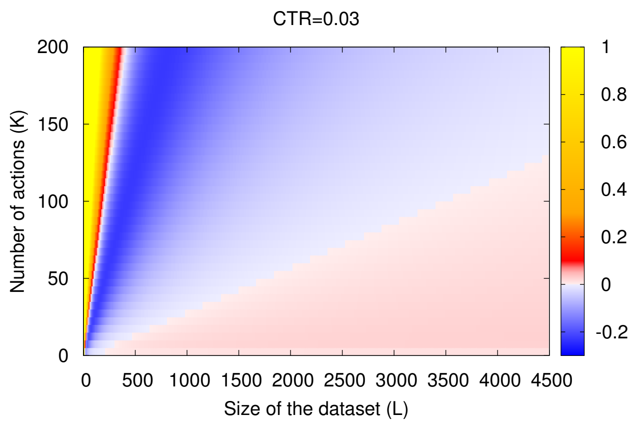
Let us assume that $T = K$ and thus that $\frac{T}{K} = 1$. Then let us consider the possible values of $\sum V_t$.

- If $\sum V_t = 0$, $\hat{g}_\pi^{(replay)} = \hat{g}_\pi^{(replay*)} = 0$.
- If $\sum V_t = 1$, $\hat{g}_\pi^{(replay)} = \hat{g}_\pi^{(replay*)}$ because $\sum V_t = \frac{T}{K} = 1$.
- The "problem" or the reason for the surprise arises when $\sum V_t > 1$. For a typical CTR of 0.1 for the evaluated policy, $\sum V_t.\vec{r}[\pi(x_t)]$ will be equal to zero most of the time. Yet all the rewards of one will be problematic and lead to a lot of variance for both methods. Indeed $\sum V_t$ and $\frac{T}{K}$ (that respectively divide the previous quantity for replay and replay*) will never be big enough to make the estimator reach a value near the true CTR. As in that case $\sum V_t > \frac{T}{K}$, *replay* has a much smaller ESE than *replay**.

This unexpected behavior occurs up to when $T \approx 2K$ but as it is shown on the figures, it quickly disappears when $\frac{T}{K} > 2$ since it becomes big enough to compress the results outputted by *replay** towards zero as what we had intuitively foreseen. We then witness, as expected, replay* outperforming replay for small datasets. Finally for reasonable datasets, replay outperforms replay* by a significant margin. Indeed contrarily to what the similar order of magnitude of



(a) Heat map (CTR=0.3).



(b) Heat map (CTR=0.03).

Figure 3.4: Log ratio of the ESE of *replay** over the ESE of *replay* for CTRs of 0.3 and 0.03. Note that the color scale is the same as for the previous heat map (3.2) to facilitate comparisons. We clearly see that for a bigger CTR (and thus a bigger variance), *replay* becomes much better than *replay** and the hollow area does not even go bellow zero. On the contrary, if the CTR and thus the variance decreases, the hollow blue area spreads and even when T grows larger the difference between the two methods becomes much less significant.

theoretical variance might have let us think, replay is significantly less variate than replay* even for very large datasets.

This concludes our study of the theoretical properties of replay and replay*. A recap of the results that were proved in this work are available in table 3.1. Before moving on to the empirical study, there are other aspects of replay that need to be addressed. In what follows we will talk about what happens when some of the assumptions considered in this work are not met in practice. In particular we will comment further on the assumption of a uniformly random logging policy that may seem extreme and see how it can be relaxed. We will also provide insights to use the full potential of the replay method when the contextual bandit problem we consider is not stationary. Indeed, stationarity can definitely not be assumed in the main problem we consider in this thesis work: dynamic recommendation.

3.6 Relaxing assumptions on the logging policy: what can and cannot be done

Li *et al.* [11] and Langford *et al.* [10], the authors of the replay method, claim all along their articles that the assumption of a random uniform policy can be relaxed. A few theoretical works going in that direction have even been published. We mention them further in this section. Here we mainly address two very important points. The extent to which the random uniform assumption can be relaxed is provided in section 3.3. Indeed as long as the conditions we mentioned are respected, replay should keep working in the theoretical framework introduced by Li *et al.* [11] and that we consider here. Yet, and this is our first point, what are the implications? How replay should be tweaked to work as described in the literature? We will briefly mention the state-of-the-art solutions to this issue. A question is not answered in the literature. What are the consequences on the theoretical results presented in this chapter? Without going into the details, we provide intuition on the subject. Finally we also briefly review a few works considering the case in which the logging policy is unknown. All this remains in the theoretical framework introduced by Li *et al.*, that is when evaluating stationary policies. The only things that are new are a few additional relaxation conditions, considerations with

Table 3.1: Recap of the results proved in this chapter on the *replay* and *replay** methods and comparison with what was the state of the art before this work (last column). As a reminder, "$\frac{T}{K}$ interactions with $\mathcal{D}$" is an ideal experience in which $\frac{T}{K}$ interactions are made with the real world. It is only meant as a baseline for comparison so as to quantify the cost of the uncertainty on the number of evaluations by replay and replay* whose expectation is T/K. Consequently we ignore the fact that T/K might not be an integer.

	Real world evaluation (T/K interactions with $\mathcal{D}$). Baseline for comparison.	*Replay* method*	*Replay method*	*Replay method* (State of the art)
Estimator	$\frac{K}{T}\sum_{t=1}^{T/K}\vec{r}[\pi(x_t)]$	$\frac{K}{T}\sum_{t=1}^{T}V_t\vec{r}[\pi(x_t)]$	$\frac{\sum_{t=1}^{T}V_t\vec{r}[\pi(x_t)]}{\sum_{t=1}^{T}V_t}$	Same estimator
Bias	0	0	$\left(\frac{K-1}{K}\right)^T.\mathbb{E}\Big[\vec{r}[\pi(x)]\Big]$	Not analyzed and not clearly mentioned
Concentration bound around g_π with probability $(1-\delta)$	$\sqrt{\frac{3Kg_\pi ln(2/\delta)}{T}}$	$\sqrt{\frac{3Kg_\pi ln(2/\delta)}{T}}$	$O\left(\sqrt{\frac{3g_\pi Kln(4/\delta)}{T}}\right)$	$O\left(\sqrt{\frac{(g_\pi^2+g_\pi)3Kln(4/\delta)}{T}}\right)$
Variance	$\frac{K}{T}Var(\vec{r}[\pi(x)])$	$\frac{K}{T}Var\left(\vec{r}[\pi(x)]\right)$ $+\frac{K-1}{T}\mathbb{E}\left[\vec{r}[\pi(x)]\right]^2$	$\left(\frac{K-1}{K}\right)^T\mathbb{E}\left[\vec{r}[\pi(x)]\right]+$ $Var(\vec{r}[\pi(x)]).$ $\sum_{i=1}^{T}\frac{1}{i}\binom{T}{i}\frac{(K-1)^{(T-i)}}{K^T}$	Not analyzed
Cost of the uncertainty on $\sum V_t$ in terms of variance	0 (baseline, $\sum V_t=\frac{T}{K}$)	$O\left(\frac{1}{T}\right)$	$O\left(\frac{1}{T^2}\right)$	Not analyzed

respect to the results we introduced in section 3.5, such as replay* or the bias/unbiasedness. Our second point however is about learning algorithms for which no result exist besides theorem 2 and the impossibility to derive convergence bounds. The theoretical framework we consider only deals with the evaluation of static policies. Nonetheless, when the logging policy is uniform, given the asymptotic unbiasedness for all algorithms (theorem 2) as well as the convergence properties of classical algorithms, we justified in section 3.5.5 that things work out as expected for learning algorithms as well. Is it the same for non-uniform logging policies? Can theorem 2 be verified in that case? This very important problem, that was not studied in the literature, is discussed here. This discussion is backed by new theoretical results.

In the remainder of this thesis work, we will keep the random uniform assumption. By the end of this section, the reader may have a clearer pictures of the reasons that motive this choice. We can already mention a few of them. Firstly datasets that meet this assumption are starting to pop out. We can for instance mention the Yahoo! R6B dataset [12] that was made public and on which we make most of our experiments in this thesis work. The dataset provided by Adobe for the Exploration *vs.* Exploitation challenge 2 (chapter 2) is another example. It was however not made public. Another reason is the fact that the uniform assumption yields clear and simple results without altering the intuition on what would happen in more complex cases. Finally this setting is considered by the main paper about replay (Li *et al.* [11]) and using it allows to easily compare the results we derive. Other reasons are discussed in this section. As a consequence, non uniform logging policies will almost only be discussed here.

3.6.1 Non-uniform logging policy

Let us first consider the setting in which the main results of this chapter were proved, that is when a static policy is evaluated. The adapted or corrected replay method presented here is proposed by Langford *et al.* [10]. We simply add more details in the light of the analysis that we conducted in this chapter.

When the logging policy π_{LOG} is not uniform, to make the policy evaluation work as expected, it is important to give a weight to each recommendation. The idea is to artificially balance the evaluation by giving more importance to actions that are not chosen very often by π_{LOG}. The CTR is thus computed as follows by the evaluation method:

$$\hat{g}_{\pi}^{(replay)}(S) = \frac{\sum_{t=1}^{T} V_t.\vec{r}[a_t].w_t}{\sum_{t=1}^{T} V_t.w_t}.$$

The w_t are the weighting coefficients and are simply the inverse of the probability that π_{LOG} chooses action a_t given x_t:

$$w_t = \frac{1}{\mathbb{P}(\pi_{LOG}(x_t) = a_t)} .$$

To obtain a truly unbiased estimator, we need to divide by the expectation of $\sum_{t=1}^{T} V_t.w_t$ for it can be equal to zero. This expectation is T as the very definition of w_t yields $\mathbb{E}\, V_t = w_t^{-}1$. Thus the replay* estimator is defined as follows:

$$\hat{g}_{\pi}^{(replay^*)}(S) = \frac{\sum_{t=1}^{T} V_t.\vec{r}[a_t].w_t}{T} .$$

Note that for this to work, for all context x, the probability of choosing any action a must be strictly positive. Note also that this method, usually referred to as *inverse propensity score* (IPS) [97] perfectly balances the estimator. Therefore, as long as π_{LOG} explores all the actions, the estimator has the same expectation as the one given by the replay method. It also remains asymptotically unbiased.

In addition a few things are worth discussing. First if the w_t are not well distributed, this will generally cause a low data efficiency, that is to say a high variance. When considering

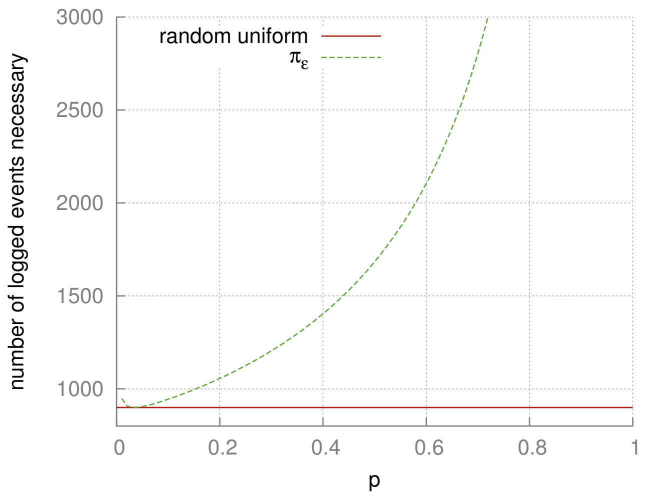


Figure 3.5: Number of logged events required to let an evaluated policy play each action once ($K = 30$). For π_ε the results depend on p, the probability to chose the best action. Note that when $p = \frac{1}{K}$, π_ε is the uniformly random logging policy. If p is equal to 0.7, the evaluation already requires three times more data. If p gets greater than that, the necessary amount of data grows very fast.

the evaluation of deterministic policies/algorithms, the reason is rather straightforward. To see that let us consider an ε-greedy logging policy π_ε that always chooses the same action a with probability p, and another one at random with probability $1-p$. Such a policy is typical. It may be the best policy a company has at its disposal, randomized to allow future evaluation. The first simple symptomatic example is a policy π that always chooses the same action (different from a). It will not use one Kth of the data but a fraction $\frac{1-p}{K-1}$ of it. More importantly this is problematic for all the index-based bandit algorithms that are the most widely used such as UCB, LinUCB, KL-UCB which are deterministic. Let us indeed consider UCB that needs to play each action once at the beginning for instance. In expectation, this takes K^2 logged events if the logging policy is random uniform. If the logging policy is π_ε, it requires much more logged events:

$$\frac{1}{p} + \sum_{i=1}^{K-1} \frac{K-1}{1-p} = \frac{1}{p} + \frac{(K-1)^2}{1-p} \; .$$

This phenomenon is caused by the fact that since UCB is deterministic, each time it wants to perform an action which is rare in the data, it has to wait for a long time. In the meantime it is completely unable to use the data that goes by. See figure 3.5 to get an idea of the consequences of setting p to too high a value.

It is not as obvious that the evaluation stochastic policies would also exhibit a low data efficiency. Indeed such policies will not be blocked waiting for a rare event in the data like the deterministic policies are. The difference is thus definitely going to be less important than with deterministic policies in terms of number of records used. Yet it is also rather intuitive that an non-uniform logging policy yields a higher variance in that case as well. Indeed some events will be rare and get a high weight. Therefore when they happen they have a big impact on the estimate, increasing the variance. The case that ensures the lowest variance is when all the w_t are uniformly distributed (random uniform policy). Indeed when all the weights are equal, the respective variance of the corresponding events balance each other out. Note that the uniform logging policy is optimal if we do not know in advance the policy we seek to evaluate. We will see in chapter 5, section 5.8.2 that otherwise, better can be achieved (theorem 17). Theorem 17 also quantifies the variance of replay when the logging policy is not uniform.

A non-uniform logging such as π_ε could be a good idea if the evaluated policy is close to it.

This may be the case in a process of iterative improvements. In this context, one may be able to achieve a better data-efficiency than $\frac{1}{K}$. Bottou *et al.* [98] successfully used such an approach to iteratively improve ad display in Bing, Microsoft's search engine. Although the approach is basically what we just described, they present it within a different framework named *causal inference*. We provide tools to improve this approach in chapter 5.

Finally let us mention that we made the implicit assumption here that the logging policy needs to be stochastic. Note however that in that case stationarity is not compulsory as long as we know the probability of each action at each time step. Even contextual independence is not compulsory if the probabilities of each actions are strictly positive and known. Stochasticity is not a strict requirement either. Since all the couples $(x, \vec{r})$ that are drawn from $\mathcal{D}$ are *i.i.d.*, a deterministic policy that performs all the actions in turns would work exactly as a random uniform policy. More generally, any deterministic policy that performs all the actions in at least a small proportion within each considered time frame can be used. The inverse of these proportions are then used as weights inside the corresponding time frame. Note that in the deterministic case, contextual independence of the logging policy is crucial unless the size of the context space does not prevent the following condition to be possible: each action has to be performed in at least a small proportion in each context within each considered time frame. In that case the weights are context and action dependent. However this condition is rarely possible given the tremendous amount of information available in a context vector.

3.6.2 Unknown logging policies

Most of the work that tries to extend the replay method by Li *et al.* [11] and Langford *et al.* [10] simply considers unknown logging policies to be able to use a broader range of datasets. Nonetheless the problem is interesting and is worth being mentioned. This is why, for the sake of thoroughness, this section reviews these works. Since the literature circumscribes the problem pretty well, nothing new is added here.

The assumption that the logging policy is known is necessary to build the replay estimator. Yet it is possible to deal with an unknown logging policy as long as it respects the other assumptions. This can be done in two ways:

- The first method consists in using the data to estimate the rewards given an action and a context. This is not a replay method but a simulator based on a model learned using the data. Such a method is obviously biased by nature and favors some types of algorithms whose model is similar to the one used by the simulator. Beygelzimer and Langford [99] account of some of the difficulties that can arise when trying such an approach.

- The other method consists in estimating the logging policy using the data. Then we simply consider it known and weight the updates as presented in the previous section. The first work on the subject was proposed in the same article that first presented the replay method. It is based on a simple frequency analysis [10]. Note that such a straightforward approach is only possible when we assume that the logging policy does not depend on contextual information. When it does, since the context space is generally huge, one has to use more complex tools in order to estimate the logging policy (*e.g.* clustering or regression [97]).

More interestingly, a *double* method was proposed by Dudík *et al.* [100]. The idea is to build an estimator that uses both an estimation of the reward distributions given the context and an estimation of the logging policy. Indeed evaluations based on reward models suffer from a high bias whereas evaluations based on logging policy models suffer from high variance. By combining the two estimators, the authors of the doubly robust method introduce a trade off between their variance and bias. They prove that if at least one of the two estimates is good, it yields accurate policy evaluations. They also exhibit interesting empirical results. Nevertheless the problem remains: both estimation tasks are quite difficult. A last work by Strehl *et al.* [97] tackles this issue. They see the problem as no longer a policy evaluation task but as a policy comparison task. Their method is basically an inverse propensity score (IPS) with a lower threshold τ on $\mathbb{P}(\pi_{LOG}(x) = a)$. Therefore the new weights introduced are as follows:

$$w_t = \frac{1}{max\left(\mathbb{P}\left(\pi_{LOG}\left(x_t\right) = a_t\right), \tau\right)}.$$

Empirically $0.01 \leq \tau \leq 0.05$ seems to work fine [97]. This technique obviously introduces a bias in the estimator but severely reduces its variance. In machine learning this is the well known as a bias *vs.* variance dilemma. The smaller τ is, the less the method is biased but the more it is variate. When τ grows, we add bias but the variance drops since it limits the high variance introduced by IPS when some estimated probabilities for $\mathbb{P}\left(\pi_{LOG}(x) = a\right)$ are too small. This variance drop makes the comparison of policies easier (that is to say possible with much less data). In return policies tend to be underestimated. Note however that in an iterative process, the bias introduced may just hide improvements as they will most likely be due to changes in regions where the threshold will be activated. Strehl *et al.* [97] also prove a very nice high probability bound on the difference between the policy maximizing this new estimator and the best policy in the real world. The reader is referred to the full paper for more details [97].

This section and the former (3.6.2 and 3.6.1) are meant to exhibit that some of the assumptions made in this chapter could be relaxed but not without price. Indeed with non-uniform policies, variance is introduced. For stationary policies, this cost is quantified in the works we mentioned here and does not change the nature of the method (we only add weights). Similarly, if the policy is unknown, we obviously pay the cost of its estimation. It is hard to comment on that cost without real life examples. Since this is clearly not the focus of this thesis work, we will not dig into this any further. Another reason is that this problem seems less crucial than others. Indeed if a data scientist trying to improve a company's system does not know about the policy who logged the data he or she is working on, the problem seems more political than scientific.

3.6.3 Evaluating a learning algorithm when the logging policy is not random uniform

The techniques we described to handle non uniform logging policies are basically what was done by Langford *et al.* [10]. We simply added a few possibilities of relaxation (*e.g.* some context-dependent logging policies, the case of replay* *etc.*) and details on the consequences. We then reviewed what could be done with an unknown logging policy. It is important to emphasize that all the work we talked about only applies to stationary policies. In the literature the argument that it is possible to use all sorts of logging policies by means of minor changes is used rather lightly. We may wonder however if this is true for non stationary algorithms and in particular for learning algorithms. Indeed the convergence results of stationary policies evaluated on uniform logging policies can be extended to the evaluation of learning algorithms. Nonetheless, this does not mean *a priori* that the results on non uniform logging policies can be extended as well. Since this was never discussed before and because the results on stationary policies may mislead people into thinking that they are somewhat valid in general, we derive new results to clarify all this. The purpose of the proof of the first result we present (theorem 10) is twofold. First it obviously proves the theorem. Second and mainly, this proof aims at illustrating, by way of a few extremely simple examples, the intuition that the evaluation of non-stationary algorithms without a random uniform logging policy is much trickier than the evaluation of the static policies considered in our theoretical framework (sections 3.6.1 and 3.6.2).

Theorem 10. *There exist a contextual bandit distribution $\mathcal{D}$ (even with only two actions and a unique context) and a stationary, random but non-uniform logging policy π_{LOG} such that for all datasets S of **infinite** length containing interactions acquired using π_{LOG} on $\mathcal{D}$, it is not possible to design a set of weights such that the replay method or replay* is unbiased for the evaluation of all algorithms.*

Proof. The proof is simply by construction of an counterexample. We also take advantage of this example to exhibit a few other things along the way.

Let us consider a non-contextual bandit problem with two actions 1 and 2 that yield deterministic rewards $r_1 = 1$ and $r_2 = 2$. Let us also consider a logging policy pi_{LOG} that performs 1 with probability 0.1 and 2 with probability 0.9.

Since S is infinite, the estimators yielded by replay and replay* can be written as functions of the weights w_1 and w_2, the weights correcting actions 1 and 2 respectively.

Let us consider some stochastic policy π. For simplicity we note π_i the probability that π chooses action i. It is straightforward that:

$$\hat{g}_{\pi}^{(replay)} = \frac{w_1\pi_1 0.1 + 2w_2\pi_2 0.9}{w_1\pi_1 0.1 + w_2\pi_2 0.9}\ , \qquad \hat{g}_{\pi}^{(replay^*)} = w_1\pi_1 0.1 + 2w_2\pi_2 0.9\ .$$

Indeed for both estimators, the numerator is the reward which is added on average to $\hat{G}_\pi$ with one record (without ignoring that with a rejection we add 0). More formally it is the expectation of $V_t r_t$. Since all the records are *i.i.d*, as T goes to infinity, replay* is equal to T times this value (T records) divided by T. Similarly, replay is the ratio between the sum of corrected rewards by the sum of the weights. As T goes to infinity, it is equal to T times the average reward divided by T times the average weight.

As justified in the previous section, with $w_i = {\pi_{LOG,i}}^{-1}$ replay* is unbiased for any dataset and replay is unbiased for any infinite dataset.

Now let us consider an algorithm A_a that chooses alternatively action 1 and action 2. When evaluated on a dataset generated by π_{log}, its behavior is not going to be altered: A_a will choose each action alternatively. One important detail is that it will take $0.1^{-1} = 10$ records on average to perform action 1 and $0.9^{-1} = 10/9$ records on average to perform action 2. Therefore to perform the two actions consecutively, it takes 100/9 records on average. Consequently, in that case the estimators can be written as follows:

$$\hat{g}_{A_a}^{(replay)} = \frac{w_1 + 2w_2}{w_1 + w_2}\ , \qquad \hat{g}_{A_a}^{(replay^*)} = \frac{w_1 + 2w_2}{100/9}\ .$$

Indeed the two actions are performed alternatively an infinity of times. Thus replay is the sum of the rewards yielded by the two actions divided by their cumulated weights. Replay* is that same sum of rewards divided by the average time it takes for the two actions to be performed.

It is clear that since the non-uniform logging policy does not corrupt at all A_a's behavior when it is replayed, the corrections (weights) we introduced for the stochastic policies would bias the estimators. Intuitively, since the behavior is the same as in the real world, no corrections should be needed ($w_1 = w_2 = 1$). This is true for replay. Yet it does not work for replay* and yield an average reward of 0.27 instead of 1.5. This is simply because as the algorithm is not stochastic, to many records are ignored. As a consequence, the denominator that simply takes into account the number of records is too high.

What matters here is that the weights that are needed to make replay (or replay*) asymptotically unbiased are different for A and for some stochastic policy π. Yet the intuitive weights are not the only ones. For A_a we actually only need $w_1 = w_2$. Let us consider the evaluation π_u the random uniform policy that also have an average reward of 1.5. For replay to be unbiased, we need:

$$\hat{g}_{\pi_u}^{(replay)} = \frac{w_1\pi_1 0.1 + 2w_2\pi_2 0.9}{w_1\pi_1 0.1 + w_2\pi_2 0.9} = 1.5 \qquad \Leftrightarrow \qquad w_1 = \frac{9w_2\pi_2}{\pi_1}\ .$$

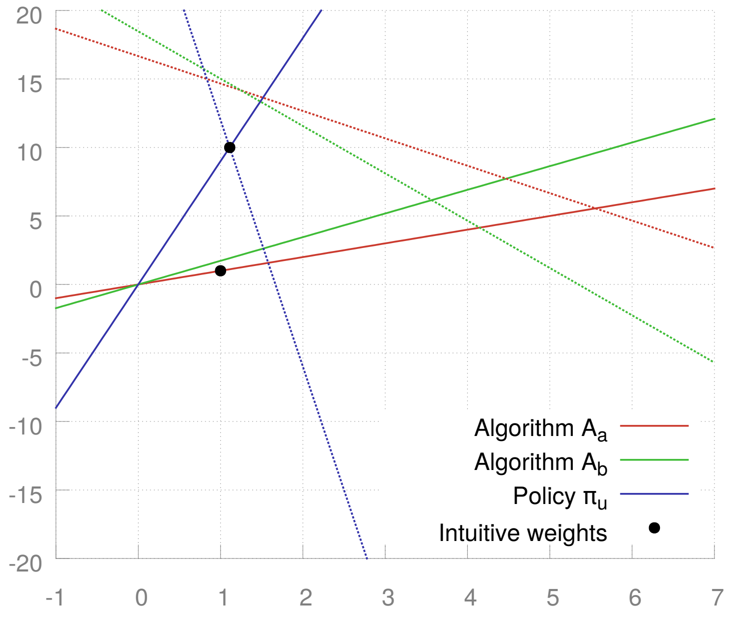


Figure 3.6: Weights that yield unbiased estimates from replay (solid lines) and replay* (dotted lines) in the example used to prove theorem 10. Notice that the intuitive value of the weights for stochastic policies - $(10/9, 10)$ - works for both estimators. On the contrary, not making corrections - *i.e.* with weights equal to $(1,1)$ - when evaluating the purely deterministic A_a only works for replay. In any case, for any method, there exists no single set of weights that allows an unbiased evaluation of all algorithms.

Similarly for replay* we need:

$$w_1 = \frac{1.5 - 0.9w_2}{0.05} \ .$$

If we fix π_1 and π_2 we have two linear equations for each estimator with two unknown variables that yield a solution for the set of weights. For replay since w_1 and w_2 are simply proportional, this solution is $(0,0)$. Yet the sum of the weights for replay cannot be equal to zero for we need to divide by it. This finishes the proof for replay: there is not a set of weight that makes it unbiased for both π_u and A_a. For replay* this solution is valid. Yet it is not going to be either one of the intuitive solutions we mentioned for both algorithms. This is a good indicator that there is a problem.

As a matter of fact, this is when we consider a third algorithm that things go astray for replay*. Let us consider an algorithm A_b that performs 1, then 2 and then an action at random before iterating over and over. This algorithm also has an average reward of 1.5. Similarly to what was done for A_a, we can derive simple formulas for replay and replay*:

$$\begin{aligned} \hat{g}_{A_b}^{(replay)} &= \frac{w_1 + 2w_2 + 0.1w_1 + 0.9 * 2w_2}{w_1 + w_2 + 0.1w_1 + 0.9w_2} &= \frac{1.1w_1 + 3.8w_2}{1.1w_1 + 1.9w_2} \ , \\ \hat{g}_{A_b}^{(replay^*)} &= \frac{1.1w_1 + 3.8w_2}{10 + 10/9 + (0.5 * 0.1^2 + 0.5 * 0.9^2)^{-1}} &= \frac{1.1w_1 + 3.8w_2}{100/9 + 0.41^{-1}} \ . \end{aligned}$$

For replay we find yet another linear relation. For replay*, things go astray as well: the equation between w_1 and w_2 is not collinear with the other two. Consequently there is no set of weights that allows an unbiased evaluation by replay* of the three algorithms and thus of all the algorithms in general. Yet, as expected, all stochastic policies are evaluated without bias by replay as long as $w_1 = 9w_2$. Figure 3.6 shows visually the weights that allows an unbiased evaluation of the individual policies and the fact that there is no common intersection.

□

This theorem is very interesting. In short it says that the weights introduced in the previous section only work for stationary policies (stochastic or not). The rest of the time they do not. However the examples inside the proof seem to exhibit a silver lining: if the algorithm is deterministic, not making corrections allows replay to work as expected. It would be nice as many bandit algorithms (UCB, KL-UCB, LinUCB *etc.*) are deterministic. Unfortunately this is false. It only works for non-contextual, deterministic algorithms which are very restrictive. They, for instance, forbid personalization which is preposterous in the context of recommendation. Indeed, if the algorithm is not contextual but even remotely stochastic (*e.g.* Thompson sampling), similarly to what happened with A_b it does not work. Indeed since the exploration is randomized, the imbalance of the logging policy will make the algorithm explore faster some actions than others which would bias the learning process. Even with corrections on the rewards that are provided via the *update* function, there is no saying that the algorithm would handle weighted records the same way as several non-weighted ones. As an example, some estimators within the algorithm will be more variate as based on less data. The same thing goes for deterministic, contextual algorithms. Some contexts will be explored faster which would bias the process. For the same reasons as before corrections are not possible in general. This yields the following theorem that we do not prove formally. The existence of A_b in particular and stochastic and/or contextual algorithms in general are intuitive proofs.

Theorem 11. *There exists a contextual bandit distribution $\mathcal{D}$ such that for all datasets S of* ***infinite*** *length containing interactions acquired using a stationary, random but non-uniform logging policy on $\mathcal{D}$, all integers T we have that some histories of interactions h_T have different probability whether A is played against $\mathcal{D}$ or evaluated using the replay method on S.*

Note that we do not talk about unbiasedness here. Indeed, for any algorithm (at least the consistent ones), since the dataset is infinite, it is very likely that some corrections using weights could yield an unbiased estimator of g_A when evaluating using either replay or replay*. Yet it would simply be a coarse trick to hide the bias without getting rid of it. Indeed it is clear that the probability distribution of the history would be different from the real world regardless of the weights which is actually what matters. Indeed we care about way the algorithm learns.

As a conclusion, one interested in evaluating stationary policies such as in iterative improvements [98] can significantly relax the assumptions on the logging policy as explained in the literature and summarized here. Nevertheless, a fact that had not been discussed before is that one interested in the evaluation of learning algorithms has no real choice but to acquire random uniform data. This may seem extreme but considered that what is measured is not what is learned but how it is learned, one dataset is enough to evaluate any approach at any time. It is an important cost to pay but it allows to "play forever".

3.7 Practical considerations

First of all we recall that the experimental study of the replay method is left for chapter 4. Therefore this is not what this section is about. Here we simply provide useful figures and techniques to be able to successfully use the method in practice.

3.7.1 On the necessary amount of data

So far we have mainly analyzed the bias of the *replay method* and compared its convergence with *replay**, its unbiased counterpart. Yet we said very little on its convergence in absolute terms. In other word what amount of data do we need, depending the number of actions K? In the literature it is claimed that this method is only helpful when K is relatively small, that is less than a hundred [11]. It is obvious that the rejection process, which only allows us to use one Kth of the data limits the number of actions when using this method. Moreover this method is meant to evaluate learning algorithms that need to have time to interact with the environment before reaching their final regime, hence the need for relatively large datasets and small pools of actions.

Yet one may wonder if regular recommendation could be evaluated using this method. It would be a way to evaluate something that directly reflects user interactions with the system as opposed to prediction accuracy. The question is worth being raised because with classical recommender systems, one is only interested in evaluating a static policy. This completely satisfies the theoretical assumptions made in this chapter and the learning process is done offline. Therefore we could expect a concentration rate in $O\left(\sqrt{\frac{K}{T}}\right)$. Note however that in general, for prediction accuracy estimators the rate is much faster ($O\left(\sqrt{\frac{1}{T}}\right)$). Then, considering that classical recommender systems can be evaluated, how much data do we need to do so?

To answer this question, let us consider the *replay* method* for the sake of simplicity. It should not matter much anyway since we have established that it has a similar rate of convergence as the *replay method*. The variance of the estimator corresponding to the application of this method on a dataset of size T, with K actions (or items in the context of classical recommendation) is:

$$\frac{K}{T} Var\left(\vec{r}[\pi(x)]\right) + \frac{K-1}{T}\mathbb{E}\left[\vec{r}[\pi(x)]\right]^2 \quad = \quad \frac{K}{T}p - \frac{1}{T}p^2 \quad \approx \quad \frac{K}{T}p,$$

if we consider the simple Bernoulli case and denote by p the probability of an item chosen by π to be clicked on (that is $\mathbb{E}\left[\vec{r}[\pi(x)]\right]$). The last line is a quite accurate approximation given that p is generally small (less than 0.1).

Now let us consider that a satisfying estimation accuracy is reached when the theoretical standard deviation σ of the estimator is less or equal than εp, p being the quantity we try to estimate and ε a real number in $\mathbb{R}^{+*}$. From there we can deduce the quantity of data necessary to reach that precision:

$$\sigma \leq ep \quad \Rightarrow \quad \sqrt{\frac{K}{T}p} \leq ep \quad \Leftrightarrow \quad T \geq \frac{K}{e^2 p}\ .$$

Then to quantify what we need we can take $p = 0.1$, which is a typical CTR in news recommendation, and say that we are satisfied with a standard deviation of 5% of this CTR. Therefore with K actions or items, we would need a dataset of size $T = \frac{K}{0.1\ 0.05^2} = 4000K$ to reach the desired precision. With $\varepsilon = 10\%$, a dataset of size $T = 1000K$ would be enough.

This means that with $K = 50$, a typical value in dynamic recommendation, one needs between $50,000$ and $200,000$ events to be confident in the results. This is very reasonable. Note that this means something else. In dynamic recommendation, the environment is changing (changing pool of items, possibly shifting reward distribution *etc*). In this case, even if we had a dataset containing millions of events, we could not be sure of anything. What we need is to be able to assume that the environment usually remains roughly static during $1000K$ events.

In classical recommendation, things are different. It is usually assumed that the environment is more or less static since predictors learned on past data are used to predict future events. Thus to evaluate a recommendation application with say $100,000$ items, one would need a dataset of size $100,000,000$. This is huge but not completely unrealistic on the web. Note that if the CTR is less than 0.1, say 0.01 or 0.001 (typical in ad display), the required amount of data will be multiplied by respectively 10 or 100. We are not aware of whether this may seem like a reasonable option to people working on real recommender systems. Anyhow we provide the numbers so that anyone can decide depending on his or her constraints.

What is sure is that evaluating learning algorithms with thousands of actions is completely impossible since one needs data so that the estimator converges but also to let the algorithm interact with the environment in order to learn. Indeed we are at least as interested in how well the algorithm deals with the exploration *vs.* exploitation dilemma as in the performance of its fully trained model. Even with 50 actions, it is difficult to estimate the necessary amount of data. It is sure that we will need at the very least $50,000$ events but a slow learning algorithm will require much more to be evaluated correctly. Since algorithms typically have different learning speeds, there is no way to tell the necessary amount of data in general.

3.7.2 Dynamic pool of actions

Let us consider the simple case of evaluating a static policy π for clarity of exposition. In dynamic recommendation the pool of actions changes over time. Typically some actions are periodically removed and/or added. The simple use of the *replay** method (resp. *replay* method) is no longer unbiased (resp. no longer has an exponentially low bias). Why is that and what can we do about it?

Between two changes, the problem is exactly the one we analyzed. Therefore the estimator given by *replay** (resp. *replay*) between two changes is unbiased (resp. has an exponentially low bias). Thus let us split the problem into the minimum number C of stationary sub-problems $\mathcal{D}_1..., \mathcal{D}_C$ that possess one static pool of actions of respective sizes $K_1..., K_C$. Similarly we split S into disjoint datasets $S_1..., S_C$ of respective sizes $T_1..., T_C$. The sum of these sizes is obviously T, the size of S.

We are interested in $g_\pi(T, \mathcal{D})$. If we have access to $\mathcal{D}$, the mean reward of π on $\{\mathcal{D}, \mathcal{T}\}$ is a realization of a random variable whose expectation is what we want. We can decompose that expectation by simply weighting accordingly the expectations of π on each sub-problem:

$$g_\pi(T, \mathcal{D}) = \frac{\sum_{i=1}^{C} T_i . g_\pi(T_i, \mathcal{D}_i)}{T} \ .$$

Now is we perform replay naively on S, we will obtain a biased estimate of g_π, even if $\mathcal{D}$ and all the sub-problems are infinite. For finite dataset, the bias is not going to be exponentially low but much larger. Performing replay* is not even possible because is requires to multiply the sum of rewards by K/T. K does not exist here and using the average of $K_1, ..., K_c$ weighted by $T_1, ..., K_c$ would yield a bias estimate as well. The problem is that the rejection rate differ from on sub-problem to another as the sizes of the pools of actions are different. Indeed, in expectation π is evaluated $\frac{T_i}{K_i}$ times on S_i. Since the various K_i are different, this does not respect the proportions that occur in a real life evaluation, hence the bias of a naive replay estimation. To take this into account, we simply need to compute the estimates on each sub-problem separately and weight them by the sizes of the datasets:

$$\hat{g}_\pi^{(replay^{(*)})}(S) = \frac{\sum_{i=1}^{C} T_i \hat{g}_\pi^{(replay^{(*)})}(S_i)}{\sum_{i=1}^{C} T_i}.$$

The notation with the star between parenthesis means that this estimator has the properties of *replay** (resp. *replay*) if the sub-datasets are evaluated with *replay** (resp. *replay*).

When implementing the method, to avoid storing intermediate estimates for each part of the dataset, one can simply change the way to update the estimate in algorithm 10. This is done by replacing:

$$\begin{array}{ccc} \widehat{G}_A \leftarrow \widehat{G}_A + r & \text{by} & \widehat{G}_A \leftarrow \widehat{G}_A + r.K_t, \\ V \leftarrow V + 1 & \text{by} & V \leftarrow V + K_t, \end{array}$$

where K_t denotes the size of the pool of available actions at time t. Indeed without the corrections, each sub-dataset S_i has an expected weight in the global estimate of $\frac{T_i}{K_i}$. The correction restore this weight to what it needs to be: T_i. Note that for replay*, we divide $\widehat{G}_A$ by T instead of T/K.

Note that even though we lose the convergence guarantees for learning algorithms, it remains important to weight the updates of the estimate so that all the parts of the dataset are taken into account the same way in the evaluation. Note also that this technique is very similar to what needed to be done for non-uniform logging policies (section 3.6.1).

3.8 Conclusion

Table 3.1 is a recap of the results we obtained in this chapter. In a nutshell we clarified the bias/unbiasedness situation of the *replay method* in the state-of-the-art theoretical framework by adding new results. We also exhibited another method which is truly unbiased in this framework and which has slightly different properties. We then conducted a thorough analysis of these two methods to identify the one to use. In this process, we significantly improved the state-of-the-art concentration bound but argued that this concentration approach, with the usual tools at our disposal was not enough to compare the two methods. This is why we studied the variance of the two methods (more specifically the expected squared error) and concluded that the biased method was more accurate in general but that its unbiased counterpart was better at dealing with small datasets. In a last part, we reviewed the various approaches to deal with non-uniform logging policies. In so doing we made a last contribution by exhibiting formally that such a relaxation of one of the main hypothesis of this work was not possible (or at least very difficult) for learning algorithms.

It is also important to note that in the theoretical framework that we consider here, the contextual bandit problem $\mathcal{D}$ is completely static which may not be the case in practice. Furthermore even though the part on the unbiasedness saying that the evaluated algorithm deals with “the same data“ (theorem 2) is valid for all algorithm, the other results only hold for *static policies*. They could however be extended using smoothness assumptions on the considered algorithms but this would needlessly complicate the results. Indeed the non-static algorithms we are interested in are not like the one introduced as an example in the impossibility result. A learning process is generally smooth enough to be characterized so as to derive theoretical results.

In addition, in a news recommendation problem, the pool of possible actions may typically evolve as well as the user preferences. This is not taken into account in this theoretical framework either. This is the reason why a broad set of experiments would appear to be necessary. A few experiments were already conducted by the authors of the method [7, 11] and the results seem promising. As far as we are concerned, instead of running ourselves an extensive set of experiments, we consider the ICML Exploration *vs.* Exploitation challenge 3 that we organize to be an amazing opportunity to do so without being biased by our preconceived ideas. With this challenge we open the *replay method* to be challenged by a great number of scientists. They shall push the method to its possible limitations by looking for all the ways to score as high as possible. Note however that this will not prevent us from running plethora of additional experiments with four main ideas in mind:

- providing the replay method with a set of reproducible, valid and reliable experiments,
- deepening our understanding of how it works in practice when the assumptions made for the analysis are not all met,
- sharpening the very interesting conclusions drawn at the end of this new challenge,
- starting exploiting this method by comparing state-of-the-art bandit algorithms and a few other simple ideas on a real application.

Chapter 4

Exploration *vs.* Exploitation challenge 3 (ICML-2012): organization and investigation of the surprising results

Abstract of the chapter *In this chapter we present the exploration vs. exploitation challenge 3 that we organized in 2012. The results were announced during the ICML workshop "New Challenges for Exploration and Exploitation". This challenge is one of the few to propose an online learning task in which one faces a so called exploration vs. exploitation dilemma. Moreover it is the first one to be evaluated on real data by a supposedly unbiased method introduced by Langford et al. in 2008. This challenge is an exciting opportunity to push this new, promising method to its limits. In this chapter we briefly present the results that were both surprising and quite disappointing. Thus this chapter also contains a set of experiments meant mainly to answer the questions raised by those results but also to test a few recent advances in the bandit field. In particular this process led us to present a new optimistic bandit algorithm (Optimistic Greedy) that outperformed the state-of-the-art methods both in simplicity and performance. One may view this chapter as the experimental counterpart of chapter 3. Last but not least in this chapter we identify a crucial problem in the evaluation of contextual bandit algorithms:* time acceleration. *As such this work introduces and motivates the main research of this thesis work, which provides ways to address time acceleration.*

Keywords (Contextual) Bandits - Replaying past data - Challenge - Offline Evaluation - Exploration - Exploitation - Optimism - Bias - Time acceleration - Overfitting - Benchmark.

Contents

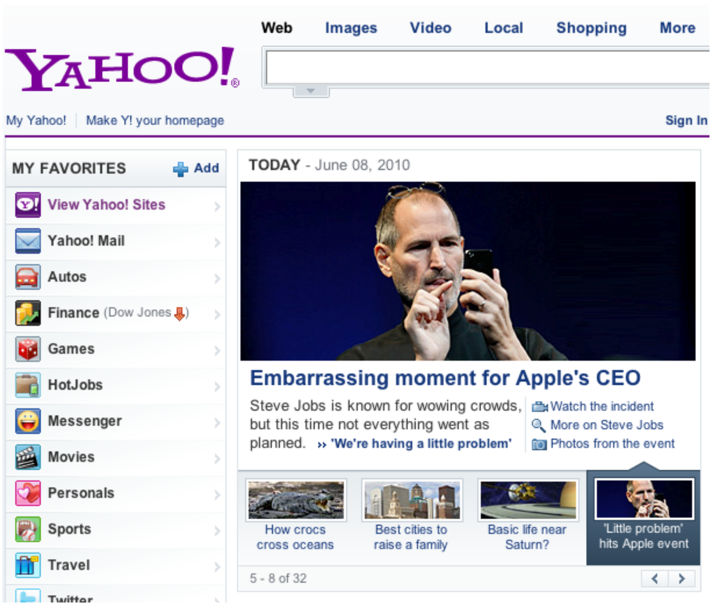


Figure 4.1: The news recommendation application on Yahoo! Today's front page.

4.1 Introduction

The *Exploration vs. Exploitation challenge 3* was the first of its kind and aimed at designing *online* news recommendation policies for a specific application: Yahoo! Today. This application is the main feature of on one of the most visited web pages in the world: *Yahoo!*'s web portal. The application consists in recommending a news article (a featuring picture and a summary) to every visitor of the portal given contextual information about him/her. Figure 4.1 is a snapshot of the application. Note that three other news are displayed along with the main one. They are not taken into account at all here. We consider that we made a successful recommendation if and only if the visitor clicked on what is displayed in order to read the full article. We will give more technical details on the application in this chapter to further justify that it is a perfect example of dynamic recommendation and for which an online learning solution would be very helpful. Finding such a solution is the main purpose of this challenge.

This challenge is original is many ways but its most notable trait is that it is the first one to propose a contextual bandit task evaluated by an "unbiased" data-driven method. We organized this challenge in collaboration with Yahoo! who provided a new dataset that fits the requirement of the method. In other words, it was acquired via a random uniform policy. This method is *the replay method* that was introduced by Langford *et al.* in 2008 [10], analyzed by Li *et al.* in 2011 [11] and that we discussed in-depth in chapter 3. In particular we explained that despite the fact that it is not really *unbiased* whatever the theoretical setting we pick, it does offer much more interesting guarantees than what was tried in the past. See for instance the method used in the Exploration *vs.* Exploitation challenge 2 (chapter 2) or all the methods based on user models that are biased by design. Besides such a challenge is a very exciting opportunity to verify that the evaluation method but also the classical contextual bandit algorithms that are theoretically justified work in practice as well when some assumptions are not met. As we mentioned it before, some experiments to show that this method works in practice were presented by Li *et al.* [11]. Nevertheless when coming up with a new approach such as an evaluation method or even a bandit algorithm, it is often very hard to design experiments without being biased by one or several factors such as our own intuition, preconceived ideas within the community, bad parametrization, an unexpected bias in the evaluation method *etc.* Moreover the existing experiments are not very numerous (due to the small size of conference articles) and are not reproducible as they were conducted on a private application. This challenge is a unique chance to push the *replay method* to its limits by opening it to a great number of people whose goal

```java
public interface BanditAlgorithm {
        public void init();
        public Action choose(Context ctx, List<Action> pool);
        public void update(Context c, Action a, Double reward);
}
```

Figure 4.2: Java interface of a contextual bandit algorithm.

will not be to make the method work as expected but at their advantage. It will also be a very interesting opportunity to see how the state-of-the-art algorithms behave compared to all the approaches the participants came up with and have had plenty of time to tune perfectly. To summarize, the purpose of this challenge is to:

- challenge the *replay method* as hard as possible,
- situate the performance of the state-of-the-art algorithms compared to other approaches,
- hopefully lead to the design of novel algorithms that work well in practice.

This chapter will be organized as follows. First we will explain the task in detail and talk a little bit about the data. Then we will reveal the results of the challenge. These results are quite surprising and actually led us to keep working on a new evaluation method which we consider to be the main contribution of this thesis work. Yet before coming to it, in this chapter we will also present a few interesting experiments that we run in order to understand the results of the challenge. These experiments also allow us to comment on recent and interesting results in the bandit field. In particular we introduce an optimistic bandit algorithm based on a recent work by Maes *et al.* [101] that outperforms all its state-of-the-art equivalents in both performance and simplicity.

4.2 The Challenge

4.2.1 The practical replay method

The spirit of this challenge is very similar to the Exploration *vs.* Exploitation challenge 2 (see chapter 3) as it aims at achieving the same thing: encouraging scientists to work on contextual bandits and as a side effect, testing a new evaluation method. In order to participate, one had to implement an algorithm able to learn online without any prior knowledge of the data. Such an algorithm, written in java or in python had to implement the three functions necessary to describe a bandit algorithm (*init*, *choose* and *update*) provided by the java interface displayed in figure 4.2.

An implementation of the *replay method* is given in chapter 3 (algorithm 10). In this implementation, an algorithm is a function that maps a history of events to a policy, a policy being itself a function mapping a context to an action. We chose the formalization in order to be consistent with the literature about the *replay method* [10, 11] and to be mathematically rigorous (as this avoids variables with internal states). The implementation given here in Java (see figure 4.3) is more practical and is clearer about the respective roles of each function. It is interesting to note that although those two versions are both a representation of the very same *replay method*, they imply different things. Indeed all the theory is valid unconditionally for the version given in chapter 3. To be more specific, the evaluation of a learning algorithm by the *replay method* on a dataset of size T is an almost unbiased estimate (with exponentially small bias) of its CTR when played $\frac{T}{K}$ times in the real world. The quasi-unbiasedness holds because an algorithm is considered to be a function that maps an history of events to a static policy.

```java
public double evaluate(BanditAlgorithm algo) {
        algo.init();
        double r=0,t=0;
        for(Datum d : data) {
                Action a=algo.choose(d.context);
                if(a==d.action) {
                        r+=d.reward;
                        t++;
                        algo.update(d.context,a,d.reward);
                }
        }
        return r/t;
}
```

Figure 4.3: Java version of the replay method to show the purpose of the three functions of the bandit interface. This implementation shows why these functions have to be separated when implementing a bandit algorithm. Indeed they are not used the same way as in a real life evaluation. This implementation is more practical than the classical one (algorithm 10) but suffers from a theoretical flaw due to the hidden internal state of the algorithm.

Indeed all the histories of records have the same probability to occur whether we use the replay method or we are in the real world (see theorem 2). The big problem with that implementation is that a contextual bandit algorithm would be much harder to implement. It would also be very inefficient. Indeed at each time step, the implementation implies that the policy used to choose an action has to be computed entirely using the complete history of past events. This is equivalent to using an offline algorithm at each time step in order to deal with an online task. The practical version makes everything much more efficient and "online oriented". Nonetheless it suffers from a hidden flaw in theory. In the mathematically rigorous implementation the variables, and especially the algorithm which is a function of the entire history, have no internal states. This is not necessarily the case with the practical implementation. The *update* function implies that the algorithm is not a simple function but a complex object. This object is doted with an internal state: an online model which is updated step by step by the *update* function and which is used by the *choose* function to make decisions. In other words using the *update* function of an algorithm naturally changes the behavior of its *choose* function.

This hidden state is theoretically problematic. The *update* and *init* functions are not an issue as they are called the same way as in the real world. However the *choose* function is called K times more than the update function in average due to the rejection mechanism which does not happen in the real world. Since the algorithm has an internal state, it is important to keep in mind that if the *choose* function has the slightest effect on that internal state, we lose our theoretical guarantees. Indeed the internal state would not be modified the same way in reality as the choose function would be called K times less. To be more specific concerning the challenge, a participant could "cheat" by using unsupervised learning techniques on all the users it sees via the *choose* function and thus perform better with the replay method than in reality. This is actually something that we tried with the six displays of each batch in the Exploration *vs.* Exploitation challenge 2 (chapter 2) but we did not manage to improve our approach this way. Anyway, whatever modification of the model the *choose* function makes, it biases the evaluation. Interestingly the winning algorithm of this challenge is an example of such a biased evaluation. This will be detailed in the experimental section.

Similarly to the previous challenge, this one also has 2 phases with very similar purposes. During the first one, the participants are allowed to submit successively various versions of their algorithms, yet one at a time. The only output given to them is a CTR. To allow this, the evaluation on the dataset provided by Yahoo! is performed on a remote server. Moreover, no data is released before the end of the challenge but a very small sample with added noise for debugging purposes. During this phase it is quite clear that even though the only output following hours of evaluation is a single real number, people will tend to overfit the dataset a little. Therefore the second phase is meant to see by how much. To do so the algorithms are evaluated on the first third of the dataset during phase 1. During phase 2, the best algorithm of each participant is the last two thirds of the dataset, that is on brand new data.

Moreover one key feature of the web is that responses have to be given in a limited time span so as to avoid hurting the *user experience.* $100ms$ is often considered to be that limit. Due to various technical issues (OS, garbage collector...) it is not easy to impose fairly such a constraint on the algorithms. In the previous challenge this even turned out to be quite problematic. This is why we chose to simply impose that an entire evaluation should not last more than 20 hours. This is not perfect as it allows participants to run batch-like computations once in a while. However such regular batch computations may not be that unrealistic. Indeed we may consider an approach that would be corrected once in a while by experts or as the result of an offline computation. Also this prevents us from the trouble of accurately controlling the duration of each response and of thinking of what to do when they are too long. Indeed in the real world, a late response may cause the user to leave, or not. It may also lead him or her to never come back again. Therefore repetitive delays may significantly hurt the application regardless of the quality of the recommendations. Since we have no data accounting of all this, we simply ignored this phenomenon in this work. Yet we can only encourage people building a recommender system to keep it in mind.

4.2.2 Technical organization

Organizing such a challenge required setting up a few things. Here is how everything worked. We set up a website, that displays the current results stored in a database. This website was also used by the participant to submit an algorithm, in the form of a java class, implementing the java interface proposed on figure 4.2. The participant had access to the whole evaluation software, that they could run with their algorithm on some sample data to detect bugs. Yet, only the code of the aforementioned class was submitted. A submission on the website would send the task to a scheduler (a bash script) on the frontend of an INRIA cluster, and lock the submission form until the current evaluation was completed. The scheduler would then compile the class within the software set up for the cluster and run the algorithm against the data on a node if one was available or put it on hold until another submission had ended. The algorithms were run in first-arrived-first-served fashion, regardless of the number of submissions of a participants or their average length. At the end of a submission, OAR (on which the cluster was based) would finally notify the scheduler, which would put the result in the database, unlock the submission form and notify the participant by email.

4.2.3 The data

The dataset is the R6B dataset [12] from Yahoo!'s webscope program which was released publicly right after the challenge in 2012. It consists in a list recommendation events, an event being a tuple made of a context (web page description, user profile...), an action (or recommended item) and a reward (click or not). We already mentioned that the recommended items are news article to be displayed on *Yahoo!*'s web portal. The data was acquired using a uniformly random policy for 15 days from October 2 to 16, 2011. This makes the replay method usable in its most simple version. Note that the algorithms are evaluated on the first 5 days of the dataset ($\approx 9M$ records) during the first phase of the challenge and on the last 10 days ($\approx 18M$ records) during the second phase. Each record contains one last piece of information:

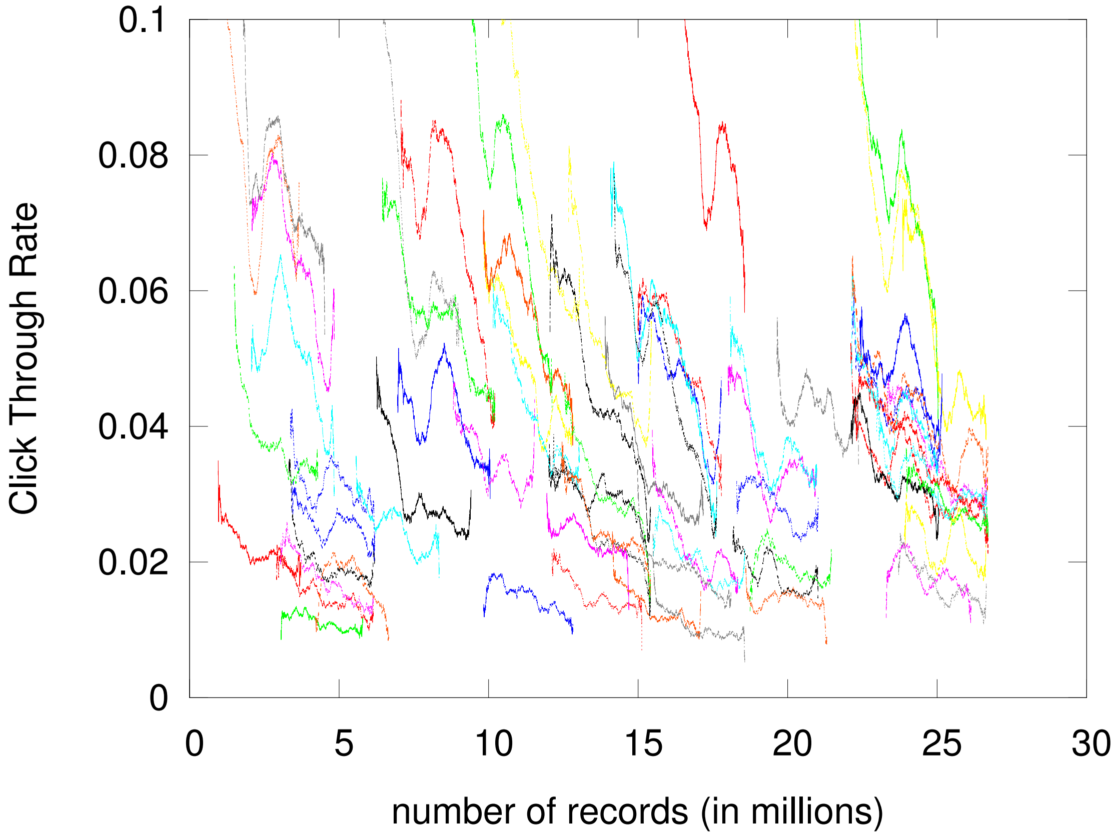


Figure 4.4: CTR over time of the 50 news having the longest lifetimes. Time is in number of records because the volume of audience that sees a news is more relevant than actual time. Notice the overall downside trend that clearly appears.

the pool of available items. Indeed, the news that people read evolve over 15 days. Some new articles are frequently released while others disappear for they become out-of-date. Therefore to avoid giving more weight to areas in which few news are available, it is necessary to weight the updates of the estimator of the replay method (see section 3.7.2 for more details).

This section is a review of a few additional characteristics of the data that are very interesting and worth being mentioned but that can be skipped without hurting the understanding of this document. Some of them are quite surprising whereas others are more or less what can be expected of such a dataset. Note that this is only a small sample of what can be said.

Basic characteristics

News recommendation can be considered as a contextual bandit problem. The context of each iteration is a user visiting the website. It is characterized by a binary feature vector of dimension 136 (one of which is always equal to 1) that gives information about the user's demographics (age, gender, location). Other kinds of features are also available such as behavior targeting features. Note that no information is given on the webpage/website as it is very common in such datasets. Indeed, all the recomendations are performed on the web portal and nowhere else. For sensitivity and privacy reasons, the meaning of each individual feature are not revealed as well as any information contained in the cookies. Therefore no expert knowledge could be used *a priori* even if part of the data has been made available to the participants.

Click Through Rate

The average click through rate of this dataset is a little more than 3%. This means that this is what a random policy would perform in the application. This is a start but there is more

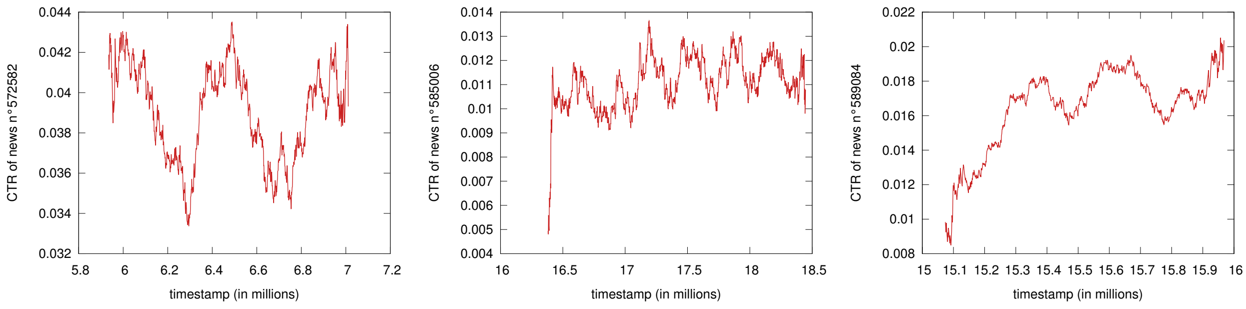


Figure 4.5: CTR of some marginal (see fig. 4.4) news for which it is not decreasing. This makes a potential modelization of this decay trickier than if it was completely systematic.

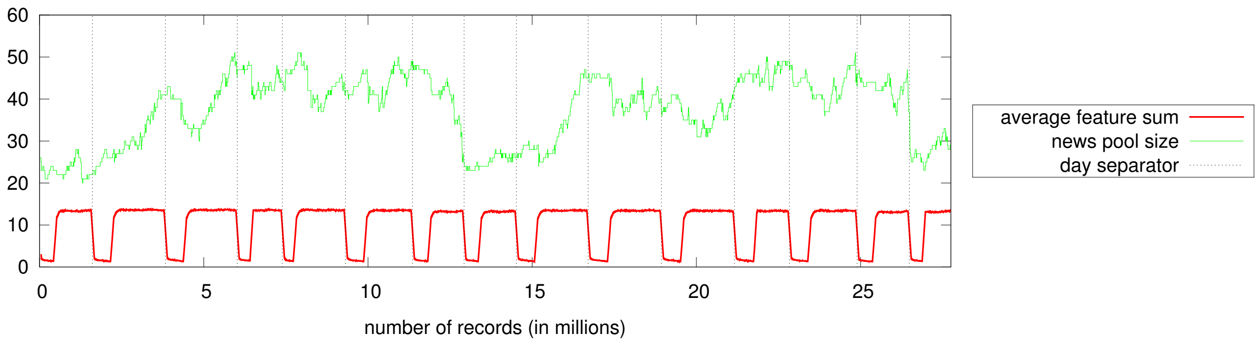


Figure 4.6: Average feature sum and size of the pool of news over time.

interesting. It is well known among applied researchers that the CTR of ads [102] or news [103] tends to decrease over time. As it is shown on figure 4.4, this dataset is another proof of this phenomenon that happens almost systematically here. Note that a few news do not follow that pattern but they represent less than 5% of the total (see figure 4.5). This almost systematic decay is, to the best of our knowledge, not tackled in theoretical papers even though it would be easy to take it into account. With this challenge, we do hope to see emerge algorithms that deal successfully with this temporal component.

Nights and days

The dataset was acquired on a fraction of the audience during 15 days. It is worth noting that we do not know why but no feature are available from midnight until early in the morning (see figure 4.6). It turns the problem into a non-contextual one approximately one fifth to one quarter of the time. Apart from that, the feature space seems to remain quite stable over time. Figure 4.6 also clearly exhibits differences between the week days and the weekends. The week days are longer (in terms of the number of records) and more news are available (≈ 50) whereas on Saturdays the pool of news decreases down to 20 news before going back up on Mondays. This behavior may require algorithms with several regimes (*e.g.* one at night after midnight, one during week days and one during the weekends).

Pool of news

It was already visible on figure 4.6 but it is important to stress that the pool of news is in constant evolution. Indeed it implies the need for an algorithm able to deal with new items popping up all the time. Over those 15 days, more than 600 news come and go. Figure 4.7 shows the distribution of their lifetimes. Most of the news remain *alive* for more or less one day. However about a quarter of them are thrown away more quickly whereas very few of them remain available for display up to two days.

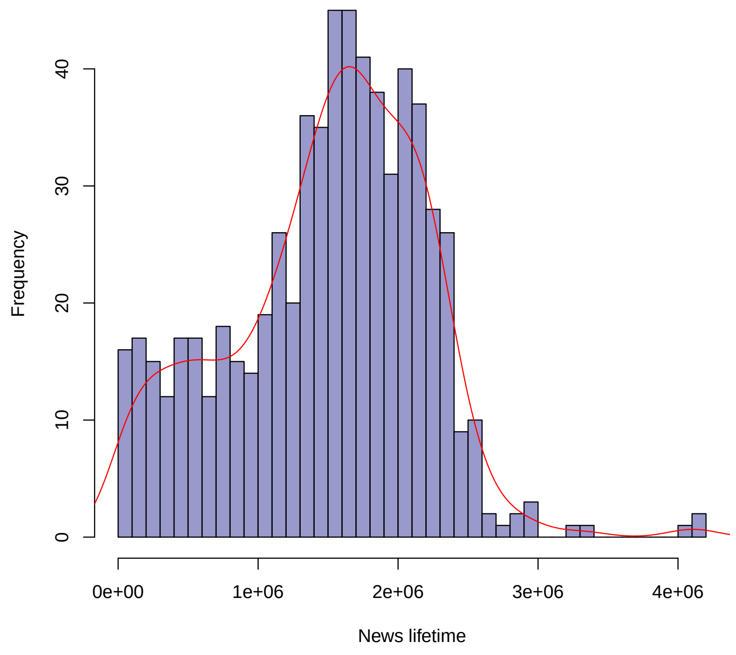


Figure 4.7: Distribution of the lifetimes of the news in number of records (1 *day* ≈ 2*M records*).

Remark The participants of the challenge were not made aware of these very interesting characteristics (apart from the CTR decay). After the facts it may have been interesting to release more information along the course of the challenge to enable the participants to work on more interesting issues than pure parameter tuning.

4.3 Results

Eighty participants entered the challenge and about 20 of them scored more than the standard non-contextual baseline: UCB [13] that we described in section 1.4.2. We recorded approximately 5,000 algorithm submissions during the three months of competition. The results of this challenge were equally surprising as the ones from the previous challenge but in quite a different way. During phase 1, nothing really surprising happened. All the active participants quickly reached a CTR of about 8.5% which is approximately what UCB scores and from there, slowly improved their score up to 9% for the best of them. Note that this represents 3 times the performance of a random policy. Unsurprisingly, the participants who won that phase were those who put the more effort into the challenge. To give a flavor of the kind of approaches that won the first phase of challenge, the three most active participants (in decreasing order of performance) implemented the following algorithms:

1. A very complex UCB-like algorithm with at least 5 different components and many heuristics such as the decreasing CTR of the news, that sometimes exploration becomes useless when a news is better than some threshold... It could almost be considered as an ensemble method and possesses a large number of tunable parameters.

2. KL-UCB [51] who is more recent and has a better regret bound than classical UCB algorithms. It is based on the comparison of the distributions of each arms using the *Kullback-Leibler* divergence. The algorithm was also enhanced by some heuristics.

3. This last approach is more original and is based on Bayesian ideas. The author tries to compute the probability of click conditioned to the contextual information using the Bayes rule. This is obviously impossible with so little data given the size of the context space. Yet approximations make it feasible and quite efficient. See Martin *et al.* [104] for a detailed description of the approach.

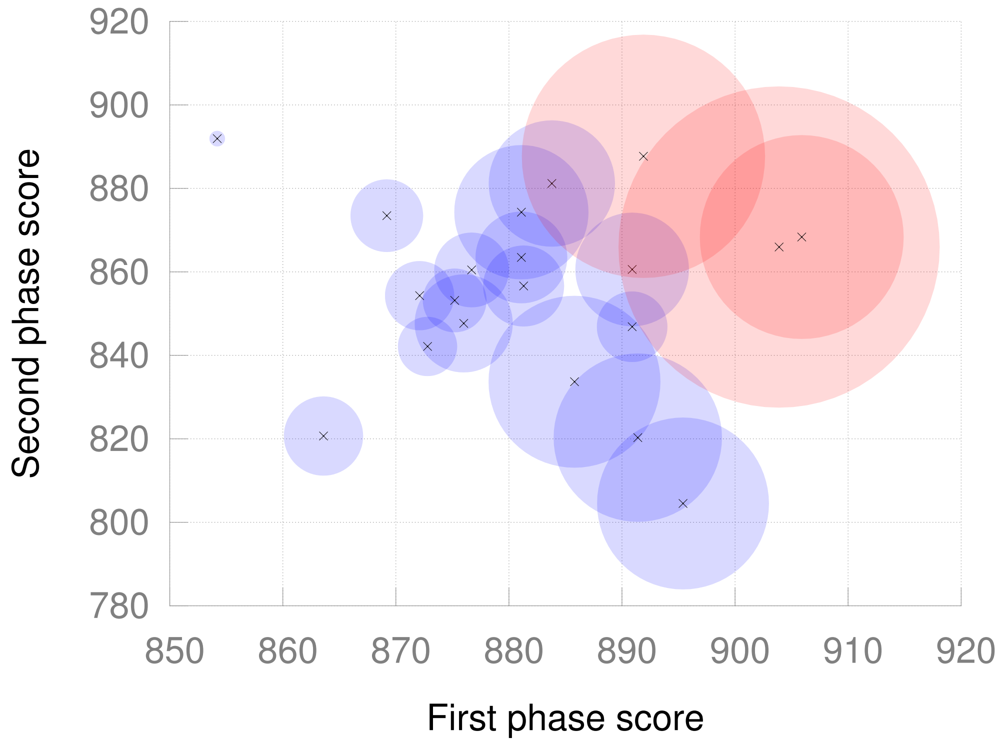


Figure 4.8: Scatter plot of the scores of the participants on both phases. The winner of the second phase is on the top left corner. The radius of the disks is proportional to the square root of their number of submissions. These numbers range from 4 to more than one thousand. The red disks are the three participants with more than 500 submissions. Note that the scores of the blue circles (between 4 and 350 submissions) seem linearly correlated.

The surprise came with the second phase whose purpose was to check the amount of overfitting induced by the repetitive submissions during the first phase. The final results are a caricature of the worse that could have been predicted about overfitting. The winner of the second phase only did 4 submissions in the first phase and implemented without any particular tuning nor modifications the non-contextual algorithm *UCB-V* [49] (one of the numerous variants of *UCB*). We already briefly described this algorithm in section 1.4.2. Yet it is interesting to remember that *UCB-V* uses a variance estimate of the arm distributions to sharpen the confidence bounds. One may thus conclude that using the variance in this application is a good idea. Nevertheless in the case of Bernoulli rewards (as here), the variance of each arm j is calculated using the mean μ_j: $(\hat{\mu_j}(1-\hat{\mu_j}))$. Therefore no additional information is given to the algorithm. In fact, we realized later that a finely tuned *UCB* achieved similar performance. However the use of *UCB-V* can still be justified here. Indeed, in general the exploration parameter α of *UCB* has to be set to a value of the order of magnitude of the average reward of the arms. The addition of the variance in *UCB-V* is a way of doing so automatically in the Bernoulli case. Therefore no prior knowledge of the average CTR of the application is needed to achieve good empirical performance.

Basically this outcome means that all the improvements made by the participants to their respective approaches compared to a non-contextual baseline can be considered as mere overfitting of the task they had at hand. Yet there is even worse. We draw a scatter plot of the scores of the participants on the two phases which is quite explicit (figure 4.8, the winner of phase two is on the top left corner). When we have a look at the blue disks, which corresponds to the participants who outperformed the winner of phase two during phase one but who did less than 400 submissions (all of them but three), we clearly see a linear correlation which is not the one we would have expected; the more they scored during phase one, the less they scored

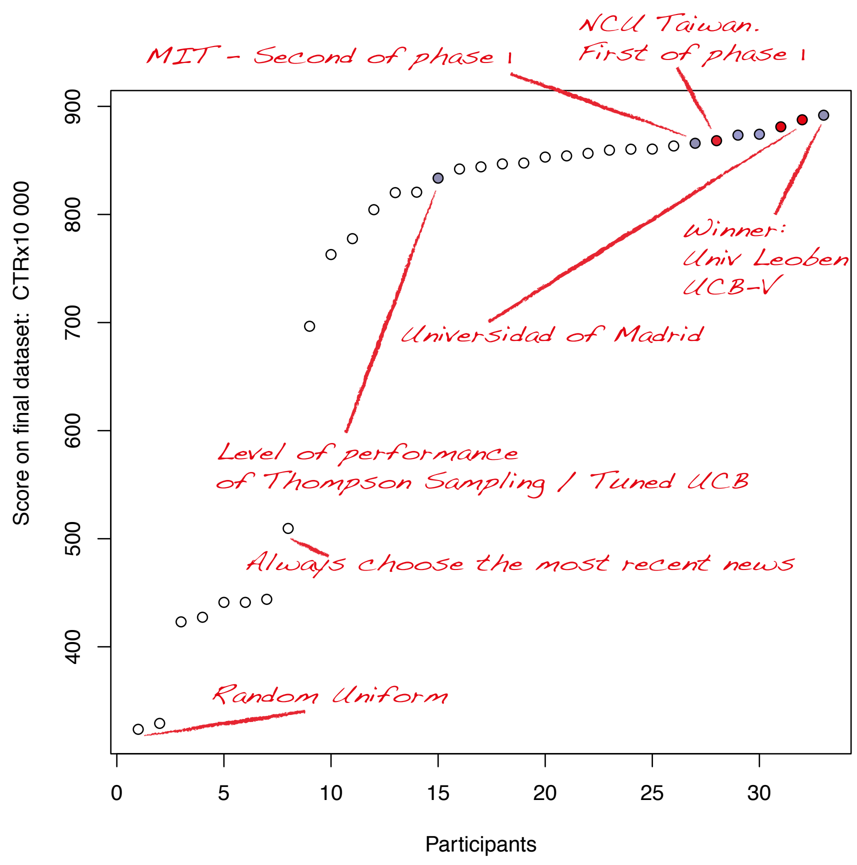


Figure 4.9: Scores of the participants and a few baselines on the entire dataset.

during phase two. In fact this means that not only did the participants overfit the data, their "improvements" actually made their algorithm worse in general. When looking a little closer, we see that the winner of phase two is not the outlier we would have thought him to be. Even though he is a little further away from the cloud of points, he is well aligned with the other ones.

There is one last thing to note on this scatter plot. The radius of the disks accounts for the number of submissions of each participant. The blue disks, that represent all the participants but the three more active ones follow a clear line. Nevertheless the three competitors who made more than 500 submissions and that are displayed in red do not really fit our simple linear model. Their algorithms, even though they were outperformed by the simple UCB-V during phase two turned out to be more resilient to overfitting. One of them even ended up in second position of the second phase. One explanation could be that this high number of tests allowed them to reduce the variance of their algorithm. This can help reducing the loss of performance due to overfitting that they still experienced like the others but not as harshly.

Finally a plot of the results of phase two is given here (see figure 4.9). Note that choosing the most recent news is already significantly more efficient than choosing a news at random. This confirms the CTR decay observed in section 4.2.3. Note however that relying on the age of the news is not enough to achieve decent performance. For a more thorough view of the results of both phases, the reader is referred to the website of the challenge and the associated ICML 2012 workshop [105]. Videos and papers presenting the various approaches are also available.

4.4 Experiments

Due to its surprising outcome, this challenge did not answer to all the questions we had. On the contrary it has even raised some more. For example at this point we are not even sure that it is possible to use the context in order to do better than a non-contextual algorithm.

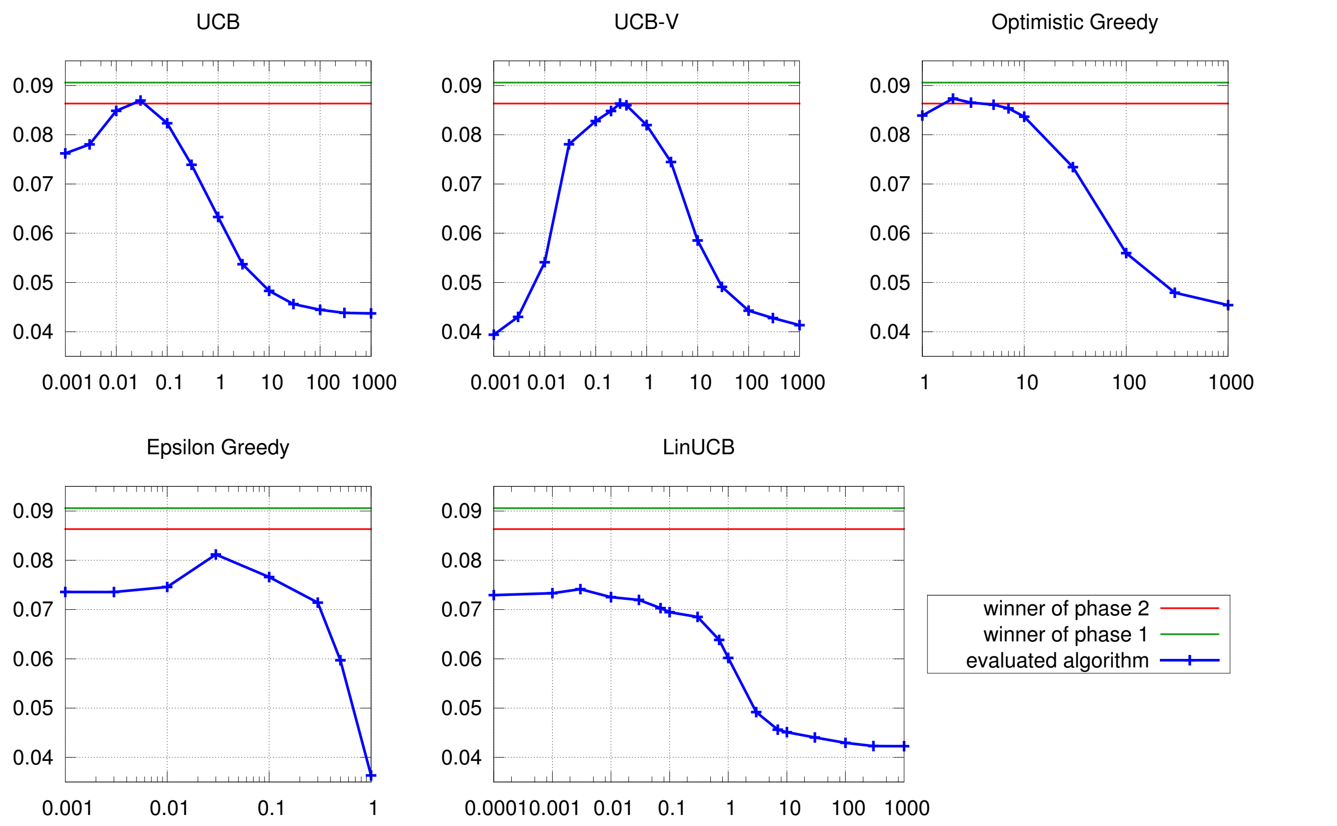


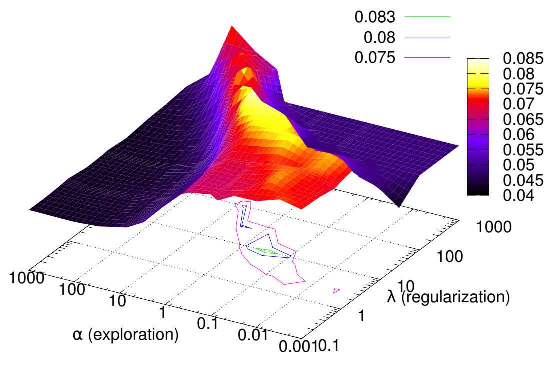
Figure 4.10: Replayed CTR (on the dataset used in phase 1) of various bandit algorithms over the value of their parameter (α for UCB-like algorithms, ε for ε-greedy and k for optimistic greedy). Note in particular the poor performance of LinUCB as it is presented in the literature [7], hence the need to add a regularization parameter as suggested in section 1.4.3. Note also the surprisingly good performance of Optimistic Greedy in spite of its extreme simplicity. The best results of each algorithm are compared on figure 4.13.

This is why in this section we start with comparing standard non-contextual algorithms with the most well known contextual algorithm: *LinUCB* who has also the advantage of having been designed by Yahoo! research for this very news recommendation application. We also run a few experiments based on recent pieces of research about bandits. We then run a simple experiment to highlight the problems that can be caused by the fact that the *choose* function is called more than the *update* function in the replay method. Finally and mainly we try to make sure that the context is informative, and in particular that it could be used to outperform a "most popular news" approach. The main contributions of this thesis work stem from the conclusions of this last section stem that are presented in chapter 5.

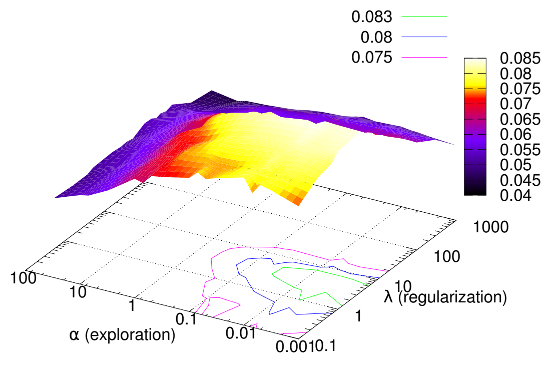
4.4.1 Comparison of bandit algorithms

What we do here is very simple. We try a few standard non-contextual bandit algorithms since most the work done on bandits was done in this setting. Then we compare the hopefully better results given by *LinUCB* which is supposed to be able to take advantage of the contextual information. This little empirical study is also an opportunity to compare the various approaches to the classical multi-armed bandit problem on real data which is not something that was not done a lot in the past. This experiment is also meant to comment on the results presented in two recent papers.

Let us describe the experiment. We evaluate all the algorithms using the *replay method* on the portion of the data used in phase 1. We average the results over only ten runs as evaluating the random policy already takes 45 minutes on a regular laptop. Since most of the algorithms

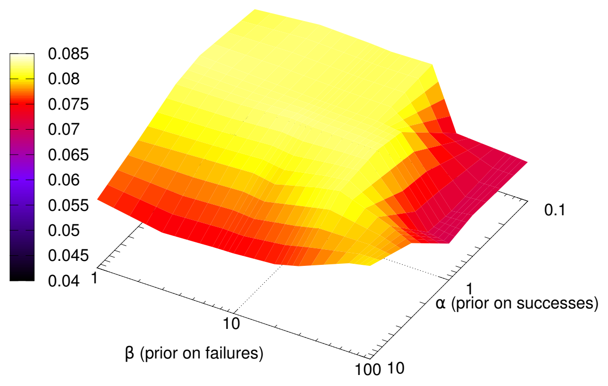


(a) LinUCB.

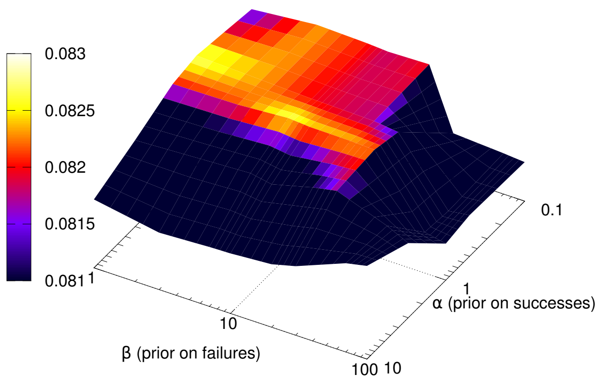


(b) Logistic Regression UCB (LogR-UCB).

Figure 4.11: Replayed CTR (on the dataset used in phase 1) of the regularized version of LinUCB and LogR-UCB. Note that although their maximum of performance is similar (0.084) they do not respond similarly to parametrization. LinUCB's ridge indicates that the more it regularizes, the more it has to explore. On the contrary LogR-UCB does not seem to take advantage of exploration at all and its smoother plot makes the parametrization easier.



(a) Same color scale as on figure 4.11: no white peak meaning lower but steadier performance. Plateau for α (resp. β) between 0 and 3 (resp. 1 and 30).



(b) Color scale focused on the plateau: the results are between 0.0812 and 0.0827. The best performance is achieved when α is between 0.5 and 1.

Figure 4.12: Evaluation of Thompson Sampling on the data of phase 1 using *replay*. Note the plateau meaning that Thompson Sampling is not very sensitive to the value of its parameters.

evaluated here are mostly deterministic except for the first few choices, the variance is very low anyway. All the algorithms are tried with different parameter values. Figure 4.10 shows the results depending on the value of the parameter for the first five algorithms we try. Figure 4.13 is a recap of the best performance of each algorithm. As expected UCB and UCB-V, gifted with a smart exploration policy outperform ε-greedy who naively handles the exploration/exploitation dilemma at random. It is quite surprising however to notice that LinUCB performs extremely poorly on this dataset (0.074 while UCB scores 0.086) especially considering that it was designed for that particular application and that its empirical performance seemed quite good in the paper presenting it compared to UCB [7]. We do not have enough information to give an explanation. It could be that the features used in their experiment were different or preprocessed in some way. Another possibility could be a lack of tuning of the algorithms it was compared to. Anyway this is because of those poor results that we add a regularization parameter λ to LinUCB as mentioned in section 1.4.3 and algorithm 5.

Figure 4.11 shows the results obtained by this regularized version of LinUCB ($LinUCB^R$). The best parameters ($\alpha = 0.3$, $\lambda = 10$) allow this new version to achieve a CTR of 0.084 which

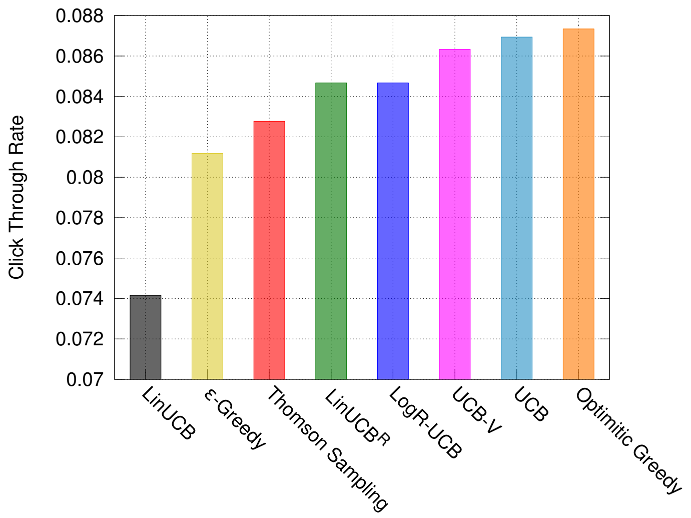


Figure 4.13: Recap of the best performance of all the algorithms evaluated with replay on the data used during phase 1. $LinUCB^R$ is the regularized version of $LinUCB$ whose detailed results are presented in figure 4.11.

is a great improvement but remains slightly below UCB's performance. The figure also shows the results of another algorithm: *Logistic Regression UCB (LogR-UCB).* Since the performance of the regularized version of LinUCB remained disappointing, we designed another simple algorithm. Usually Logistic Regression is more suited to predict probabilities which is what LinUCB is trying to do with the CTR of each item. Thus, *LogR-UCB* computes K logistic regressions, one for each item, that take the contexts as input in order to predict the CTR. At each update, the regression associated to the item that was played is recomputed using all the corresponding past data. We used a simple gradient descent to do so. The exploration is handled in a non contextual way, exactly as in Perceptron-UCB (see algorithm 9) by adding the UCB exploration term to the prediction made by the regression. More formally, given a context x this algorithm consists in choosing the item i maximizing the following expression:

$$x^T\theta_i + \sqrt{\frac{\alpha lnt}{t_i}},$$

where θ_i is the vector of parameters learned by performing a logistic regression on the contexts (input) in which i was played so as to predict the associated rewards (output). This algorithm proved to be easier to tune but did not outperform LinUCB in the end (see figure 4.11). We stopped looking for better contextual bandit algorithms here. We figured that if the participants of the challenge had not managed to do it in over 5,000 submissions, it must be difficult to design a simple contextual algorithm that outperforms UCB. Looking deeper into this would definitely be interesting. However let us recall that our first purpose here is to study evaluation methods.

Discussion about Thompson Sampling

Thompson sampling [45], introduced in 1933 is the oldest bandit algorithm we know of. The results of Thompson sampling on the data of the challenge are given by figure 4.12. To understand our choice of parameters, it is important to be aware of their respective roles. In a nutshell it is a pure Bayesian algorithm that deals with the exploration/exploitation dilemma by sampling from the estimated prior distribution of the CTR of each arm and then playing the arm from which the greatest sample came from. If no prior information is given to differentiate the arms, we can consider that it has two parameters: α and β. Those two real numbers are the parameters of a beta distribution which is the prior of a Bernoulli distribution that we assign to a new arm. Note that when some information is available, it is possible to give different priors to different arms and the number of parameters becomes $2K$. Basically α (resp. β) acts as a prior on the number of times an arm was pulled and received a reward (resp. did not receive a reward). Therefore the estimated CTR of a new arm is $\frac{\alpha}{\alpha+\beta}$. More details are given in algorithm 2. A typical news CTR is between 1% and 10% in this dataset so in our experiments, β has a larger range of values and α can take values between 0 and 1.

At first we had decided to take UCB as a non-contextual baseline and to focus only on contextual algorithms. However we noticed that in the small bandit community in general and in our research group in particular, Thompson sampling had acquired the reputation of being empirically superior to UCB and other index-based algorithms in general. Nevertheless this is highly counter intuitive. Indeed the exploration strategy of Thompson is randomized. This is a very nice property in case of delayed or lost rewards as we already mentioned it in section 1.4.2. Nonetheless this nice property should have a price in terms of performance compared to a deterministic policy that always selects the arm that needs the most to be explored in order to find the best one. This reputation is mainly due to a paper written by Chapelle *et al.* in 2011 [56], published in one of the two main conferences in Machine Learning. Note that other articles following this one present similar results [55]. The authors argue that Thompson Sampling should be included in the standard baseline for comparison with new bandit algorithms. They justify that by exhibiting empirical results in which Thompson sampling clearly outperforms UCB. We do not disagree with the thesis of the paper. On the contrary the fact that Thompson Sampling requires less tuning effort than UCB (see the plateau on figure 4.12 and the peak on figure 4.10) is interesting enough. Moreover the randomized exploration makes it extremely convenient to use in practical applications. Nonetheless it is important to highlight that UCB was not tuned in their study and the results they present can simply be explained by the fact that as the figure shows it, a poorly tuned UCB performs *really* poorly. Yet in all the experiments we made during this thesis work, Thompson Sampling never outperformed a well tuned UCB. This set of experiments is a perfect example (see figure 4.13 for the best performance of each algorithm and the significant gap between Thompson Sampling and UCB). To summarize Thompson sampling is a very elegant way of dealing with the exploration *versus* exploitation dilemma but randomness, as elegant as it can be, makes it less efficient than UCB which, at each iteration, hits exactly where the uncertainty is maximal as far as the identification of the best arm is concerned. Therefore the main strength of the randomness implied by Thompson sampling randomness is not its pure performance but its robustness to technical difficulties and bad parametrization.

The simplest optimistic bandit algorithm

The reader who has had a close look at the final results (figure 4.13) of this empirical study may have noticed that the best algorithm was not described yet. This algorithm has an interesting story. Without going into the technical details, Maes *et al.* [101] designed a very interesting method to automatically create index-based bandit algorithms that work well in practice. Most of the index-based algorithms in the literature (such as UCB and all its variants) can be said to be *optimistic*. Indeed, their exploration strategy consists in selecting the arm which has the highest potential, considering optimistically that it might be reached (see section 1.4.2). Interesting enough, many of the formulas automatically discovered by Maes *et al.* [101] are also optimistic. Yet even though the method allows UCB-like algorithms to be found, the discovered formulas are in fact much simpler. This is why in this study we try to design the simplest optimistic bandit algorithm there is and see how it performs. This algorithm that we call *Optimistic Greedy* works as follows: initially we imagine that all the arms have been played *and* clicked (or got the maximum reward) k times. Therefore k is a parameter of the algorithm. These k imaginary pulls will always be taken into account when estimating the average reward of the arms. Thus the expected reward of each arm is overestimated. From there, we always play greedily the arm with maximum expectation. See algorithm 11 for the detailed implementation. Surprisingly, this simple algorithm outperforms all the other ones we try on this dataset (see figure 4.13).

Remark k is the only parameter here. Nevertheless in the same spirit as what is done in Thompson Sampling, one could achieve a finer level a parametrization by taking two parameters k and k': one for the “prior” on the number of maximum rewards and one for the prior on the number of pull. Then these two different parameters would be used to tweak r_i - the sum of reward for arm i - and t_i - the number of times i was pulled - separately by initializing them as follows: $r_i = k.MAX\_REWARD$ and $t_i.k'$. Note that depending on the chosen values, the algorithm may not be optimistic anymore. For the sake of simplicity and because it already achieves excellent performance, we keep a single parameter in this experiment.

Algorithm 11 *Optimistic Greedy*, a very simple - yet extremely efficient - optimistic bandit algorithm.

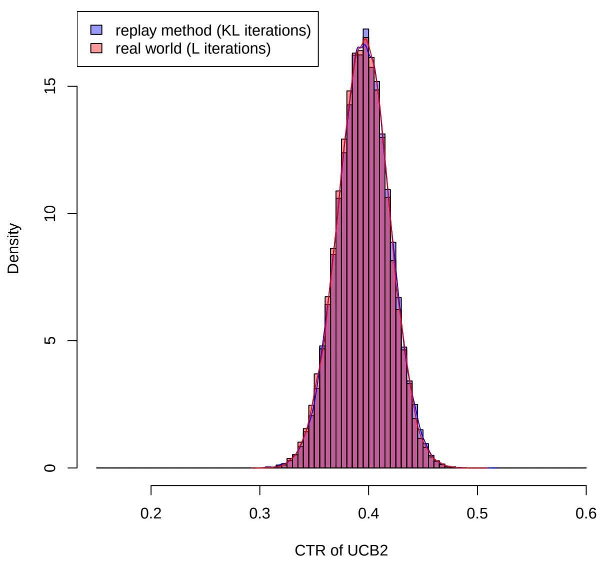
```
param $k$ : The optimistic parameter

function INIT
    $\forall i \in \{1, K\}, r_i \leftarrow k.MAX\_REWARD, t_i \leftarrow k$
end function
function CHOOSE($x \in \mathcal{X}, \{1, K\}$)
    return $\text{argmax}_{i \in \{1,K\}} \frac{r_i}{t_i}$                ▷ Ties are broken arbitrarily.
end function
procedure UPDATE($x \in \mathcal{X}, a \in \{1, K\}, r$)
    $r_a \leftarrow r_a + r; t_a \leftarrow t_a + 1$
end procedure
```

4.4.2 On the Choose function problem

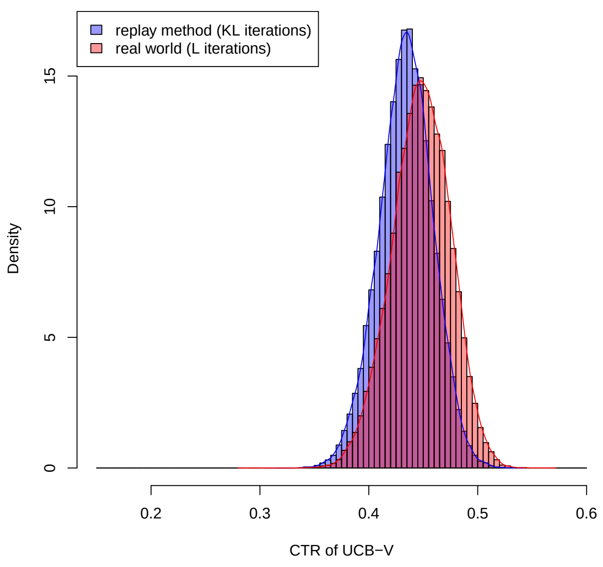
In this last experiment, we tackle a different problem. We explained in section 4.2 that the fact that the *choose* function is called K times more in average than the *update* function could bias the evaluation method. Note that we are not talking about the bias due to time acceleration here.

Obviously a data scientist looking for a way to improve a recommender system would not try to *cheat* while evaluating his new policies by implementing some unsupervised learning within the choose function. Nevertheless this could happen in the context of a challenge although to the best of our knowledge no one managed that here.

The point of this section is that an honest yet careless algorithm design can lead to surprises. What is funny is that the winning algorithm of the second phase is a perfect example of that remark. At the end of the challenge but before the workshop, we were really curious to know how it was possible that a participant with only 4 submissions had been able to win the second phase. That person was kind enough to send us his code and we were surprised to recognize an implementation of UCB-V exactly as it is described in the literature [49] (with an exploration parameter set to 0.3). Yet as we discussed it in the first chapter, bandit algorithms are usually presented in one block, mixing the choice of the arm and the update of the model. Without going into the details, UCB-V requires a variable t that counts the number of arms pulled in total (among other things). In the implementation we received, this variable is updated in the *choose* function. Therefore it is multiplied by K in average when evaluated by replay compared to when it is played in the real world. The reader familiar with UCB-like algorithms may notice that this new UCB-V should tend to explore more when evaluated by replay. Indeed, the exploration part of the indexes increases with the logarithm of t which is overestimated



(a) (Almost) unbiased evaluation of UCB.



(b) Highly biased evaluation of UCB-V.

Figure 4.14: Evaluation of UCB($\alpha = 1$) and UCB-V($\alpha = 1$) using on one hand T iterations on the real model and on the other hand the replay method on a dataset built with TK interactions of a random policy with that same model. Given the results proven in chapter 3, for each algorithm the two evaluations should converge toward the same thing. We obtain exactly the results we expected. The flawed UCB-V has more variance on the real world but performs better on average as it explores less. Note that replay would not necessarily underestimate the CTR of this UCB-V: if the exploration had been carefully tuned for an evaluation with the replay method, the algorithm would not have explored enough against the real model and would have suffered a dramatic loss of performance.

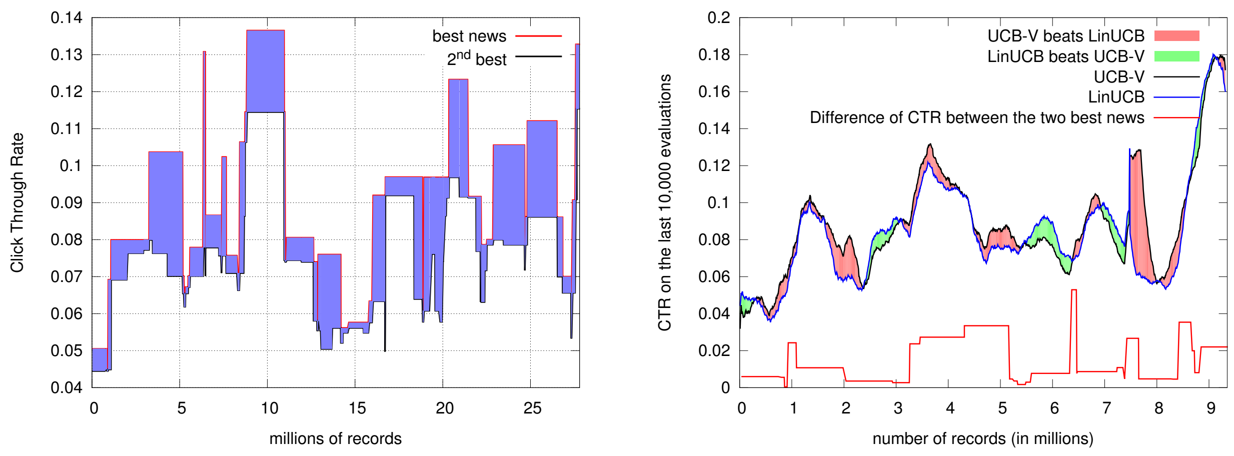
(see the implementation of UCB in algorithm 1 for more details). Note in addition that as K varies over time in this dataset, this algorithm also has a varying exploration behavior within one evaluation. We ran a small experiment using the same Gaussian model as described in section 2.6.5 of the chapter 2 (with $F = 10$, $q_{max} = 4$, $K = 10$ and an horizon of 500) to exhibit that this implementation of $UCB - V$ biases the evaluation process. See figure 4.14 for the commented results of the experiment.

In this example that we were lucky enough to find in the code of the only participant who gave us his could be easily fixed. Indeed, to get the same performance in the real world, we would just have to add K to the variable accounting for the number of pulls in the *update* function. Nevertheless people generally do not read each others code, especially in the industry so it is quite important to stress that this kind of careless algorithm design could lead to bad surprises if they are first evaluated using the *replay method* or another similar method.

4.4.3 Contextual information

Study of the instantaneous CTR

Given the results from the challenge and our empirical study, we may wonder whether the contexts from this dataset contained any relevant information at all. One basic solution would be to try to find different correlations between the context and the CTR of each news using statistical tools. However even if we found something, this would not ensure that it is possible to design a contextual algorithm that would outperform UCB. As an example finding that in a given context, a news doubles its CTR from 3% to 6% is useless if the most popular news has an overall CTR of 10%. In this section however we provide evidence that it *must* be possible to use the contexts to do better than a most popular approach.



(a) CTR of the two best news at each time step on the entire R6B dataset [12] and the gap between them.

(b) CTR of LinUCB and UCB-V computed via replay on the phase 1 dataset and impact of the "CTR gap".

Figure 4.15: Impact of the "CTR gap" between the two best active news on LinUCB's CTR.

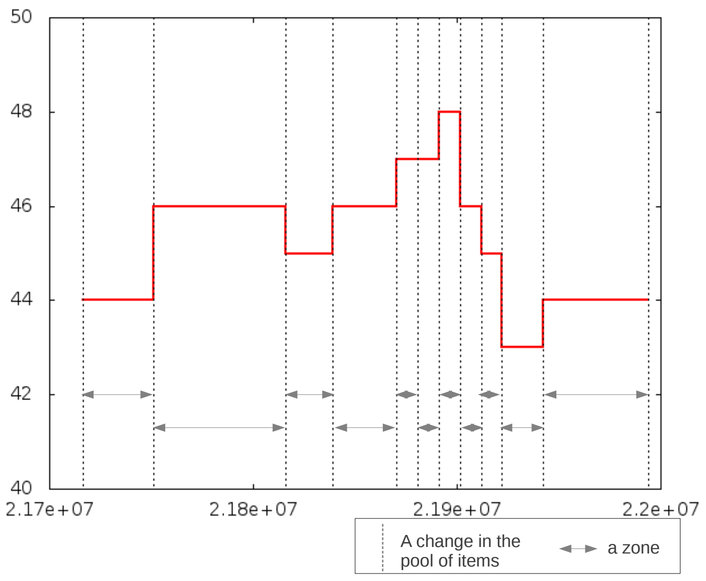


Figure 4.16: Dividing the dataset into the minimum number of zones allowing each of these zones to possess a static pool of news. This procedure divides the dataset into 926 zones.

First it is interesting to see that during phase 1, LinUCB sometimes outperforms UCB-V (Figure 4.15b) which indicates that contexts can be informative. Furthermore it is interesting to notice that this seems to happen when the difference between the best news and the second best is small, that is when trying to personalize their recommendations can easily make a difference. The rest of the time, LinUCB loses too much time trying to learn correlations that are not worth it as the best news is very popular. Now one has to keep in mind that the dataset we have at hand is only what we can call exploration data. It was acquired on a small portion of the audience, maybe 1%. Moreover the replay-method *accelerates* time by a factor K. This means that we are actually evaluating algorithms on $100 \times K$ less users than in reality. Having that much more time would considerably reduce the relative cost of exploration that LinUCB has to pay to learn how to personalize its recommendations and the successful personalization that we notice on the graph would not be negatively compensated by the time it took to learn it. Furthermore it would probably be possible to learn more correlations with more time and LinUCB would probably beat UCB more regularly, and with more significance than what is shown on figure 4.15b.

Classifiers ensures the existence of useful correlations

To give more weight to our claim that the contexts are informative in this application, we also use the *classify to recommend* procedure, a very interesting technique that we only mentioned in chapter 2 and published in our article about the challenge we won [5]. The idea of this method is simple. We consider a dataset of recommendations. We only keep the *positive* rewards (the clicks). Then we consider the contexts as the input variables of a classifiers, and the action as the output variable (or the class). We can then use classical classification techniques, cross validations *etc.* Very interestingly, it can be proved that the hit rate of a classifier is proportional in expectation (with a known coefficient) to the CTR of the corresponding policy in the real world. Indeed, a classifier takes a context as input and outputs an action. Therefore it can be used as a contextual bandit policy. This means that all the work from classification, which is extensive and much richer than the work on contextual bandit, can be used *out-of-the-box* for diagnostic purposes in dynamic recommendation. For instance, we can get a sense of the kinds of models that can find interesting correlations (linear, logistic, neural networks, trees, forests *etc.*) to design an appropriate learning algorithm. Also if one is simply interested in stationary policies, the method directly outputs such policies (the classifiers) with an estimation of their CTR by cross validation.

Another very nice result about this method is that, although it only considers a small fraction of the data - the clicks - its variance is the same as the variance of replay*. Therefore this method can also be seen as a way to accelerate replay* by a factor 10 if the average CTR is 10% for instance. A much more detailed description of the classify to recommend method is available in appendix B, alongside with the proof of the aforementioned results and a few experiments.

Since this method requires a static pool of ads, we simply divided the dataset into zones as explained on figure 4.16. The results are presented and commented in table 4.1. Note that in half the zones we consider, the procedure manages to learn correlations that lead to a better policy than simply displaying the most popular news. Note also that it is easier to do better than a non-contextual approach when the best news has a relatively low CTR. Notice that the current pool of news seems to have a great impact on whether or not it is possible to take advantage of the context. When it contains either no very popular news or best news with similar CTR, contextual algorithms seem to be able to outperform UCB even with little data. Figure 4.15a shows that in the whole R6B dataset, these conditions are met approximately half of the time. This indicates that designing a personalized recommendation algorithm for this particular application is feasible.

Discussion: time acceleration prevented contextual algorithms to perform well

To conclude, in this section on contextual information, we believe that we highlighted a few evidence that the contexts of this dataset can be used to do better than just serving the most popular news to everyone. From there, we can propose an explanation for which we have no more evidence than the following line of reasoning:

- LinUCB, and all the approaches proposed by tens of participants over 3 months did not outperform UCB, on the contrary.
- Yet we have a proof that the contexts can be used to do better than non-personalized recommendation. So doing is possible using linear correlations (Linear SVN, Logistic Regression in table 4.1).
- This means that LinUCB should be able to learn from the data.
- If LinUCB is outperformed by UCB, it is necessarily because it does not have enough time to learn helpful correlations online for they do exist.

- The replay methodology accelerates time by a factor K, more if we take into account the fact that the dataset was not acquired on all the audience. With more time the additional exploration required by LinUCB will have less weight in the final CTR and will already naturally improve its overall performance. Therefore more time can only help to improve LinUCB's performance comparatively to UCB's.

- Nevertheless nothing tells us for sure that there will be enough time to learn the correlations even with much more time but we do have very good signs: (i) LinUCB sometimes manages to do it even when being accelerated (figure 4.15b) (ii) we found the correlations on very small datasets (figure 4.16). Indeed we trained our classifiers on each individual zone. An online algorithm can obviously use information from a previous zone to perform well in the next ones.

A last evidence of the high likelihood of the fact that personalized recommendation should perform better than UCB in the real world is the following. The dataset we have at hand was only logged a small portion of the audience, say 1% although we do not know the exact figure. We exhibited that offline approaches (table 4.1) allowed to outperform a most-popular-item approach in zones of that data. Therefore let us consider the epoch-greedy algorithm. We explore uniformly during τ time steps with τ of the order of the size of the big zones we considered and then exploit during 99τ time steps. By the end of the exploitation phase, we should have covered the amount of time it took to log one of the aforementioned zones given that only a portion of the audience is logged. Let us say as figures 4.15a and 4.15b suggest, personalization can outperform most popular approaches half the time. In this case, let us consider a lift of 5% as table 4.1 suggests, so 2.5% on average. Let us also consider that a most popular approach does not have a learning cost and that it performs 3 times as much as a random policy. In this case, the performance g_{epoch} of this very basic approach would be:

$$\begin{aligned} g_{epoch} &= 0.01 * g_{rand} + 0.99 * 1.025 * g_{mostpop} \\ &= \frac{0.01 * g_{mostpop}}{3} + 0.99 * 1.025 * g_{mostpop} \\ &= 1.0181 g_{mostpop} \, . \end{aligned}$$

In other words, such a coarse algorithm would outperform the most popular approach by 1.81%. Note in addition that many improvements could be performed such as taking into account the previous zones, cutting the exploration phase in case of a very superior piece of news *etc.* This reasoning is highly heuristic and cannot be verified. Yet all we did here indicates that it is very likely, and even quite pessimistic given all that could be improved in the approach.

Whether or not it is possible to beat non-contextual approaches in this application, learning online when $K-1$ out of K events are filtered out is a completely different problem. This is even more problematic when the environment is dynamic. Indeed with a static environment, an online algorithm would eventually converge. Thus a sufficient amount of data would allow a fair evaluation. Here, with a dynamic environment, no convergence can be expected. The underlying model of the algorithm will keep evolving along with the environment. Therefore the time acceleration phenomenon introduces a bias in the evaluation that cannot be fixed by acquiring more data.

4.5 Conclusion

The main contribution of this chapter is the empirical study of mostly non-contextual bandit algorithms made on real data and that we already commented at length in section 4.4. We also used these results to highlight that optimism in face of the exploration *versus* exploitation dilemma is a very good choice. So good that even its simplest implementation (Optimistic

Table 4.1: Table of results given by the *classify to recommend procedure* (app. B) on test data using state-of-the-art classification algorithms on the 8 largest zones of the R6B dataset (the results displayed are the lift compared to the policy picking the best news). We manage to take advantage of the context significantly when the CTR of the best news is low. Otherwise it seems very difficult to do better than the policy that always chooses this news. The algorithms used are (i) Random forest [106] (RF), that achieves the best performance, (ii) SVM [107] with both Gaussian and linear kernels, (iii) Naive Bayes [108] and (iv) "1 *versus* All" Logistic regression (LogR). No effort was put into parameter tuning and data preprocessing, discarding the eventuality of overfitting. We could also have increased the size of the zones by weighting the different news in order to get more training data and thus better results.

Zone index	11	324	393	451	496	627	776	834
CTR of the best news	0.052	0.124	0.070	0.064	0.061	0.090	0.110	0.123
LogR	+8.2%	−4.3%	+7.2%	+19.2%	+7.1%	−5.0%	−5.2%	−4.9%
RF	+4.8%	+0.0%	+4.2%	+21.2%	+1.4%	0.8%	−1.0%	−0.3%
Naive Bayes	−3.2%	+0.6%	−4.5%	−4.2%	−5.8%	+0.3%	−0.3%	+0.6%
Linear SVM	+9.2%	−6.6%	+4.6%	+18.1%	+5.2%	−0.7%	−2.8%	−1.0%
Gaussian SVM	2.8%	+0.7%	−1.2%	3.7%	8.2%	−0.8%	−0.8%	−0.2%

Greedy) seems to perform extremely well on this dataset, even better than its more complicated and popular counterparts such as UCB, Thompson sampling or UCB-V. The results also exhibit that as the intuition would suggest, Thompson Sampling is less efficient in practice than a carefully tuned index based algorithm such as UCB, most probably due to the random exploration. As intuitive as this seems, this goes in contradiction with recent empirical studies whose results are surely due to a lack of proper tuning [55, 56].

Nevertheless another contribution of this chapter will be the starting point of the rest of this thesis work. The disappointing results of the challenge in terms of contextual algorithms led us to study a little bit further the replay method. This method has strong theoretical guarantees that we described in chapter 3. Yet the fact that it accelerates time for a learning algorithm is in fact a huge bias that penalizes exploration and thus more complex models such as the one needed to personalize the recommendations.

After this challenge and our study, the so-called *unbiased* replay method seems to have a much more limited scope of application than we first thought. It does work as expected with non-contextual algorithms or static policies. Nevertheless the unexpected bias introduced by time acceleration makes it impossible (at least with this news recommendation dataset) to evaluate fairly the algorithms we are really interested in, that is those that are learning how to do personalized recommendations. This statement is the actual starting point of the last part of this thesis work. It is also interesting to note that the evaluation procedure from the first challenge that we vehemently criticized and discarded does not suffer from this problem. Indeed it accelerates time by a arbitrary factor L that does not depend on the problem. An idea to improve it could be to pre-cluster the users and only ask an algorithm to choose within a batch of a few records containing similar users so that user selection only would not be a reasonable option. Yet we left this thought for future digging and decided to pursue our work in a different direction.

The remainder of this thesis work will actually be about new methods that improve the data-efficiency of replay methodologies and are able to reduce time acceleration. We will study how this new methods behave theoretically in the setting introduced in the literature [10, 11] and that we used in chapter 3. We will also try to guard these new methods against the problems that could arise when some assumptions imposed by the theoretical framework may not be met. As the challenge exhibited it, is often the case in practice.

Chapter 5

BRED: A bootstrap oriented approach to data-oriented evaluation of contextual bandit algorithms and policies

Abstract of the chapter *This chapter and the one just after are about the main contribution of this thesis. This contribution consists in a broad set of tools to overcome the time acceleration phenomenon and improve the data efficiency of replay methods used to evaluate contextual bandit algorithms using real data. In this first (and main) part we define a simple contextual bandit problem in which time acceleration is an issue when using a replay method. This new problem is artificial and simpler than what we call dynamic recommendation but time acceleration is problematic in the very same fashion. We argue that solving time acceleration in one situation is equivalent to solving it in the other (as long as the solution does not involve more data). Then we introduce a solution based on bootstrapping, usually used in the context of bagging in Machine Learning. We justify it theoretically using the framework introduced in the literature and that we already used in this thesis. We point out in this chapter that this theoretical analysis opens many perspectives. In particular we argue that replay methodologies and the CTR metrics they compute can be used in classical recommendation even though the number of items may seem prohibitive in that case. We also present very encouraging empirical results on both synthetic and real data. Furthermore, this novel approach comes with a highly desirable property: the ability to provide estimators of the quality of the evaluation. This property is especially interesting in the context of recommendation for it allows to minimize further the risk entailed by putting online new algorithms. We provide convergence results for this estimator of quality that are based on bootstrapping theory. Finally if this work improves the evaluation of online learning algorithms, it also provides a significant contribution in another context referred to as iterative improvements.*

Keywords Bootstrap - Bagging - Smoothing - Contextual bandits - Dynamic recommendation - Offline evaluation - Expansion - Density estimation - Replaying past data - Knowledge quantification - Variance - Bias control - Risk minimization - Time acceleration - Iterative improvements.

Contents

5.1 Introduction

In chapter 4 we described the Exploration *vs.* Exploitation challenge 3 which was about designing contextual bandit algorithms to solve a news recommendation task. The algorithms were evaluated offline using the replay method presented in chapter 3. This challenge highlighted a bias in the method that was unexpected. Indeed, surprisingly this competition was won by a very simple non-contextual algorithm. Yet in chapter 4 we also exhibited that contextual information should have allowed to do better than a most-popular-item approach. Many hints supporting that claim were indeed provided. Although it is difficult to provide real hard evidence, we do believe that contextual algorithms would be able to outperform the best non-contextual ones when played in the real news recommendation application and that only this new bias prevented that from happening in the challenge and in our experiments.

This unexpected bias is caused by what we call *time acceleration*. More specifically a dataset acquired via T interactions with the real application only allows a replayed algorithm to perform $\frac{T}{K}$ recommendations in average. Thanks to theorem 2 we know that a replayed algorithm is evaluated exactly as it would be in the real world. In other words an algorithm has the same vision of the world whether it is replayed or played against the real application. Yet when the environment evolves over time, which is typical for news recommendation, things change. Since the replay method causes $K-1$ out of K events to be ignored, a replayed algorithms in fact interacts with an environment that **evolves K times faster**, hence the term time acceleration. In other words, let us consider a simplistic version of reality in which a change in the environment (new item, event in the real world...) occurs every v visits. Let us also consider that an algorithm needs a significant fraction of v, say 1 or 10% to explore and adapt its behavior to a change. When replayed, the interval between two changes shrinks to v/K. Yet the amount of time necessary to learn remains unchanged. Thus the algorithm, even it still has enough time to adapt in this case (but which is not sure at all in general), is underestimated for the learning phase lasts K times more in proportion. Therefore it penalizes much more the estimate. As a consequence fast learning algorithms are favored and personalized recommendation algorithms that are of interest to us and require some time to learn correlations, tend to be dramatically underestimated.

As a result, the best performing algorithms on the R6B dataset provided by Yahoo! seem to be the non-contextual ones. Such a bias is not acceptable and almost makes the replay methodologies irrelevant even when using a dataset that was built for that very purpose. Note that with a less favorable dataset, time acceleration might be even worse for we saw in section 3.6.1 that a non-uniform dataset made an evaluation less data efficient at best and impossible at worst.

In this chapter we will consider the same theoretical setting as in chapter 3 and bring a solution based on bootstrapping to reduce the time acceleration phenomenon. More specifically we consider the following problem which is significantly simpler than the one we just mentioned. We consider a static contextual bandit problem made of a distribution $\mathcal{D}$ defined as in section 1.4.3 and that has a finite horizon T. Therefore when playing a randomly uniform policy against this problem, we obtain a dataset S of length T. Remark that in this context, simply adding more data would solve our problem. Nonetheless we assume that no more data than S can be acquired. Thus we need to get more accurate estimations of new algorithms than what is done by the replay method using only S. One can consider again that in a simplistic version of reality, the problem of evaluating an algorithm in a dynamic environment using replay, which cannot be solved by adding more data, is a sequence of this simple problem we consider. Therefore tackling time acceleration in the context of the simple problem is equivalent to tackling it in the more complex one. Our theoretical setting only considers the evaluation of static policies. Therefore let us remind that the estimator yielded by the evaluation of a static policy has a concentration bound in $O\left(\sqrt{\frac{K}{T}}\right)$. What we want to do is to achieve a bound that is free from the dependence in K or at least that includes an integer constant Γ, smaller than K that reduces time acceleration. More formally we would like to achieve a concentration bound in $O\left(\sqrt{\frac{\Gamma}{T}}\right)$ ($\Gamma \in [1, K[$). The ideal case we will want to come close to is obviously when $\Gamma = 1$ and time acceleration is completely wiped off. Yet as this may seem Utopian, a value of Γ significantly lesser than K would already be satisfactory.

Remark In this chapter we will define a notion of knowledge or information contained in average in an interaction logged by the logging policy with regard to the evaluation of another policy. It will allow us to analyze further the accuracy gain yielded by our method. Γ is in fact the inverse of the knowledge contained in one interaction and, in a way, quantifies time acceleration. Consequently Γ is an element of the problem, the same way as K - the number of actions - or T - the horizon of the bandit game. Therefore, although it might have seemed surprising, we do consider that if $\Gamma \neq 1$ and $\Gamma \neq K$ then $O\left(\frac{1}{T}\right)$, $O\left(\frac{\Gamma}{T}\right)$ and $O\left(\frac{K}{T}\right)$ are all different from one another.

To reduce time acceleration we propose a method based on *bootstrapping*[1] techniques. In statistics, bootstrapping is a resampling technique that aims at estimating properties such as the a confidence interval of an estimator or constructing tests of hypothesis. The idea is relatively old but still very popular nowadays in both statistics and machine learning. It was introduced by Efron in 1979 [14]. It is inspired from the *Jackknife*, an older, coarser and less computationally expensive resampling method that was introduced in 1949 [111] and studied in more depth in 1956 [112] by Quenouille for bias estimation purposes. As bootstrapping is the core of this chapter, let us describe in a few words the classical approach as it is viewed in the statistic field. Note that we will not be very technical. The basic idea is that given some sample data, inferring characteristics of the population it comes from can be done by resampling from this sample data. In other word bootstrapping consists in seeing inference about a population from sample data as analogous to inference about that sample data from resamples.

Since this may seem a little bit abstract, let us take an example. Suppose we are interested in the average weight of the people in a country. It would be very hard to measure the weight of everybody. Instead we can sample only a very small part of the population. Say we measure T weights. That gives us a mean value. Yet, to be able to infer something about the population, we need to have information on the variability of this estimation. The most basic form of bootstrapping (non-parametric bootstrap) consists in sampling with replacement thousands of new samples of size T (called bootstrap resamples) and to compute their respective means. This process generates a histogram of values which is an estimate of the distribution of the sample mean of the weight of the population (when, of course, the sample is of size T). Using this density estimation, we can answer all sorts of questions about the distribution of the estimator of the mean such as having a sense of its variance. Note that this method for the mean is valid for many other statistics or estimators. Note also that such a resampling method was only made possible in modern statistics by the development of computers. A replay method that would also output such a distribution would be very desirable so as to minimize the risk

[1]Interestingly, the use of the term "bootstrap" comes from a popular American expression: *pull oneself over a fence by one's bootstraps*, which is absurd and impossible. Bootstrapping is a metaphor for self sustained processes, that is that proceeds without help from the outside. This is basically the idea of the statistical bootstrap that we consider: the density estimation of a statistic requires several samples, therefore some more are built from the only one at hand. Due to its name that conveys an idea of absurdity, bootstrapping is sometimes mocked by statisticians, although it is rigorously, theoretically justified [109]. The word bootstrapping can also be found all over the scientific literature to describe processes that work without external help. For instance in computer science a bootstrapper is a part of a piece of software that can run on its own to install the rest. Even in Machine Learning and in particular in the field of recommendation, the term bootstrapping has been used to denote something completely different than what a statistician would have in mind. Bootstrapping a recommender system is a way to bypass the cold start problem by asking directly to new users a few of their tastes in order to be able to provide them with good recommendations right away [110].

of putting online a new algorithm, which in fact is our original goal.

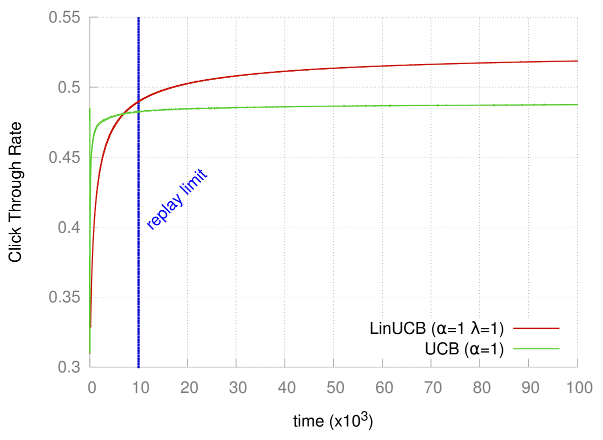
This chapter depicts how we used bootstrapping to tackle the issues raised by the time acceleration phenomenon implied by replay methods but not only. We also justify that our method increases the accuracy of replay methods in several other ways. We first exhibit with a simple but revealing experiment the reasons why time acceleration is problematic when dealing with learning algorithms. Then we describe how bootstrapping is classically used in the literature and how we use it to solve our problem in our theoretical setting (stationary policies). This solution is inspired from a work by Kleiner *et al.* [113], that involves data expansion. We also provide a theoretical analysis for this method, called BRED in a classical setting and show that it improves the state of the art. Note that the method needs to be adjusted in order to handle learning algorithms. Solutions to fully address this issue are proposed and justified after the theoretical analysis. This first analysis is modeled on Kleiner's work but we then exhibit that the Utopian quantity of improvement we proved is solely due to a lack of precision within the proof. Another problem with this work are a few assumptions that are made blindly and without explanation. After exhibiting why this is and what it means, we justify the assumptions and lift the inaccuracy that was problematic in the previous analysis. This new analysis is based on classical proof techniques used with the bootstrap and a way to quantify the knowledge included in a sample with respect to the evaluation of a policy. Fortunately, even with this sharpened analysis, BRED still offers significantly better guarantees than replay. Then we realize that this analysis proves much more than what we were after in the first place. It proves that replay methodologies and the CTR metrics they compute can be used for classical recommendation, which was never done in practice because $\sqrt{K/T}$ seemed too big. We propose a procedure that we call iterative improvements, based on ideas by Bottou *et al.* [98] that achieves that and manages to tremendously improve the convergence rate of replay methodologies by designing the recommender system step by step. Furthermore this chapter contains a broad set of experiments that exhibits the good empirical performance of this method in various settings. The improvement compared to the state of the art is spectacular in all of them. This chapter also highlights the roles of the various parameters of the methods in the various possible settings. Finally, as mentioned in the abstract, our new method comes along with a highly desirable property in the context of minimizing the risks entailed by putting online new recommendation algorithms. Indeed, bootstrapping allows to compute the empirical distribution of the CTR obtained by an algorithm. From this distribution, many quality estimators can be derived so as to acquire a plethora of additional information. Therefore with such a method controlling risk, which is all we are after when designing evaluation methods, is much easier. This chapter also analyzes theoretically the convergence of this distribution estimation. As amazing at it may seem, it converges in $O(\sqrt{\Gamma}/T)$ against $O(\sqrt{\Gamma/T})$ for the estimator of the mean.

5.2 Time acceleration in our simple setting

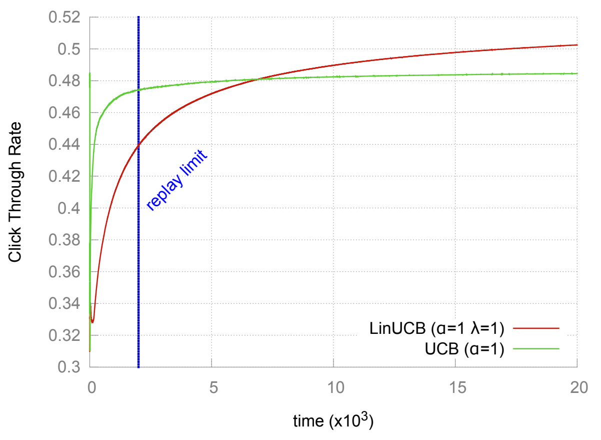
In this chapter we consider the simple problem of evaluating a contextual bandit algorithm A or policy π on a classical, static bandit game $\mathcal{B} = \{\mathcal{D}, T\}$. The support of $\mathcal{D}$ is composed of $\{1..K\}$ - the action set - and $\mathcal{X}$ - the context set (see section 1.4.2 for a thorough description of the problem). We assume that a dataset S was built by letting a randomly uniform policy play against $\mathcal{B}$. Therefore S contains T events.

The replay method, when used on S, will only evaluate an algorithm on $\frac{T}{K}$ events on average. For a policy this means that $\hat{g}_\pi^{(replay)}(S)$ will concentrate around g_π in $O\left(\sqrt{\frac{K}{T}}\right)$. For a learning algorithm, this will be more problematic unless T is infinite as it will have much less time to learn and thus it will be underestimated. More formally, $\hat{g}_\pi^{(replay)}(S)$ is in fact an estimator of $g_A(\frac{T}{K})$ and not $g_A(T)$. This is much more problematic that it may seems at first glance.

We already explained the nature of the problem but for clarity, let us take an example in



(a) T=100,000: The algorithms are underestimated, especially LinUCB. Yet replay manages to identify LinUCB as the best algorithm.



(b) T=20,000: When the horizon is smaller, the underestimation is much worse and even slightly affects UCB. Here, replay does not manage to exhibit that LinUCB is better than UCB. The bias is huge and the evaluation misleading.

Figure 5.1: CTR over time of LinUCB and UCB when evaluated against the actual synthetic model. When used on a dataset of size T, replay is an unbiased estimator of only $\frac{T}{K}$ interactions with the model. Therefore the blue line delimits what replay would actually see. Note the obvious issue that time acceleration raises, especially when the horizon is small.

the precise setting we consider. Moreover in this chapter, in order to empirically evaluate our new evaluation methods compared to the replay method, we will need synthetic data. Indeed, a notion of ground truth is needed to measure the errors of estimation. We will use the model that we already used in this thesis work and that is described in section 2.6.5. Note that we will take a number of features $F = 15$ (plus a constant feature equal to one which is needed by some algorithms), a number of items $K = 10$ and set the parameter q_{MAX} to 3. Let us remind that q_{MAX} is the maximum number of informative features of a given item. As a consequence it reflects the level of possible personalization. Apart from that, the synthetic model is just based on the intuitive idea that some items are very popular in general whereas some other are only of (high) interest to a small community of users. It is also very noisy to be closer to real life applications and prevent an algorithm from learning all the model in a few interactions. The reader is referred to section 2.6.5 for more details. We will consider the evaluation of two distinct algorithms:

1. A non contextual algorithm: UCB [13] with the exploration parameter α set to 1. Such an algorithm can only hope to achieve a ”most popular item” policy, which is not considered as personalized recommendation.

2. A contextual algorithm: LinUCB [7] with $\alpha = 1$ and λ, the regression parameter set to 1 as well. Since the model is linear, we can assume that such an algorithm would eventually be able to capture its full complexity.

Both are described and commented in section 1.4.3 and are the most representative algorithms of their respective categories. Their implementations are given in algorithm 1 and algorithm 5 respectively. Note that *all* the synthetic data we will refer to in this chapter is generated by the model we just discussed.

Our first experiment is simply an evaluation of the two aforementioned algorithms against the synthetic model. The experiment is meant to see graphically what replay actually evaluates and how this can be problematic. The results are shown and commented on figure 5.1.

To conclude on this experiment, note that if we consider a bandit game with fixed horizon T, in most applications it would be possible to acquire more data than T events and somehow solve the problem. This is only if we consider that two instances of the game are similar. If they are not, the problem is more complicated. Yet this is not the main concern. Indeed when considering a dynamic bandit game such as news recommendation, that can be for simplicity assumed to be a sequence of static games of horizon T, time acceleration can never be solved with more data. Indeed even with an infinite dataset, a learning algorithm would only have $\frac{T}{K}$ interactions with each game in the sequence instead of T, giving the impression that time is accelerated when the dataset is replayed. This would severely bias the evaluation especially if the span for which the static assumption holds is relatively small, hence the need for a new method.

Note that some changes can be smooth such as the CTR decay of some items [114]. In this case, time acceleration is even a worse issue as by replaying data, it really feels like the decay has a steeper slope.

5.3 The approach

5.3.1 How to use bootstrapping?

Since its introduction in 1979, bootstrapping has become a mature and widely used tool. A thorough introduction, which is now the reference book in the field, was written by Efron and Tibshirani in 1993 [109]. Note that this book also describes many variants of the bootstrap procedure. Later in this chapter we will describe one of them that is of interest for our problem. In this work we will only scratch the surface of what bootstrapping is and can accomplish. In statistics, the bootstrap is helpful in many situations. Ader *et al.* [115] make a series of recommendation as to when it should be used. The most common of them are the following.

- When the statistic that we are interested in has a complicated (or unknown) theoretical distribution, bootstrapping, which is distribution independent can help assessing its properties by using the empirical estimate of the density function.

- When we are dealing with a small sample that makes classical inference difficult, bootstrapping can help telling whether or not the inferred properties are representative of the population.

These two situations more or less fit ours. We do not want to make assumptions on the data. Therefore classical inference is difficult. Moreover our problem of reducing time acceleration is more or less about doing more with the same small amount of data. Yet if having a sense of the variability of our estimate would be reason enough to bootstrap the replay method (and it is), we are after increased accuracy, that classical bootstrapping does not enable in theory.

Bootstrapping has also been used successfully in Machine Learning in the past. We can first mention an example that is very similar to what is done in statistics. When learning a regression on a dataset, it is a relatively common practice to use bootstrapping in order to get an interval of confidence on the parameter of the regressor [116]. It can also be used to estimate the error rate of the predictor. In this case the bootstrap estimator is closely related to the estimator outputted by k-fold cross validation. Efron *et al.* [117] studied the difference and strength of several approaches based on either bootstrapping or cross validation. In brief, cross validation tends to have little bias and more variance whereas bootstrapping tends to have less variance but more bias due to different sampling methods. Note that the *.632+* rule (0.632 is the average proportion of distinct records in a bootstrap resample) is a modified and more recent bootstrapping technique that outperforms cross validation empirically via mainly a smarter sampling process [118].

The most well known application of bootstrapping in Machine Learning is not about estimating properties of an estimate such as the sample mean, a regression parameter, an error rate *etc.* *Bagging* [119] that stands for *Bootstrap AGGregatING* (and also known as bootstrap averaging) is a method to improve the prediction accuracy of predictors trained on a dataset. The basic process consists in training multiple predictors on bootstrap resamples and building a super predictor that averages all of them. It is generally applied to decision trees, in which case the average is in fact made by a voting process. In many applications, bagging exhibits much better results than the training of only one predictor. Note that people tried to theoretically justify these results [120] ever since but they are still not fully understood. Note that bagging which is tightly related to bootstrapping is very often confused with Boosting. Note that boosting is about combining very simple predictors (weak learners) trained on the same dataset so as to create a strong learner by aggregating them. This is a bias reduction technique that does not involve bootstrap resamples. Bagging aggregates the results of strong learners so as to reduce their variance. Anyway, bagging is a way to use bootstrapping to achieve a better accuracy without acquiring more data which is much closer to what we want to do. It is tempting to believe that since the replay method rejects a big fraction of the data, including randomness via bootstrapping and then averaging the estimated CTR would lead to a better accuracy. Yet we would not get the improvement we seek ($O\left(\sqrt{\frac{K}{T}}\right)$ to $O\left(\sqrt{\frac{\Gamma}{T}}\right)$ or even $O\left(\sqrt{\frac{1}{T}}\right)$) and it would be of no help to evaluate learning algorithms more fairly.

Finally let us mention a last work that inspired the method we present here. In its early days, bootstrapping has been used to improve situations in which too little data was available. Using bootstrap-like methods on small datasets is sometimes the only way to acquire knowledge on estimators that are computed on them since classical inference systematically fails with too little data. Note that bagging is another more recent example in which bootstrapping helps improving a situation without acquiring new data but in a different way. At the time, nobody had too much data. Yet nowadays the amount of data available increases exponentially and people often face situations in which the limiting factor is no longer the quantity of data but the computational resources. When computing a regression, an estimator, or any other statistical object on a huge dataset is difficult, bootstrapping this object to know it properties is unthinkable. Very recently Kleiner *et al.* [113] proposed an approach called the Bag of Little Bootstraps (BLB) to tackle this problem. BLB is a simple bootstrapping technique. The only difference is how the bootstrap resamples are built. Indeed to build one resample, BLB takes a sub-samples of size m (without replacement) from the big dataset of size T with $m << T$ (*e.g.* $m = T^{0.6}$ in their experiments). Then the small resample is expanded by sampling T times with replacement and the estimator is computed on this expanded sample. Note that as most estimators, regressors, classifiers *etc.* can work with weighted representations of the data, this process tremendously reduces the time needed to deal with one resample. Their authors prove that BLB achieves a quality of estimation of the density of a statistic that is of the same order as the one of the regular bootstrap as long as the number of resamples B tends to infinity. BLB can be used both as a classical bootstrapping approach, that is to estimate properties of a statistic or as a bagging procedure to construct, for instance, an aggregated regressor.

5.3.2 Bootstrapped replay on expanded data

Before describing our approach, let us remind the standard bootstrap approach and introduce a few notations. The general idea of bootstrapping is to build B datasets $S_{b\in\{1,\dots B\}}$ each of size T out of a single dataset S. A dataset S_b, also called a bootstrap resample, is simply built by drawing T records randomly with replacement from S. Note that in each of the resulting datasets S_b, there are records that occur more than once, and other ones that do not appear at all (and 0.632 distinct records on average). Then B bootstrap estimates $\hat{\theta}(S_b)$

or $\hat{\theta}^{(b)}$ for short, estimating a quantity θ that depends on $\mathcal{D}$ are computed on all the S_b using an estimator $\hat{\theta}$. From there, several things are usually done in the literature. Bagging (**B**ootstrap **AGG**regat**ING**) consists in averaging the $\hat{\theta}_b$ to get a "bagged" estimate. This yields a bagged estimator $\theta^{(\hat{bagged})}$. In some Machine Learning applications, it was shown that computing $\theta^{(\hat{bagged})}(S)$ instead of simply $\theta(\hat{S})$ increases the estimation accuracy. Yet this is not really justified in classical bootstrapping theory for this is not the original purpose. Note as a simple remark that in this work, we justify accuracy gain via bootstrapping in our special case of replay evaluation.

Pure bootstrapping consists in taking the set of bootstrap estimates all together so as to obtain an empirical distribution (basically a histogram). The resulting empirical distribution is an estimator of the distribution of the estimates $\hat{\theta}(T, \mathcal{D})$ produced by $\hat{\theta}$ when computed on T samples drawn from $\mathcal{D}$. Therefore note that in fact, the bootstrap resamples are a way to fake sampling from $\mathcal{D}$ so as to be able to compute several realizations of $\hat{\theta}(T, \mathcal{D})$ with only one dataset S of size T. These realizations are an estimator of the distribution of $\hat{\theta}(T, \mathcal{D})$. It is this bootstrap distribution estimator that is of interest to us so as to figure out whether our estimations of θ are reliable or not.

From a theoretical point of view and under mild assumptions, the estimator of the density of the distribution of $\hat{\theta}(T, \mathcal{D})$ converges with no bias at a speed in $O(1/T)$, which is better than the previous analytic methods that are in $O(1/\sqrt{T})$ [14]. Note however that if the density converges in $O(1/T)$, the mean still converges in $O(1/\sqrt{T})$ which does not improve in theory the accuracy of estimation of θ. Nonetheless bootstrapping replay to get an estimation of the distribution of $\hat{g}_A^{(replay)}(T)$ would only solve part of our problem. Indeed the empirical distribution would probably allow to acquire information so as to minimize the risk of putting new algorithms online. Yet we would actually estimate the distribution of $\hat{g}_A^{(world)}(\frac{T}{K}, \mathcal{D})$. In other words, we would not solve time acceleration and do not achieve any theoretical improvement in terms of evaluation accuracy. From this point on, we will use a few additional notations. As we estimate distributions, we will denote by $CTR_A(T)$ the distribution of the CTR of A after T interaction with $\mathcal{D}$ (we drop $\mathcal{D}$ to lighten notations). $\widehat{CTR}_A(T)$ is an estimator of this distribution. Since it is classical in bootstrapping theory [14, 113], we will use $\xi(.)$ to denote an estimator quality assessment. In fact we will show that $\xi\left(\widehat{CTR}(T)\right)$ (when estimated via the method we describe below) converges with no bias to $\xi\left(CTR(T)\right)$. The reader should really see $\xi(.)$ as a technicality that is used to make the theorems accurate and bring more generality to the results. Indeed, in the literature ξ can be many things: a bias, a standard error, a confidence interval *etc.* $\xi(.)$ can also be any point of the density function of the distribution [14, 121].

As surprising as it may sound, our method is inspired by BLB. This is paradoxical as the goal of BLB is to be able to use more data whereas ours is to require less data to work correctly. Yet they expand their resamples in the process which is exactly what we need to do with our dataset to be able to evaluate algorithms with more accuracy. In a nutshell instead of subsampling (to reduce the dataset to m records) and then expanding the sample by sampling T records with replacement, we will do the opposite. More precisely we will first expand the dataset S via sampling $K.T$ records with replacement and then evaluate a policy on this expanded dataset which is a sort of procedure in which T records in expectation are subsampled. The main idea behind that is that the evaluation of a learning algorithm is biased because the number of simulated interactions is insufficient (see figure 5.1b). The evaluation only uses one out of K records of the dataset but K times more are in fact available. A bandit algorithm has a shifting behavior: to simplify, first it mainly explores, resulting in a wide choice of action and then its behavior smoothly shifts towards mainly exploiting resulting in a very stable behavior at some point. Therefore we hope that the algorithm will take advantage of the expanded dataset in two ways. First since its behavior evolves, the algorithm will make different choices if a same context is presented at different times, therefore using more than $\frac{T}{K}$ distinct records in average. Also, when the behavior is more stable, we simply aim at providing more time to exploit and get high reward so as to lower the cost of exploration to its cost in the real world where time is not accelerated. In other words, we aim at slowing time back to its real pace by taking advantage of the "friendly" behavior of a typical learning algorithms. A last idea of the approach is inspired from Bagging. Since there is more randomness involved in each bootstrap resample than the simply the one induced by the resampling process, a bagged estimate should be more accurate than a simple estimate. Note that these justifications are only intuitive. We will justify this method by other more concrete means all along this chapter, for both static policies and learning algorithms. Besides the obvious interest regarding bandit evaluation, a key contribution is that in the end, we manage to demonstrate formally why bagging helps and to quantify that accuracy gain in the specific context that is ours.

Let us describe the approach with more accuracy, using more precise notations. From a dataset of size T with a choice between K possible actions at each step, we generate B datasets of size $K.T$ by sampling with replacement, following the non-parametric bootstrap procedure. In other words, we sample with replacement from S. Then for each dataset S_b we use the classical replay method to compute an estimate $\hat{g}_A^{(b)}(T) \overset{\text{def}}{=} \hat{g}_A(S_b, T)$. Therefore A is evaluated on T records on average. In our theoretical setting, when considering static policies (that are stochastic), a coarse way to see this step is as a subsampling step that allows to return in the classical bootstrap setting. One significant difference though is that we end up with bootstrap resamples of size $T^{(b)}$. If we do have $\mathbb{E}\, T^{(b)} = T$, they are different in general. Our analysis will however show that this is less problematic than it seems, for we have a concentration result for $T^{(b)}$. Note also that it would not work the same way for a purely deterministic and static policy. Estimating the distribution of the estimator would work anyway (yet with weaker guarantees). However for obvious reasons such a policy would not take advantage of the data expansion by making always the same choices and thus using always the same records. Nonetheless we will exhibit how this statement can be slightly soften. A technique called smoothed bootstrap can indeed be considered in this case for better results. Theoretical justifications are provided in section 5.9 and probing experiments in section 5.10.1 but the content that comes in between is important for a full understanding.

Finally $\hat{g}_A(T)$, the bagged estimate, is given by averaging the $\hat{g_A}^{(b)}(T)$. Together, the $\hat{g_A}^{(b)}(T)$ are also an estimation of $CTR_A(T)$, the distribution of the CTR of A after T interactions with $\mathcal{D}$ on which we can compute our estimator quality assessment ξ. More formally, the bootstrap estimator of the cumulative density of $CTR_A(T)$ is estimated as follows:

$$\widehat{CTR}_A(T)(x) = \frac{1}{B}\sum_{b=1}^{B} I\left(\sqrt{T}\left[\frac{\hat{g}_A^{(b)}(T) - \hat{g}_A(T)}{\hat{\sigma}_A(T)}\right] \leq x\right) ,$$

where $\hat{\sigma}_A(T)$ is the empirical standard deviation obtained when computing the bootstrap estimates $\hat{g_A}^{(b)}(T)$. Note that ξ can stand for a point of that density but can also be something else such as a confidence interval. Note also that using $\widehat{CTR}_A(T)$ for both the cumulative density and the distribution is not perfectly rigorous but since we only use it here, it avoids introducing more notations.

The complete procedure, called Bootstrapped Replay on Expanded Data (BRED), is implemented in algorithm 12.

To complete the BRED procedure, one last practical detail is necessary. Each record of the original dataset S is contained K times in expectation in each expanded dataset S_b. Therefore a learning algorithm may tend to overfit which would bias the estimator positively. Considering that we want to minimize the risk of putting a new algorithm online, this is something we want to avoid at all cost. Therefore to prevent this from happening, we introduce a small amount of Gaussian noise on the contexts. This technique is known as Jittering and is well studied in the neural network field [122,123]. The goal is the same, that is avoiding overfitting. In practice it is however slightly different as neural networks are generally not learning online. They are actually scanning through datasets several times successively until convergence, each record being used several times during training. Jittering is thus seen as a regularization technique which is more or less what we want anyway. In bootstrapping theory this technique is known as the smoothed bootstrap and was introduced by Silverman and Young [15]. We will discuss this further in section 5.9 so as to fully justify this technique in our context. Yet note that it is not necessary in our analysis based on stationary policies as the previous ones, that is in chapter 3 and Li *et al.* [11]. Finally, as explained in introduction, time acceleration is particularly problematic when the environment evolves. As exposed here and in algorithm 12, BRED cannot handle such environments. Yet, in section 5.11 (and in particular in section 5.11.1, we show that a mere tweak can be used to adapt BRED to dynamic environment.

Algorithm 12 *Bootstrapped Replay on Expanded Data* *BRED.*
We sketch this algorithm so that it looks very much like the replay method in Alg. 10. The same remark may be done regarding the histories $h^{(b)}$.

Input

- A (contextual) bandit algorithm A
- A set $\mathcal{S}$ of T triplets (x, a, r)
- An integer B

Output: An estimate of g_A

```
$h^{(b)} \leftarrow \emptyset, \forall b \in \{1..B\}$  /*empty history*/
$\widehat{G}_A^{(b)} \leftarrow 0, \forall b \in \{1..B\}$
$T^{(b)} \leftarrow 0, \forall b \in \{1..B\}$
/* Bootstrap loop*/
for $b \in \{1..B\}$ do
    /* estimation of $CTR_A^{(b)}(T)$ */
    for $i \in \{1..T * K\}$ do
        Sample with replacement an element $(x, a, r)$ of $S$
        $x \leftarrow$ JITTER$(x)$  /*optional*/
        $\pi \leftarrow A(h^{(b)})$
        if $\pi(x) = a$ then
            add $(x, a, r)$ to $h^{(b)}$
            $\widehat{G}_A^{(b)} \leftarrow \widehat{G}_A^{(b)} + r$
            $T^{(b)} \leftarrow T^{(b)} + 1$
        else
            /* Do nothing. */
        end if
    end for
end for
return $\frac{1}{B} \sum_{b=1}^{B} \frac{\widehat{G}_A^{(b)}}{T^{(b)}}$        OR        return $\frac{\sum_{b=1}^{B} \widehat{G}_A^{(b)}}{\sum_{b=1}^{B} T^{(b)}}$
/* The first one is the bagged estimate as it is usually defined in bootstrapping. The second comes with the nice property of having a bias in $O(a^{TB})$ instead of $O(a^T)$, with $a = \frac{K-1}{K}$ which is slightly less than one. This may be significant if $T$ is small.*/
```

5.4 A theoretical analysis based on a state-of-the-art method

In this section, we make a theoretical analysis of our evaluation method BRED. The core loop in BRED is a bootstrap loop; henceforth, to complete this analysis, we first restate the theorem 12 which is a standard result of the bootstrap asymptotic analysis [113]. Notice a small detail: each bootstrap step estimates a realization of $CTR_A(T)$. The number of evaluations - which is also the number of non rejects - is a random variable denoted $T^{(b)}$. Note that here, we model the analysis on the work of Kleiner *et al.* [113] and as a matter of fact, our procedure is more or less equivalent so their results apply to our case. Moreover keep in mind that these results are asymptotic but the issue will be tackled later.

Theorem 12. *Suppose that:*

- *A is a recommendation algorithm which generates a fixed policy over time (this hypothesis can be weakened as discussed in remark 2),*
- *K items may be recommended at each time step,*
- *$\xi\left(CTR_A(T)\right)$ admits an expansion as an asymptotic series (called Edgeworth expansion [124])*

  $$\xi\left(CTR_A(T)\right) = z + \frac{p_1}{\sqrt{T}} + \ldots + \frac{p_\alpha}{T^{\alpha/2}} + o\left(\frac{1}{T^{\alpha/2}}\right)$$

  where z is a constant independent of the distribution $\mathcal{D}$, and the p_i are polynomials in the moments of $CTR_A(T)$ under $\mathcal{D}$ (this hypothesis is discussed and explained in remark 1),
- *the empirical estimator $\xi\left(\widehat{CTR}_A(T)\right)$ admits a similar expansion:*

  $$z + \frac{\widehat{p}_1}{\sqrt{T}} + \ldots + \frac{\widehat{p}_\alpha}{T^{\alpha/2}} + O\left(\frac{1}{T^{\alpha/2}}\right). \tag{5.1}$$

- *$\frac{K}{T} \to 0$ as $T \to inf$. In other words, T is much bigger than K.*

Then, for $T \leq T^{(b)} \times K$ and assuming finite first and second moments of $\widehat{CTR}_A(T)$, with high probability:

$$\begin{gathered}\left|\xi\left(CTR_A(T)\right) - \xi\left(\widehat{C}TR_A(T)\right)\right| = \\ O\left(\frac{Var(\widehat{p}_\alpha^{(1)} - p_\alpha|D_T)}{\sqrt{T \cdot B}}\right) + O\left(\frac{1}{T}\right) + O\left(\frac{1}{T\sqrt{T}}\right)\ ,\end{gathered} \tag{5.2}$$

where $\mathcal{D}_T$ is the resampled distribution of $\mathcal{D}$ using T realizations.

Proof. It is actually a straightforward adaptation of the proof of theorem 3 of Kleiner *et al.* [113].

Also note that this theorem is a reformulation of the bootstrap main convergence result as introduced by Efron [14]. The reader unfamiliar with bootstrapping may find that fast convergence rate to be somehow unsettling but it is classical in bootstrapping theory and is the main reason why it is used in practice by statisticians. □

Now, we use theorem 12 to bound the error made by BRED in the theorem 13.

Theorem 13. *Let us assume that*

$$\xi\left(CTR_A(T)\right) = z + \frac{p_1}{\sqrt{T}} + \ldots + \frac{p_\alpha}{T^{\alpha/2}} + o\left(\frac{1}{T^{\alpha/2}}\right) .$$

Then for a algorithm A producing a fixed policy over time, BRED applied on a dataset of size T evaluates the expectation of the CTR_A with no bias and with high probability for B and T large enough:

$$\left|\xi\left(CTR_A(T)\right) - \xi\left(\widehat{C}TR_A(T)\right)\right| = O\left(\frac{1}{T}\right) .$$

This means that the convergence of the estimator of $\xi\left(CTR_A(T)\right)$ is much faster than the convergence of the estimator of $g_A(T)$ (which is in $O(1/\sqrt{T})$. This will allow a nice control of the risk that $\hat{g}_A(T)$ may be badly evaluated.

The sketch of the proof of theorem 13 is the following: first we prove that the replay strategy is able to estimate the moments of the distribution of CTR_A fast enough with respect to T. The second step consists in using classical results from bootstrap theory to guarantee the unbiased convergence of the aggregation $\widehat{CTR}_A(T)$ to the true distribution with an $O(\frac{1}{T})$ speed. The rational behind this is that the gap introduced by the subsampling will be of the order of $O(\frac{1}{\sqrt{TB}})$. The full proof can be found in appendix G.1

After this analysis, we make three remarks about the assumptions that were needed to establish the theorems.

Remark 1 The key point of the theorems is the existence of an asymptotic expansion of $CTR_A(T)$ and $\widehat{CTR}_A(T)$ in polynomials of $1/\sqrt{T}$. This is a natural hypothesis for $CTR_A(T)$ because the CTR is an average of bounded variables (probabilities of click). Making the same one on $\widehat{CTR}_A(S_b)$ may seem a bit optimistic. Indeed, the corresponding estimator uses $T^{(b)}$ records. Theorem 2 shows that in spite of this issue, we keep consistency and the order of convergence for the mean g_π in $O\left(\frac{1}{\sqrt{T}}\right)$. For the bootstrapped estimation of the *distribution*, we need the Edgeworth expansion to be true which is not proved. Nevertheless we can argue that the assumption is very reasonable. Indeed for any stationary policy, the mean is going to concentrate according to the central limit theorem (CLT). Furthermore this hypothesis, that we shall refer to as the bootstrap assumption, is omnipresent in bootstrap theory and for a wide variety of estimators [14]. For instance it is justified in econometrics by the fact that all the common estimators respect it [121]. Let us for instance quote Horowitz [121], who made many contributions in areas that connect econometrics and bootstrapping:

> *Many important econometric estimators, including maximum-likelihood and generalized-method-of-moments estimators, are either functions of sample moments or can be approximated by functions of sample moments with an approximation error that approaches zero very rapidly as the sample size increases. Thus, the theory outlined in this section applies to a wide variety of estimators that are important in applications.* (Horowitz [121], 2001)

Note that we solve this assumption issue in what follows.

Remark 2 Let us consider algorithms that produce a policy that changes over time (a learning algorithm in particular). After a sufficient amount of recommendations, a reasonable enough algorithm will produce a policy that will not change any longer (if the world is stationary). Thus the CLT will apply and we will observe a convergence of $\hat{g}_A(T)$ to its limit in $1/\sqrt{T}$ when played in the real world, making our analysis applicable. Nevertheless nothing holds true here when the algorithm is actually learning. This is due to the fact that the Chernoff bound no longer applies as the steps are not independent. However the behavior of classical learning algorithms are smooth, especially when randomized (see Auer *et al.* [125] for an example of a randomized version of UCB). Li *et al.* [11] and ourselves (in section 3.5.5) argue that in this case convergence bounds may be derived for replay (which then could be applied to BRED) at the cost of a much more complicated analysis including smoothness assumptions. For non reasonable algorithms and thus in the general case, no guarantees can be provided. By the way note that a very intuitive way to justify Jittering is to consider that it helps the Chernoff bound being "truer" in the case of a learning algorithm. We will also see in what follows that Jittering can also help to achieve higher accuracy even in the case of static and deterministic policies (see section 5.10.1).

Remark 3 An interesting detail that does not appear very clearly in the theorems due to light notations is the following. If the assumptions are met, the distribution estimator and in particular $xi\left(\widehat{CTR}_\pi\right)$ converges with no bias at a speed in $O\left(\frac{1}{T}\right)$. Although it is not crucial, it is very interesting to understand that it does not exactly converge in spirit toward what we would expect, that is xi computed on the distribution of the CTR of π after T interactions with the world ($xi\left(\widehat{CTR}_\pi(\mathcal{D}, T)\right)$). In fact it converges toward xi computed on the distribution of the CTR of π after being replayed on a dataset of size $K.T$ ($xi\left(\widehat{CTR}_\pi^{(replay)}(S_{KT})\right)$). Yet theorem 13 shows that this estimation concentrates the same way as a real world evaluation using the concentration of $T^{(b)}$. Let us also recall that replay has an exponentially low bias. Furthermore we showed in chapter 3 that replay (resp. replay*) introduces an amount of variance proportional to $\frac{1}{T^2}$ (resp. $\frac{1}{T}$) compared to a real evaluation because the number of simulated interactions is a random variable (denoted by $\sum V_t$ in this work, see section 3.5.6 for more detail). The order of these differences between the first two moments perfectly fits the results presented in theorem 13.

Many more remarks about this analysis could be done but in our opinion they are much easier to understand when backed by experiments and figures. Therefore they are interleaved with the experiments in the upcoming sections.

5.5 Toward a finer analysis: how does BRED work in practice?

5.5.1 Discussion of the previous analysis

The analysis we just made is completely heuristic. Indeed, although it is based on classical results, two key assumptions are just admitted: the existence of an Edgeworth expansion in $\frac{1}{\sqrt{T}}$ for $CTR_\pi(T)$ and the existence of a similar expansion for $\widehat{CTR}_\pi(T)$, which is its bootstrap estimation. This is exactly what was done by Kleiner *et al.* for the BLB [113] so we did it as well in the exact same situation. As discussed in remark 1, the first assumption is natural in a real world evaluation. It seems possible to verify that the second one, when $T^{(b)}$ is random, is met in our case. Yet this remains to be proved.

Furthermore this analysis has the major drawback of being asymptotic since K/T has to tend to 0. This cannot be avoided with the bootstrap [14]. Yet since our result on the convergence of the bagged estimate depends on that result, it is asymptotic as well which could be avoided. Here we consider dynamic applications, so only small portions of datasets can be considered static and bootstrapped. This makes the relevance of this result quite questionable. Moreover, most of the time asymptotic results still give intuition as to what happens with a finite amount of data. This is not the case here which is very unfortunate. In our opinion, the lack of intuition is the major drawback of the analysis. Also, after more thought, the convergence result in $O\left(\frac{1}{\sqrt{T}}\right)$ for the mean and $O\left(\frac{1}{T}\right)$ for the density, whilst S only contains T **partially labeled** records seems really unrealistic for finite datasets. In fact in would enable the use of this method for datasets with millions of items which seems unlikely (this statement is limited by the fact that K/T has to tend to 0). This is due to the fact that with Kleiner *et al.*'s analysis, a multiplicative factor such as K, that could appear here is just washed out. Since the big O is used it does not make the result incorrect but only much less accurate and almost useless for our purpose, that is actually reducing the dependence in K. Indeed, one has to remember that it is very important in our context to be able to tell if T is big enough given the value of K. As a conclusion, although the similarity of our problem with another previously studied one was particularly encouraging, it really turns out to be a false promise, especially because the blind assumptions that are made make difficult to use this analysis to understand practical cases. Nonetheless let us not minimize the range of this result either. It can be considered as a proof of concept: bootstrapping replay works. By working we mean that the estimation of the density of the estimator by expanded bootstrap is asymptotically unbiased, consistent and that it concentrates at a rate of the best order we could expect. As a matter of fact, the significance of this contribution was acknowledged by the community. Indeed an article presenting the time acceleration issue, these two theorems and an empirical study was accepted at ICML-2014 [126]. The only problem of this interesting proof of concept is that the concentration is not determined accurately enough. Consequently it prevents anyone from deriving intuition from this result. As such it does not answer, in theory, our original concern which was to increase replay's accuracy. Note that we specify "in theory" because we will see that doing better than replay is possible in all the practical situations we will consider in this work.

In what follows we propose to study how we could give theoretical basis to justify that. To do so we need to get a result that fits our requirements, that is a result that exhibits how it depends on K and which is just more precise in general. Also we will show how one of the two bootstrap assumptions that are made by Kleiner *et al.* [113] is unnecessary and support the claim that in our problem of data replay, the remaining one can be met. Finally, the previous work is more or less asymptotic (K/T goes to zero). We need to get rid of this for we consider big datasets but acquired from dynamic applications. Indeed only small portions of such datasets can be considered stationary and thus bootstrapped.

In this section we lay the first stone of this new analysis. This first stone is intuition. We first verify in our simple theoretical setting that BRED does yield increased accuracy. We also investigate the conditions of such a gain and pay attention to the role of parameters such as B in this context. Then we set our focus to density estimation and simply check that BRED estimates what is expected in spite of this slight uncertainty we have on the second Edgeworth expansion. In what follows, we will use the intuition acquired in this section to show formally and precisely that BRED allows very accurate density estimations of several estimators. This will be done without relying on unproved assumptions, as likely as they may be. We will also quantify with precision the gain of accuracy in the estimation of g_π. To do so we will go back to the very basics of bootstrapping theory and build from there to solve our problem.

5.5.2 Accuracy gain

The purpose of this first subsection is two-fold. First we really want to clarify how BRED works and the different things it can output. Second we want to exhibit naively the gain in terms of pure accuracy of evaluation compared to replay, without paying attention to the bootstrapped density estimation for now. To do all this, we evaluate the random policy. We also briefly study the effect of the bootstrap parameter B and try a variant of the sampling process.

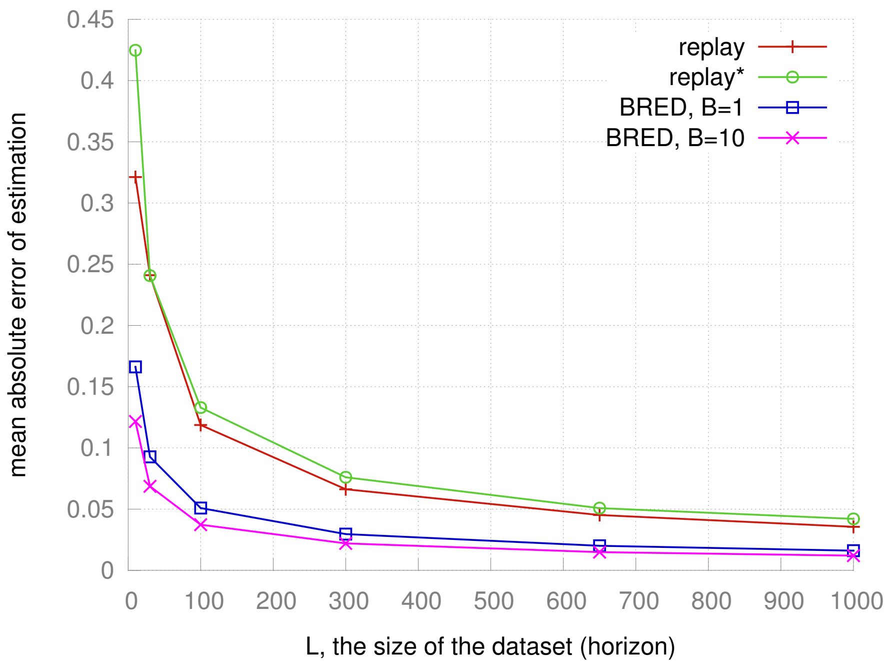
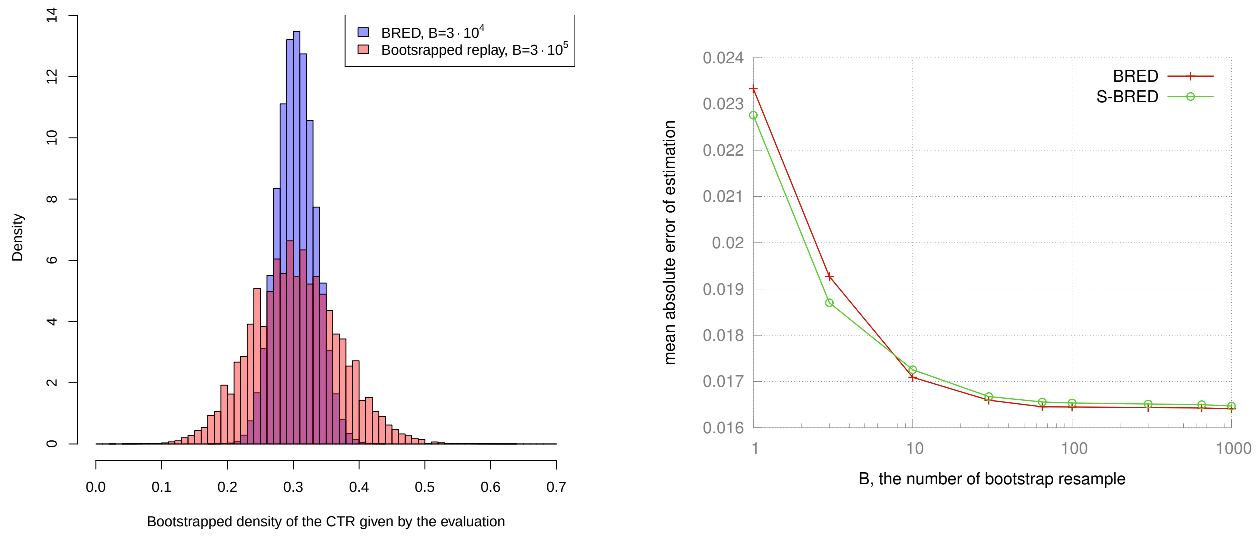


Figure 5.2: Mean absolute error (MAE) of the estimation of the CTR of the uniformly random policy on synthetic data. Lower is better. Note that in average, BRED (B=10) has a MAE that is 3.2 times lower the MAE of replay and 3.6 times lower than replay*. BRED also performs better when B=10 instead of 1. The variation of the MAE as a function of B is studied further on figure 5.3b.

Here we first are interested in the raw performance of the three methods we have at hand: replay, replay* and BRED. The results, that are computed for various horizon sizes T are displayed and commented on figure 5.2. They look very much like what was expected, that is BRED is better than the simple replay methods with the same amount of data. Note also that if the errors of all the methods shrink when the horizon increases, the ratio between the errors of replay and BRED remains constant (between 3 and 4). Note also that no accuracy improvement would be possible for a non-stochastic policy. Indeed, using replay once would teach us everything we have to know. Scanning the data more times would only lead to the same choices and thus the same evaluation. Note also however that it does not prevent us from bootstrapping to find out the CTR distribution. Figure 5.2 also displays the improvement brought by increasing B, the bootstrap parameter. This phenomenon is more precisely studied on figure 5.3b. We studied the impact of B on the results for BRED and a faintly modified version: S-BRED. Instead of performing B bootstrap resamples of size KT, S-BRED gets the B resamples as follows: it expands the dataset by simply appending $K-1$ copies of S to itself. S-BRED then applies a random permutation to all the S_b (this is called shuffling, hence the S in front of the acronym). Note however that when evaluating a stochastic static policy, this in not necessary. This is only needed to evaluate learning algorithms for which the order of the records matters. This is also needed if a stochastic policy is approximated by a sequential policy, *e.g.* the uniformly random policy can be transformed into a sequential and deterministic policy that plays all the actions in a predefined order and starts over. This is sometimes done to save the resources used by the random sampling. Therefore notice a small detail. In this very special case, BRED can be considered as improving the evaluation of a deterministic policy. Yet if we consider such a policy to be deterministic, it can also be considered as non-stationary for it makes choices with probability one that are different from one step to another. The results of the experiments are commented on figure 5.3b.

Finally it is interesting to stress that for a static policy and *only* is it is static, using BRED



(a) Bootstrapped empirical density of two different estimators on a dataset of size $T = 500$. Their bagged estimators are strictly the same since they replay the same amount of data, sampled in the same way. Yet they do not estimate the same density: BRED($B = 3.10^4$) estimates the distribution of a CTR estimate after T interactions with the model whereas Bootstrapped Replay($B = 3.10^5$) estimates the distribution of a CTR estimate after $\frac{T}{K}$ interactions. This explains the same mean (since the policy is static) but different variance. Note that although B is greater when using replay, it yields a less accurate density estimation (noise on the histogram) because a more variate distribution is "longer" to estimate empirically (it requires greater value for B).

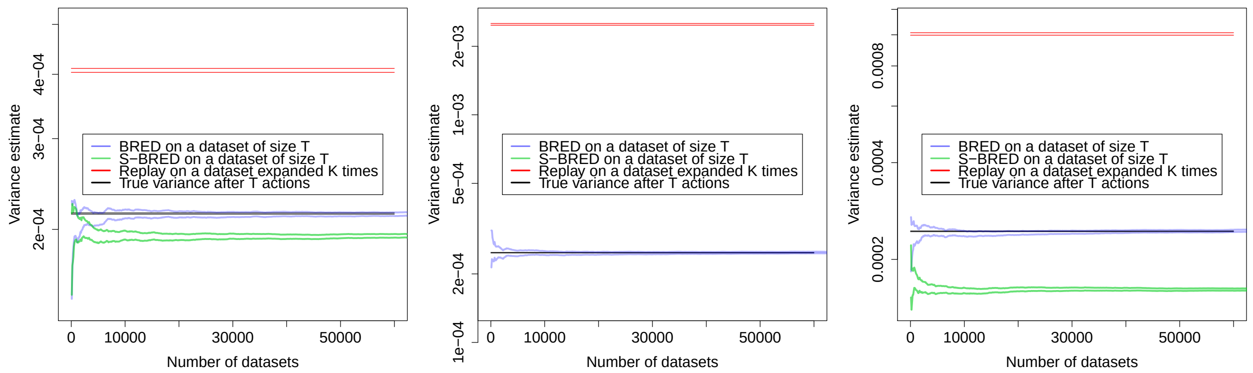
(b) Mean absolute error (MAE) of the CTR estimation using two different sampling techniques over the value of the bootstrap parameter B. Note that when B is small, that is when the computational resources are limited, S-BRED performs better. This makes sense since all the records are always present in the bootstrap resamples, which is not the case with regular bootstrapping (BRED). When B=10 or more, the difference vanishes. BRED is even systematically slightly better probably because of the greater variety in the resamples implied by bootstrap sampling. Nevertheless this difference is too small to really matter. Note that it is pretty useless to take $B > 30$ (if one is only interested in a bagged estimate) and that there is no significant improvement anymore after $B = 60$.

Figure 5.3: More on the evaluation of the uniformly random policy on synthetic data.

with a parameter B on S is very similar to bootstrapping the simple replay method with a bootstrap parameter $K.B$. This statement is entirely true if we only talk about the mean of the bagged estimate of the CTR. Indeed it is computed on average on $T.B$ records and therefore have the same convergence properties. Nevertheless this is only true for the mean and not for the other moments (the variance is clearly different). Therefore the density of the distribution that the bootstrapping procedure estimates is different. More formally, bootstrapping replay produces an estimator of $CTR_A(T/K)$ whereas BRED produces an estimator of $CTR_A(T)$. It also has different convergence properties. This is illustrated by figure 5.3a. Basically bootstrapped replay estimates the distribution of the CTR of the policy played $\frac{T}{K}$ times against the world, hence a more variate distribution. As such the density estimation converges in $O\left(\frac{K}{T}\right)$. BRED estimates the distribution of the CTR of the policy played T times against the world. We will see in what follows that the estimation converges faster than $O\left(\frac{K}{T}\right)$, without necessarily reaching $O\left(1/T\right)$ depending on the policy.

Remark It is important not to confuse the estimated variance of the distribution of the CTR estimator (width of the Gaussian on figure 5.3a) and the variance of the estimation of the density (noisy histogram to represent the distribution).

5.5.3 Bootstrap density estimation



(a) Random uniform policy. (b) Optimal policy (deterministic). (c) ε-optimal policy ($\varepsilon = 0.5$).

Figure 5.4: 95% confidence interval on the mean of the estimation of variance outputted by both BRED and S-BRED for three representative policies. In other words we generated many datasets. For each dataset BRED and S-BRED produce a histogram (estimated distribution) that has a variance. Here we plot a confidence interval on the mean of that variance as the number of datasets grows. Note that we could have chosen the mean of any other feature of that estimated distribution (confidence interval, point of the CDF *etc.*). We also plot two baseline: the variance of replay on a dataset expanded K times (red) and the variance of the average reward after T interactions with the real world (black). Notice that BRED consistently estimates what happens after T interactions with the real world whereas S-BRED does not estimate anything significant.

The accuracy gain was obvious before even exhibiting it. Now let us dive into the more tricky part, that is the density estimation with a finite quantity of data. We keep the same setting as before and evaluate three policies:

- The uniformly random policy.
- The optimal policy that given a context chooses the best possible action. It is thus deterministic.
- The ε-optimal policy that plays the optimal action with probability 0.5 and an action at random with probability 0.5.

Given that our first result is asymptotic and makes assumptions on replay that we did not verify, one can wonder if BRED even does what is expected of it, that is estimating $CTR_\pi(T)$, the distribution of the CTR of π after T interactions with the environment. BRED and S-BRED obviously have, in the worse case, the same exponentially low bias as replay as far as the mean of $CTR_\pi(T)$, g_π, is concerned. Yet it may seem surprising that BRED can estimate $CTR_\pi(T)$ completely for any π with only T partially labeled records. Because combined with the mean it allows to compute any confidence interval which is basically what we are interested in the most, let us focus on the variance estimation outputted by BRED. First we are not interested in the variance of this estimation but only in verifying whether or not BRED manages to estimate it in expectation. Indeed, one may suspect that in fact BRED does not estimate $CTR_\pi(T)$ but in fact the distribution of the replay estimator, computed on a dataset of size T expanded K times. Note that obviously, such a distribution is more variate than $CTR_\pi(T)$ for it is based on less data. To verify this, we simply generated many datasets and computed a 95% confidence interval on the mean of the variance estimated by BRED. Without making any comment on the rate of convergence and as we can see on figure 5.4, it is perfectly clear that BRED is estimating the variance of $CTR_\pi(T)$ and not the one of replay computed on an expanded dataset, which is different. Moreover, the results do not exhibit any bias. On the contrary, S-BRED clearly underestimates the variance. This makes sense since S-BRED does not simulate $\mathcal{D}$ but only runs replay B times on the very same records. Worse, when the policy is deterministic, S-BRED completely fails and outputs a one-bar histogram and thus a variance of zero.

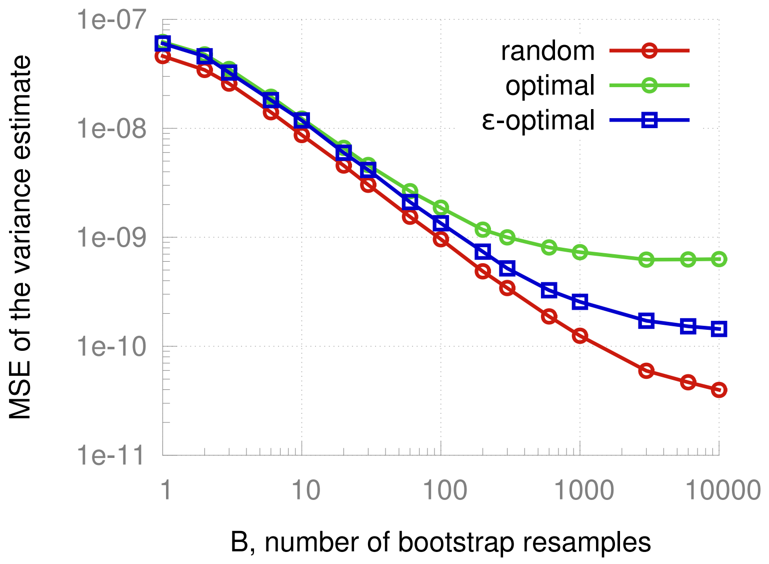


Figure 5.5: Mean Squared Error (MSE) of BRED on the variance of $CTR_\pi(T)$ for three representative policies. Note that B has to be much bigger than when we are only interested in the mean (fig. 5.3b). The results were average over 10,000 runs and T was set to 1000.

Just for the record we also got ourselves interested in how large B has to be in order to get a good estimation of the distribution $CTR_\pi(T)$. Indeed, if $B = 30$ seems enough to get the best estimation possible of the mean, it does not seem *a priori* enough for the entire distribution. To verify that, we just measured the Mean Squared Error (MSE) on the variance of $\widehat{CTR}_\pi(T)$ compared to the one of $CTR_\pi(T)$. The choice of the variance is arbitrary. It could have been a point of the CDF, a confidence interval *etc.*. The results match what could be expected and do not need much commenting. As it can be seen on figure 5.5, the only thing worth remembering is that in order to achieve good performance, we need to set B to a much higher value ($B = 10,000$ is not even always enough) than when we just want to estimate the mean. Another detail is quite interesting. The distribution corresponding to the random policy is better estimated than the one of ε-optimal which, in turn, is better estimated than the one of the deterministic, optimal policy. All this will be justified and studied in what follows. Yet this is quite intuitive: the more randomness there is in the evaluation of one policy, the more it takes advantage of bagging. This is true for the bagged estimate of g_π, although we have not verified it yet, and for its distribution which is proved by figure 5.5.

To conclude on this small set of experiments, BRED does what is expected of it and provides in practice an estimation of $CTR_\pi(T)$ for any policy. Note that in theory, there is still an exponentially low bias for the mean. Indeed, what BRED is estimating is the distribution of replay computed on $K.T$ *distinct* records. This means that the estimation of the variance should also experience a bias proportional to the squared inverse of the number of records, which is very small (see section 3.5.6 for replay's additional variance in compared to a real world evaluation). On the contrary, S-BRED should only be seen as a trick to get a more accurate estimation of g_π when computational resources are limited and B has to be kept small. As we will study it later, it has much more interest for learning algorithms. Before pushing further our understanding of what BRED does and, in particular, what is the exact precision of the bagged estimate, let us justify theoretically why it works.

5.6 Showing why the bootstrap works

In this section we lay the theoretical foundations of the formal justification of BRED. To do so we review how the bootstrap is justified in the statistical literature. From there we will see how our approach connects within this framework.

5.6.1 General points

At first glance, the bootstrap, which enables to estimate the distribution of an estimator computed on a sample of size T using only one sample of size T, may seem a little bit magical. Another seemingly magical element is the concentration in $O(1/T)$ of that distribution whereas the estimation of the mean, as well as most classical distribution estimators "only" concentrates in $O(1/\sqrt{T})$. It is even more unsettling to be able to estimate the distribution of replay being played on a dataset of $K.T$ distinct records using a single dataset of size T. Yet there is no magic at play at all here. Let us see how things work.

Before that, we need a deeper understanding of what bootstrapping is about. The bootstrapping procedure that we described in introduction and used in BRED is in fact only one example of how it can be done. It is called the non-parametric bootstrap (see Efron and Tibshirani [109] for more details). What is bootstrapping in general then? The general idea is to estimate the distribution F that generated the data. For the general considerations about bootstrapping, let us simply consider that F is a real valued distribution. The bootstrap resamples are then drawn from this estimated distribution. The non-parametric bootstrap that we used is a simple way to build an estimate of F using the data:

$$\hat{F}(x) = \frac{1}{T} \sum_{t=1} TI\left(X_t \leq x\right),$$

where $X_1, ...X_T$ are the T records contained in S. Note that this empirical cumulative density function (CDF) is a proper distribution: for instance it has a mean and a variance that are S's sample mean and variance. Yet, we only need to sample from it which can be done by sampling with replacement from S so there is no need to actually manipulate the aforementioned CDF. Such a method has nice theoretical properties [109] in addition of requiring no assumptions on the data. We will detail some of them in what follows.

As opposed to the non-parametric approach, there exist parametric ways of estimating F and thus of bootstrapping an estimator. In the case of the parametric bootstrap, we assume that F is parametrized by a vector θ of parameters. Therefore one only has to estimate θ using S in order to estimate F (or F_θ since it is parametrized). The technique obviously depends on the nature of θ and the assumptions made on F. If we denote by $\hat{\theta}$ the estimation of θ using S, we build the bootstrap resamples by sampling from $F_{\hat{\theta}}$ to which we have a full access. The rest of the procedure remains unchanged no matter how F is estimated. Note that in our situation the choice of the non-parametric bootstrap is obvious as we want to make as little assumptions as possible on the data to avoid biasing the evaluations.

Possible goals of bootstrapping an estimator are to get an estimation of its cumulative density function $J_t(x)$, a confidence interval on its output or other properties. Note that sometimes, a confidence interval can be computed analytically. For instance if the estimator is the sample mean of S, a $1-\delta$ confidence interval is as follows:

$$\left[\hbar X + \tau_{\frac{\delta}{2}} \frac{\hat{\sigma}}{\sqrt{T}}, \hbar X + \tau_{1-\frac{\delta}{2}} \frac{\hat{\sigma}}{\sqrt{T}}\right],$$

with τ_α the α quantile of the standard normal distribution. Such a confidence interval is known to converge in $O\left(\sqrt{\frac{1}{T}}\right)$ and comes from the normal approximation implied by the CLT. Yet bootstrapping is very interesting in various situations such as:

1. When the analytic form of the property we are after is complicated or does not exist.
2. When too little data is available, the bootstrap is generally much more informative [115]. One reason is that the normal approximation in this case may not be very accurate.
3. For some estimators such as the sample mean, one can hope to achieve a much higher level of accuracy in $O\left(\frac{1}{T}\right)$. This is what is announced in theorem 12.

5.6.2 The example of the sample mean

Now let us see how the bootstrap works and give intuition and hard evidence justifying why it is better than classical inference and in particular than the normal approximation. Note that the material presented here (only section 5.6.2) is mainly a reorganization of a course by Hall *et al.* [124] on the bootstrap and the corresponding Edgeworth expansions. This section is meant to help us highlight that we can take advantage of all this to evaluate recommendation policies. Another non negligible goal is to introduce the state-of-the-art results that we need in what follows using our notations. It will enable us to state our results with more clarity and precision.

To achieve the aforementioned goals, let us consider the sample mean of a dataset (or sample) S, generated by T (*i.i.d.*) samples from F, F being here a real distribution. In our situation studying the sample mean is obvious for many reasons:

- it a very simple estimator so it lets the intuition come out more clearly,
- it allows the bootstrap to achieve its best performance,
- it is relatively similar to replay.

We consider arbitrarily that we are interested in J_T, the cumulative density function of the sample mean given F. In this section we will briefly detail how a convergence in $O\left(\frac{1}{T}\right)$ instead of $O\left(\sqrt{\frac{1}{T}}\right)$ can be obtained. The first proof was given by Efron [14] but the exact arguments as presented here come from Peter Hall's book *The bootstrap and Edgeworth expansion* [124].

To be able to compare easily with the normal approximation, we consider two quantities. The first one can be seen as the error on the estimation, blown up by a factor $\sqrt{T}$:

$$e_T \;=\; \sqrt{T}(\bar{X}-\mu),$$

where $\mu = \mathbb{E}(X)$ and $\bar{X}$, the sample mean is equal to $\frac{1}{T}\sum_{t=1}^{T} X_t$. X is simply a random variable such as $X \sim F$. By CLT, if $Var(X) = \sigma^2 < \infty$ which we can very assume without risk here (the rewards are clicks or ratings so they are bounded and thus so is their variance), e_T has a limiting Normal distribution $\mathcal{N}(0,\sigma^2)$. Note that $\sigma^2(X) = \sigma^2(F) = \sigma^2(e_T)$.

The second quantity we consider is the "Studentized" form of e_T and $\bar{X}$:

$$\varepsilon_T \;=\; \sqrt{T}\frac{\bar{X}-\mu}{\hat{\sigma}},$$

where $\hat{\sigma}^2 = \frac{1}{T}\sum_{t=1}^{T}\left(X_t - \bar{X}\right)^2$ is the sample variance, an estimator of σ^2. ε_T has a limiting Normal distribution $\mathcal{N}(0,1)$. As such, ε_T is asymptotically pivotal[2], which is generally recommended by textbook on statistics when using the bootstrap. Studentization or bootstrapping a

[2]"*In statistics, a pivotal quantity or pivot is a function of observations and unobservable parameters whose probability distribution does not depend on the unknown parameters.*" *Wikipedia*

pivotal quantity in general is not a strict requirement. However considering these two quantities (e_T and ε_T) will allow us to see why it is recommended.

First let us study the approximation of F, $\hat{F}$ as it is defined in the previous section and from which one builds bootstrap resamples by sampling with replacement from S. Let us consider the estimated mean and variance of X computed from S: $\bar{X}$ and $\hat{\sigma}^2$. The key of the bootstrap is that these two quantities are the exact mean and variance of the distribution $\hat{F}$ (so $\sigma^2(\hat{F}) = \hat{\sigma^2}(S)$ which is also true for the mean and all the other moments). Moreover, we have the following concentration inequalities which are straightforward:

$$\begin{aligned} |\hbar X - \mu| &= O_p\left(\frac{1}{\sqrt{T}}\right) \\ |\hat{\sigma} - \sigma| &= O_p\left(\frac{1}{\sqrt{T}}\right). \end{aligned}$$

Note also that the same concentration results hold for the next sample cumulants as long as they exist. In particular, the third and fourth central cumulants:

$$\begin{aligned} \kappa_3 &= \mathbb{E}\left((\mathbb{E}(X) - X)^3\right) \\ \kappa_4 &= \mathbb{E}\left((\mathbb{E}(X) - X)^4\right) - 3Var(X)^2, \end{aligned}$$

as well as their standardized versions the well studied skewness ($\sigma^{-3}\kappa_3$) and kurtosis ($\sigma^{-4}\kappa_4$) concentrate in a similar fashion. This can be easily justified by the fact that all these quantities have a variance in $O\left(\frac{1}{T}\right)$. Indeed, the exact variance of $\bar{X}$ is $\frac{\sigma^2}{T}$. $\hat{\sigma}$ has a variance in $O\left(\frac{1}{T}\right)$ and in the normal case, $\hat{\sigma}^{-3}\hat{\kappa}_3$ (resp. $\hat{\sigma}^{-4}\hat{\kappa}_4$) has a variance in $O\left(\frac{6}{T}\right)$ (resp. $O\left(\frac{24}{T}\right)$). For the record, since the skewness and kurtosis are standardized, the full expression of their variance does not depend on the moments of F which is not the case for the sample mean and variance (see σ^2 in the variance of the sample mean. The variance of $\hat{\sigma}^2$ depends on the squared variance and the fourth central moment). Note that in the case of the sample mean this can be dealt with by dividing by $\hat{\sigma}$. Anyway, using for instance Theorem 14.4-1 in Bishop *et al.* [127] we have that $|X - \mathbb{E}(X)| = O_p\left(\sqrt{Var(X)}\right)$ which yields the concentration in $O_p\left(\frac{1}{\sqrt{T}}\right)$ we mentioned for these four quantities. Remark that we talked about third and fourth cumulants and not moments. This is only a minor technicality that allows simpler notations for the Edgeworth expansions we will deal with later. In fact the skewness and kurtosis are generally referred to as functions of the central moments: $\sigma^{-3}\mu_3$ and $\sigma^{-4}\mu_4$ (μ_3 and μ_4 being the third and fourth central moments). We have that $\mu_3 = \kappa_3$ so there is no ambiguity about the third cumulant/moment. Yet one should be aware that the definition of the kurtosis in terms of the fourth central cumulant, which is different from the fourth central moment ($\kappa_4 = \mu_4 - 3\mu_2^2 = \mu_4 - 3\sigma^4 \neq \mu_4$) is generally the one used in practice for it has much nicer properties. Therefore all the quantities we consider here are extremely standard. More importantly, the estimations of these quantities by the sample cumulants of S that characterize the bootstrap distribution $\hat{F}$ are, with high probability, within a distance in $O\left(\frac{1}{\sqrt{T}}\right)$ of the cumulants of F.

Now let us decide arbitrarily that we are interested in the CDF of the sample mean (one could also be interested in the PDF, a confidence interval *etc.* and derive a similar analysis). We denote the CDF as follows:

$$\begin{aligned} J_T^{(e)}(x) &= P(e_T \leq \sigma x) \\ J_T^{(\varepsilon)}(x) &= P(\varepsilon_T \leq x). \end{aligned}$$

Note that e_T and ε_T are random variables since they are computed on a sample $\{X_1, ...X_T\}$ which is itself a random variable.

Following the previously described bootstrap procedure, that is building B different bootstrap resamples of size T by sampling with replacement from S, yields an estimation of J_T:

$$\begin{aligned}\hat{J}_T^{(e)}(x) &= \frac{1}{B}\sum_{b=1}^{B} I\left(\sqrt{T}\left(\bar{X}_b - \bar{X}\right) \leq x\right)\\ \hat{J}_T^{(\varepsilon)}(x) &= \frac{1}{B}\sum_{b=1}^{B} I\left(\sqrt{T}\frac{\bar{X}_b - \bar{X}}{\hat{\sigma}_b} \leq x\right),\end{aligned}$$

where $\bar{X}_b$ is the mean of the bth bootstrap resample (of size T) and $\hat{\sigma}_b$ its standard deviation. For the record, at first Efron's bootstrap method [14] did not use sampling with replacement. It was actually necessary to consider all the T^T possible bootstrap resamples. He then replaced that condition by B bootstrap resamples drawn with the sampling procedure we just mentioned, with B going to infinity. Fortunately, in practice $\hat{J}_T(x)$ converges before that.

Showing the consistency of $\hat{J}_T$ is a classical result. It relies on the CLT (at least for the sample mean when the variance is finite) and the fact that both the Kolmogorov and the Mallows-Wasserstein distances between the real mean and the bootstrap mean go to zero asymptotically [128]. It is even strongly consistent unless the variance is infinite. In this case (that cannot happen with bounded rewards such as clicks or ratings), one can show that it is still consistent but only weakly [129]. Yet since it does not really help our understanding of what is going on, we do not discuss consistency with more precision. See Hall [124] for a full proof. The only thing to keep in mind is that we do not lose compared to the classical methods by using the bootstrap. Let us focus on the interesting part, that is how the bootstrap enables to obtain more accurate estimations of density functions or confidence intervals than the normal approximation and ends up with a convergence in $O\left(\frac{1}{T}\right)$.

To achieve this convergence, the bootstrap requires one key assumption. We need the property of the distribution of the estimator we are bootstrapping to be expandable in a specific asymptotic series. This asymptotic series have to be in the cumulants or moments of F (since we have convergence rates for them). In the case of the sample mean, Edgeworth expansions were defined to describe standardized sums of *i.i.d.* variables in terms of the cumulants of these variables [124]. Therefore the case of the sample mean is straightforward: both $J_{e,T}$ and $J_{\varepsilon,T}$ admit an Edgeworth expansion in the cumulants of F by definition. Such expansions are similar, up to differences in the polynomials p_i of the cumulants of F:

$$\begin{aligned}J_T^{(e)}(x) &= = \Phi(x) + \frac{1}{T^{1/2}}p_1^{(e)}(x|F)\phi(x) + \frac{1}{T}p_2^{(e)}(x|F)\phi(x) + O\left(\frac{1}{T}\right)\\ J_T^{(\varepsilon)}(x) &= = \Phi(x) + \frac{1}{T^{1/2}}p_1^{(\varepsilon)}(x|F)\phi(x) + \frac{1}{T}p_2^{(\varepsilon)}(x|F)\phi(x) + O\left(\frac{1}{T}\right)\\ p_1^{(e)}(x|F) &= -\frac{1}{6}\sigma(F)^{-3}\kappa_3(F)\left(x^2-1\right)\\ p_2^{(e)}(x|F) &= -\frac{x}{24}\left(\sigma(F)^{-4}\kappa_4(F)\left(x^2-3\right) + \left(\sigma(F)^{-3}\kappa_3(F)\right)^2\frac{x^4-10x^2+15}{3}\right)\\ p_1^{(\varepsilon)}(x|F) &= \frac{1}{6}\sigma(F)^{-3}\kappa_3(F)\left(2x^2+1\right)\\ p_2^{(\varepsilon)}(x|F) &= x\left(\sigma(F)^{-4}\kappa_4(F)\frac{x^2-3}{12} - \left(\sigma(F)^{-3}\kappa_3(F)\right)^2\frac{x^4+2x^2-3}{18} - \frac{x^2+3}{4}\right),\end{aligned}$$

where Φ is the standard normal CDF and ϕ is the standard normal PDF. A few things are worth noting here. The $O\left(\frac{1}{T}\right)$ at the end of the second order expansion is crucial for it forbids the remaining terms to be of higher order than what was expanded. There exists a formula to write the expansion with an arbitrary high order. We omit to give it for we only need a second order expansion here. For information, the p_i polynomials are based on Hermite polynomials.

Moreover notice that the Edgeworth expansion can be understood as a correction compared to the normal approximation. A first order expansion basically means that the distribution of the sample can be approximated by the normal distribution to which is added a skewness (=asymmetry) correction term of order $\frac{1}{\sqrt{T}}$. Likewise, the term in $\frac{1}{T}$ of the expansion consists in a first order correction term for kurtosis (weight of the tail) and a second order correction term for skewness. Intuitively, the bootstrap will outperform a normal approximation by really trying to approximate the real distribution and therefore by considering all its cumulants/moments. On the contrary, a normal approximation is only based on the mean and the variance, the first two moments.

It is interesting to know that Bhattacharya and Ghosh [130] showed that this expansion is true in a wide range of settings and uniformly for x. In particular one requires smoothness assumptions and the first four moments to be finite ($\mathbb{E}(X^r) < \infty, r \leq 4$). Their work also exhibit other estimators that could be expanded similarly. Therefore the reader interested in bootstrapping something else than the sample mean should refer to their work. Moreover, similar expansions are valid for other properties. In addition to the CDF, Hall [124] discusses confidence intervals. They can actually be expanded in a very similar fashion using Cornish-Fisher expansions that are strongly linked with Edgeworth expansions. Note that the PDF of e_T and ε_T could also be expanded in an Edgeworth expansion based on derivatives of ϕ. For more details and other properties, see Hall [124] and Bhattacharya and Ghosh [130]. We do not consider other estimators than the sample mean here for replay is very similar to it. Furthermore we will propose a new estimator that exactly fits the requirements presented in this section. This is also why we do not say much on the convergence properties of the bootstrap when the bootstrap assumptions (Edgeworth expansions in the cumulants) are not met.

Now let us discuss something that is not really obvious in the analysis of the BLB by Kleiner *et al.* [113]. It is actually natural to say that $\hat{J}_T$ admits an Edgeworth expansion in the cumulants of $\hat{F}$ if the bootstrap assumption is true for J_T with F. Indeed, we would build J_T from F exactly as we would build $\hat{J}_T$ from $\hat{F}$, that is by repeatedly draw *i.i.d.* samples of size T. Therefore it is straightforward that $\hat{J}_T$ admits an Edgeworth expansion in the cumulants of $\hat{F}$ for the simple reason that it is the CDF of the same estimator as before but computed on $\hat{F}$ instead of F. Note that this result is rigorously proved by Bhattacharya and Ghosh [130]. We do not write that expansion since it is the same as for J_T but with $p_1(x|\hat{F})$ and $p_2(x|\hat{F})$ instead of $p_1(x|F)$ and $p_2(x|F)$. They are made of the very same polynomials but with the sample cumulants of S, which are also the cumulants of the distribution $\hat{F}$. In particular,

$$\begin{aligned}
\hat{\sigma^3\kappa_3} &= \frac{T^{-1}\sum_{t=1}^{T}(X_t-\bar{X})^3}{\left(T^{-1}\sum_{t=1}^{T}(X_t-\bar{X})^2\right)^{3/2}} \\
\hat{\sigma^4\kappa_4} &= \frac{T^{-1}\sum_{t=1}^{T}(X_t-\bar{X})^4}{\left(T^{-1}\sum_{t=1}^{T}(X_t-\bar{X})^2\right)^{2}} - 3.
\end{aligned}$$

These are the classical sample skewness and kurtosis. Note that these estimators are biased. Similarly to the variance (when $\frac{1}{T}$ is replaced by $\frac{1}{T-1}$), they can be corrected at least in the normal case. Yet this is not needed for their concentration in $O_p\left(\frac{1}{\sqrt{T}}\right)$ around their true value that is mentioned at the beginning of this subsection.

This is when the magic of the bootstrap operates. We know that the sample cumulants, with high probability are within distance in $O\left(\frac{1}{\sqrt{T}}\right)$ of their true value. Thus by replacing in the expression of the $p_i(x|\hat{F})$ we have that:

$$\begin{aligned}
p_1^{(e)}(x|\hat{F}) = -\frac{1}{6}\hat{\sigma}^{-3}\hat{\kappa_3}(x^2-1) &= -\frac{1}{6}\sigma^{-3}\kappa_3(x^2-1) + O_p\left(\frac{1}{\sqrt{T}}\right) \\
&= p_1^{(e)}(x|F) + O_p\left(\frac{1}{\sqrt{T}}\right).
\end{aligned}$$

Notice that we can obtain the same thing for $p_2^{(e)}$, $p_1^{(\varepsilon)}$ and $p_2^{(\varepsilon)}$. This it is in fact true for any i.

This yields a truly fascinating result:

$$\begin{aligned}
\hat{J}_T^{(e)}(x) &= \Phi(x) + \frac{1}{T^{1/2}}p_1^{(e)}(x|\hat{F})\phi(x) + \frac{1}{T}p_2^{(e)}(x|\hat{F})\phi(x) + O\left(\frac{1}{T}\right) \\
&= \Phi(x) + \frac{1}{T^{1/2}}\left[p_1^{(e)}(x|F) + O_p\left(\frac{1}{\sqrt{T}}\right)\right]\phi(x) + \frac{1}{T}\left[p_2^{(e)}(x|F) + O_p\left(\frac{1}{\sqrt{T}}\right)\right]\phi(x) \\
&\qquad + O\left(\frac{1}{T}\right) \\
&= \Phi(x) + \frac{1}{T^{1/2}}p_1^{(e)}(x|F)\phi(x) + O_p\left(\frac{1}{T}\right) + \frac{1}{T}p_2^{(e)}(x|F)\phi(x) + O_p\left(\frac{1}{T^{3/2}}\right) + O\left(\frac{1}{T}\right) \\
&= J_T^{(e)}(x) + O_p\left(\frac{1}{T}\right).
\end{aligned}$$

Similarly:

$$\hat{J}_T^{(\varepsilon)}(x) = J_T^{(\varepsilon)}(x) + O_p\left(\frac{1}{T}\right).$$

Remark that this is much better than the normal approximation where:

$$\begin{aligned}
\hat{J}_T^{(e)}(x) &= \Phi(x) + O_p\left(\frac{1}{\sqrt{T}}\right) \\
\hat{J}_T^{(\varepsilon)}(x) &= \Phi(x) + O_p\left(\frac{1}{\sqrt{T}}\right).
\end{aligned}$$

To conclude on this analysis of the classical bootstrap, one last detail is rather important. We estimated two probabilities with the same accuracy. Yet the estimation of $J_T^{(e)}$ is only as effective as it seems if we know the real value of σ, which is rare in practice. Why is that? The answer is fairly simple. We showed that:

$$\hat{J}_T^{(e)}(\hat{\sigma}.x) = P(e_t^{(b)} \leq \hat{\sigma}.x|S) = P(e_t \leq \sigma x) + O_p\left(\frac{1}{T}\right).$$

Consequently, we estimated the probability of the error on the mean, e_T, to be less or equal than $\sigma.x$. To use that estimation efficiently, we obviously need σ. For instance, let us consider that we are interested in a one-sided 95% confidence interval, that is a number y such that $P(e_t \leq y) = 0.95$ and thus $e_t \in]-\infty, y]$ with probability 0.95. We can easily use the bootstrap distribution $e_T^{(1)}, e_T^{(2)}, ..., e_T^{(B)}$ to find a value for y such that $P(e_T^{(b)} \leq y) = 0.95$, which is a very good estimation of $P(e_t \leq \sigma x)$. Here $y = \hat{\sigma}x$, so $x = y/\hat{\sigma}$. Replacing in the concentration result, it only yields that:

$$P\left(e_t^{(b)} \leq y\right) = P\left(e_t \leq \frac{\sigma y}{\hat{\sigma}}\right) + O_p\left(\frac{1}{T}\right).$$

If σ is known, everything is fine and we can correct y by multiplying by $\sigma/\hat{\sigma}$ so as to recover the previous concentration result. Yet this is rarely the case in practice and anyway knowing the real value of the variance significantly reduces the interest of bootstrapping in the first place. In most cases σ is not known so we can not make the correction. Thus, since it implicates considering that $\sigma/\hat{\sigma} = 1$, the accuracy of the bootstrap is much more strongly correlated to the quality of the estimation of the standard deviation which is in $O_p\left(\frac{1}{\sqrt{T}}\right)$. Intuitively, this is how the quality of the estimation of $J_T^{(e)}$ should look like as well. The intuition can be confirmed by considering again the Edgeworth expansions of the CDF of e_T and $e_T^{(b)}$. Replacing x by z/σ in $J_T^{(e)}$ yields:

$$\begin{aligned} P\left(e_T \leq \sigma x\right) &= \Phi(x) + \frac{1}{\sqrt{T}} p_1^{(e)}(x|F)\phi(x) + O\left(\frac{1}{\sqrt{T}}\right) \\ P\left(e_T \leq z\right) &= \Phi(\frac{z}{\sigma}) + \frac{1}{\sqrt{T}} p_1^{(e)}(\frac{z}{\sigma}|F)\phi(x) + O\left(\frac{1}{\sqrt{T}}\right). \end{aligned}$$

Similarly, taking $x = \frac{z}{\hat{\sigma}}$ in $\hat{J}_T^{(e)}$ yields:

$$\begin{aligned} P\left(e_T^{(b)} \leq \hat{\sigma} x\right) &= \Phi(x) + \frac{1}{\sqrt{T}} p_1^{(e)}(x|\hat{F})\phi(x) + O\left(\frac{1}{\sqrt{T}}\right) \\ P\left(e_T^{(b)} \leq z\right) &= \Phi(\frac{z}{\hat{\sigma}}) + \frac{1}{\sqrt{T}} p_1^{(e)}(\frac{z}{\hat{\sigma}}|\hat{F})\phi(x) + O\left(\frac{1}{\sqrt{T}}\right). \end{aligned}$$

This allows us to compare the CDF of the distribution of $e_T^{(b)}$ that is really outputted by the bootstrap procedure with what is desired, the distribution of e_T. Let us now take the difference:

$$\begin{aligned} P\left(e_T^{(b)} \leq z\right) - P\left(e_T \leq z\right) &= \Phi(\frac{z}{\hat{\sigma}}) - \Phi(\frac{z}{\sigma}) + \frac{1}{\sqrt{T}} p_1^{(e)}(\frac{z}{\sigma}|F)\phi(x) - \frac{1}{\sqrt{T}} p_1^{(e)}(\frac{z}{\hat{\sigma}}|\hat{F})\phi(x) + O\left(\frac{1}{\sqrt{T}}\right) \\ &= O_p\left(\frac{1}{\sqrt{T}}\right). \end{aligned}$$

The previous results in $O_p\left(\frac{1}{T}\right)$ came from the fact that when taking the difference, the terms of the normal CDF canceled and the difference between the polynomials p_1 was in $O_p\left(\frac{1}{\sqrt{T}}\right)$. The latter remains valid but the former is not verified this case. We even have that $\Phi(z/\hat{\sigma}) - \Phi(z/\sigma) = O_p\left(1/\sqrt{T}\right)$ which justifies the result. Consequently, bootstrapping e_T to increase the precision is just a vain effort since it yields results that are comparable with a mere normal approximation.

We do not have this problem with ε_T the Studentized version. Indeed, we showed that:

$$\hat{J}_T^{(\varepsilon)} = P(\varepsilon_T^{(b)} \leq x|S) = P(\varepsilon_T \leq x) + O_p\left(\frac{1}{T}\right).$$

As a consequence, the estimation can be used directly to compute a confidence interval for instance or any other property based on the CDF. We have as well that the distribution of $\varepsilon_T^{(b)}$ is an estimation of $\varepsilon_T = e_t/\hat{\sigma}$. Therefore multiplying the distribution of $\varepsilon_T^{(b)}$ by $\hat{\sigma}$, that we do know, yields an estimation of e_t in $O\left(\frac{1}{T}\right)$ as previously advertised but which we could not get using $e_T^{(b)}$.

This is basically the reason why Studentizing is important when bootstrapping an estimator. Intuitively, we can explain that by the fact that the bootstrap does better than a normal approximation because it takes care of all the moments and not only the mean and variance. If when bootstrapping we do not compute the bootstrap variance σ_b^2, the effort is relatively vain. In fact one can consider that all the bootstrap effort is put into scale correction (see the first term that does not cancel when the mean is not Studentized). Note that in the case of replay, Studentization is not complicated for the estimator can be seen as a sample mean on a randomized sample (or not even randomized if the policy is deterministic). However it is worth noting that bootstrapping a non asymptotically pivotal statistic is not useless in practice. Indeed, here the analytic approximation is very easy using the normal distribution. With more complicated quantities, the analytic approximation might be very complicated, or even impossible to derive whereas the bootstrap only requires to be able to compute the quantity on a sample. See Ader *et al.* [115] or Hall [124] for other advantages of the bootstrap or more details.

All this might have seemed relatively long. Yet the detail of the analysis that we gave are crucial to understand why BRED works. The bootstrap procedure for the Studentized sample mean are given by algorithm 13. Now let us focus on how the classical bootstrap analysis as it is presented by Hall can be extended to apply to our situation.

Algorithm 13 Bootstrap of the Studentized sample mean. Note that we define a function, *BootstrapSSM* that we will use it in other algorithms.

Input

- A sample $S = \{X_1, ..., X_T\}$ made of T *i.i.d.* samples from a real distribution F of mean μ.
- An integer B (that goes to infinity in theory).

Output: An estimate of the distribution of $e_T = \sqrt{T}\left(\mu - (\bar{X})_T\right)$ given F.

```
function $BootstrapSSM(S)$
    $\bar{X} \leftarrow \frac{1}{T}\sum_{t=1}^{T} X_t$
    $\hat{\sigma} \leftarrow \frac{1}{T}\sum_{t=1}^{T}\left(X_t - \bar{X}\right)^2$
    for $b \in 1..B$ do
        for $t \in 1..T$ do
            Draw $X_t^{(b)}$ uniformly with replacement from $S$
        end for
        $\bar{X}_b \leftarrow \frac{1}{T}\sum_{t=1}^{T} X_t^{(b)}$
        $\sigma_b^2 \leftarrow \frac{1}{T}\sum_{t=1}^{T}\left(X_t^{(b)} - \bar{X}_b\right)^2$
        /* Note that using $\bar{X}$ here would yield a better estimate of the variance of $\hat{F}$. Nonetheless it is crucial to use $\bar{X}_b$ for otherwise we would have to correct the final estimated distribution using a version of $\hat{\sigma}$ computed using the true value of $\mu$, which would not be possible */
        $\varepsilon_T^{(b)} \leftarrow \sqrt{T}\frac{\bar{X}-\bar{X}_b}{\sigma_b}$
        $e_T^{(b)} \leftarrow \varepsilon_T^{(b)}/\hat{\sigma}$
    end for
    return $\frac{1}{B}\sum_{b=1}^{B} I\left(e_T^{(b)} \leq x\right)$ OR $\{e_T^{(1)}, e_T^{(2)} ..., e_T^{(B)}\}$
    /* Return the CDF OR the entire distribution */
end function
```

5.6.3 Bootstrap and data expansion

Before dealing with BRED, let us consider the problem of estimating the distribution of the sample mean on more (or less) than T records given one sample S of size T. For instance, let us consider the distribution of the sample mean computed on $T' = E.T$ records. We take $E \in \mathbb{R}*+$ with only condition that T' is a positive integer.

The way we build $\hat{F}$ does not change for we will still sample records from S with replacement. In particular, with high probability, the cumulants of $\hat{F}$ are still within distance in $O\left(\frac{1}{\sqrt{T}}\right)$ of the ones of F. What changes is that we do not draw T samples from $\hat{F}$ anymore but T'.

Let us consider the Studentized CDF $J_{T'}^{(\varepsilon)}$ and its bootstrap estimator $\hat{J}_{T'}^{(\varepsilon)}$ on a sample of size T. $J_{T'}^{(\varepsilon)}$ obviously admits an Edgeworth expansion in $\frac{1}{\sqrt{T'}}$ and the cumulants of F. For exactly the same reasons as before, $\hat{J}_{T'}^{(\varepsilon)}$ also admits an Edgeworth expansion in $\frac{1}{\sqrt{T'}}$ (and not T) and the cumulants of $\hat{F}$. Why is that? Simply because a bootstrap sample mean is computed via T' *i.i.d.* samples from $\hat{F}$ regardless of the number of samples that were used to compute this estimation of F. Therefore we can follow the very same line of reasoning as before and end up with a convergence rate in high probability for $J_{T'}^{(\varepsilon)}$. There is one difference however. Obviously the estimations of the cumulants $\hat{\sigma}^{-3}\hat{\kappa}_3$ and $\hat{\sigma}^{-4}\hat{\kappa}_4$ are within distance in $O_p\left(\frac{1}{T}\right)$ (and not T') of their respective true value. Indeed they are the sample cumulants of a sample S of size T.

Replacing that in the Edgeworth expansion yields the following result:

$$\hat{J}_T'^{(\varepsilon)}(x) = J_T'^{(\varepsilon)}(x) + O_p\left(\frac{1}{\sqrt{T.T'}}\right) = J_T'^{(\varepsilon)}(x) + O_p\left(\frac{1}{T\sqrt{E}}\right).$$

Obviously the quality of the estimation of the mean remains in $O_p\left(\frac{1}{\sqrt{T}}\right)$ for bootstrapping is not a variance reduction technique. Nonetheless this intuitively justifies why BRED's expansion allows to estimate what we want.

As a conclusion of this subsection let us therefore state this result formally. The proof is what precedes. Since many different conditions can be given for the bootstrap, we let the interested reader refer to Bhattacharya and Ghosh [130] that studied lots of them.

Lemma 1. *For any set S of T independent observations from a real distribution F, any integer T', any property ξ of an estimator θ such that some smoothness assumptions on θ are respected (see Bhattacharya and Ghosh [130] for more details but note that Hadamard differentiability of θ at F is one such condition), if $\mathbb{E}_{X\sim F} X^r < \infty$ for $r \leq 4$, and if $\xi(\theta_{T'})$, that is ξ when θ is computed on T' i.i.d. observations, admits an Edgeworth expansion in T' and the cumulants of F (this is true uniformly on x for the CDF or PDF of the sample mean or in general for estimators based on a sum of these T' observations), then we have that the non-parametric bootstrap estimator $\xi(\hat{\theta}_{T'})$ yielded by repeatedly computing θ on samples S_b built by sampling T' observations with replacement from S is a consistent estimator of $\xi(\theta_{T'})$. Moreover we have that:*

$$\left|\xi(\theta_{T'}) - \xi(\hat{\theta}_{T'})\right| = O_p\left(\frac{1}{\sqrt{T.T'}}\right).$$

Note that a similar result could be derived with Cornish-Fisher expansion so as to include confidence interval in the previous result. This result needs one more comment. If one only has the estimation of the variance of the estimator in mind , one may think that expansion is fairly useless. This is what we did with the experiment on figure 5.4 but only for the sake of showing that it worked. Indeed, when the variance of the sample mean is estimated for a sample of size T with some precision depending on T, dividing this estimation by E yields an estimation of the variance of the sample mean on a sample of size $E.T$ with that same precision depending on T (not on $E.T$ obviously). This is not the case when using bootstrapping to estimate a CDF, a PDF or a confidence interval. Indeed, these objects depends on all the cumulants which may converge differently toward those of a normal distribution when the sample size increases. The Edgeworth expansion clearly explains that dependence on the various cumulants and in particular the third one which yields the most significant correction compared to a normal approximation. By estimating F by $\hat{F}$ the bootstrap takes into account all the cumulants.

This is why expanded bootstrap can be useful, and in particular better than both classical analytic methods and corrections (as for the variance) of the result of a bootstrap for a different expansion factor. Indeed, one has to remember what happens when $\hat{\sigma}_b$ is not computed at each bootstrap step but instead the result is corrected with $\hat{\sigma}$ afterwards (not Studentized *vs.* Studentized case): it does not offer the bootstrap guarantee in $O_p\left(\frac{1}{T}\right)$. Same thing would be true by correcting a bootstrap estimate of a different sample size to obtain the one we want. We would get a fair estimation but not as accurate as the one from an expanded bootstrap.

Remark In this work we are interested in values of T' that are bigger than T, hence the term expansion. The expansion lemma (lemma 1) as we call it actually also justifies the use of the bootstrap for $T' < T$. Indeed one interested in the distribution of an estimator on a sample of size T' who possesses a sample S of size T should sample from the totality of S and not from a sub-sample of size T'. This yields an estimation in $O_p\left(1/\sqrt{T.T'}\right)$ instead of $O_p\left(1/T'\right)$ for the regular bootstrap. This is nice when $T' < T$.

5.7 A precise analysis of what BRED can achieve with stationary policies and how accurately it does so

5.7.1 Applying the expansion Lemma directly

We mentioned that replay is more or less a sample mean. BRED consists in bootstrapping replay on a sample S of size T, using an expansion factor $E = K$. Taking into account the rejection mechanism, replay is a mean of rewards computed on $\frac{T}{K}$ records on average. For the sake of intuition, let us consider for a moment that replay is in fact a sample mean computed on exactly $\frac{T}{K}$ *i.i.d.* real valued samples. Bootstrapping it without expansion would yield an estimation of $CTR_\pi(T/K)$. With high probability, the estimation of the CDF would be in $O\left(\frac{K}{T}\right)$ and the estimation of g_π in $O\left(\sqrt{\frac{K}{T}}\right)$.

Now in this simplified situation, BRED would be like using the bootstrap of the sample mean on expanded data exactly as described in the previous section, with an expansion factor $E = K$. Therefore replay and its bootstrap expansion would admit Edgeworth expansion and we would have distance in $O_p\left(\sqrt{\frac{K}{T}}\right)$ between the cumulants and their bootstrap counterparts. Consequently simply using Lemma 1, with a sample of size T/K expanded K times by bootstrap we would obtain:

$$\widehat{CTR}_\pi(T)(x) = CTR_\pi(T)(x) + O_p\left(\frac{\sqrt{K}}{T}\right).$$

The expansion Lemma does not yield any improvement as far as the estimation of g_π is concerned which remains in $O_p\left(\sqrt{\frac{K}{T}}\right)$.

What about the Bag of Little Bootstraps by Kleiner *et al.* [113]? At each bootstrap step, they subsample $\sqrt{T}$ distinct records and sample with replacement T records from this sub-sample. This means that at each bootstrap step, $\hat{F}$ is different but one detail remains: with high probability its cumulants are within distance in $O\left(\frac{1}{\sqrt{\sqrt{T}}}\right)$ of F's ones since they are the sample cumulants of a sample of size $\sqrt{T}$. Therefore, if one can assume to have an Edgeworth expansion in T and in the cumulants of F for the CDF, then one exists for the CDF computed via bootstrap from all the different $\hat{F}$. This expansion is also in T and uses the cumulants of $\hat{F}$. Using classical bootstrapping theory does not yield an approximation in $O_p\left(\frac{1}{T}\right)$ for the bootstrap CDF. Rather, with the basic BLB we obtain the following result:

$$\hat{J}_T(x) = J_T(x) + O_p\left(\frac{1}{T^{3/4}}\right).$$

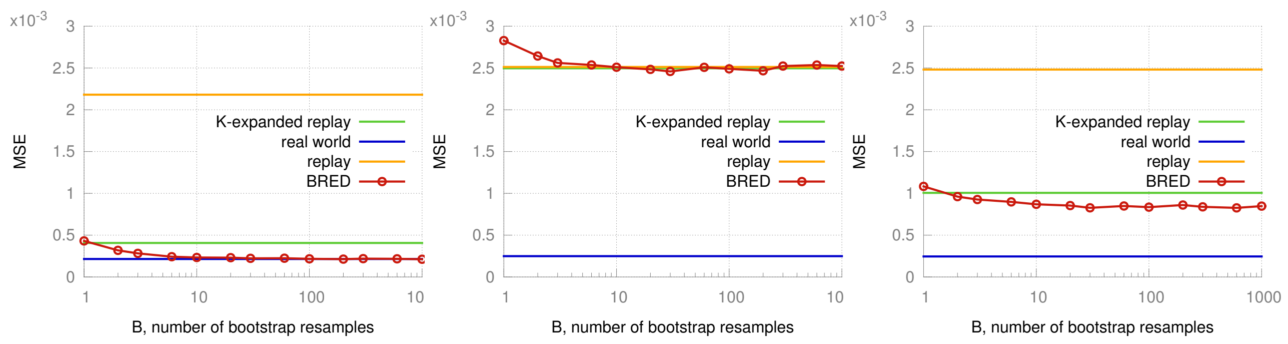
To overcome this, Kleiner *et al.* [113] actually introduced an additional mechanism. Instead of considering all the resamples one by one, they group them in bags of L. Then they compute the bootstrap estimate on each resample and the mean of these estimates for each bag. The bootstrap estimation of the distribution is produced by only gathering these means. By this process, as L goes to infinity, they gain an additional order of convergence and recover a rate in $O_p(1/T)$. It is very likely that this trick has a cost in terms of precision compared to the original bootstrap. Unfortunately their analysis does not highlight it. Indeed the analysis of the bootstrap being asymptotic, many things can be ignored. Such a cost is exactly what was proved for the *m out of n bootstrap* [131]. The *m out of n bootstrap* (that subsamples m out of n samples without expansion) was designed to deal with situations in which the classical bootstrap is inconsistent. Obviously there is a cost in terms of bias and variance that is quantified by Bickel *et al.* [131] (note that this reference is much more recent than the first paper which was published in 1997 by the same authors but it provides more details). This cost is quantified in spite of the fact that the bootstrap analysis is asymptotic. The cost for the BLB is less important as the order of convergence is the same. Yet quantifying it as a function of the parameters of the problem would probably be helpful in practice. As an example, in chapter 3, we recall that we had 3 estimators (the ideal experiment, replay and replay*) with all the same order of convergence. Yet we have been able to quantify the cost of replaying past data (replay), and then to acquire unbiasedness (replay*).

Here this is in fact what we want: quantifying the cost playing a trick with the bootstrap (we only have T partially labeled records). Indeed the bootstrap was tested for many years and although its theory is asymptotic, we know that it works as expected on small samples and in many situations in general [109]. Also, when having a look at the theory presented in section 5.6.2, the fact that it works just makes sense for bootstrapping is actually the fact of sampling from a decent approximation of the generative distribution. What we want is not to say that asymptotically BRED works more or less as the bootstrap of the sample mean (which we did in our first analysis based on the BLB). What we want is to be able to quantify the cost (if any) of BRED's rejection-expansion process by using the classical bootstrapping tools that we introduced in this section and are empirically proven to be representative on what is happening even on smaller samples.

5.7.2 Practical differences between BRED and the regular bootstrap

We basically proved that for the sample mean, expansion is useful. Indeed, it allows to estimate the distribution of the sample mean on more (or less but that was kind of obvious) than T records. Nonetheless we also exhibited that expansion does not allow to magically improve the convergence of both the mean and CDF estimates. Note however that in the case of the CDF, the expanded bootstrap allows in that case a much better convergence rate than classical methods: $O_p\left(1/T\sqrt{E}\right)$ *versus* $O_p\left(1/\sqrt{t}\right)$.

Is it the same for replay? Is it hopeless to try to do better than replay in terms of accuracy? As a matter of fact we already exhibited that it was possible. Before anything, let us recall that bootstrapping *is not* a variance reduction technique. As such it does not normally allow increased accuracy. One who is familiar with bootstrapping knows that using it to get an estimation of the mean of a classical estimator (that is not stochastic) is preposterous. Indeed, the variance of the bootstrap estimator of the mean (or bagged estimate) converges from above



(a) Random uniform policy. (b) Optimal policy (deterministic). (c) ε-optimal policy ($\varepsilon = 0.5$).

Figure 5.6: Mean Squared Error (MSE) on the per-trial reward estimation made by BRED over the number of bootstrap resample B. The MSE is compared to three baselines for three representative policies. The plots are on the same scale for a clearer comparison. *K-expanded replay* stands for using replay on the concatenation of K duplicates of the dataset.

as B increases to the variance of the actual estimator computed normally and only once on the sample. If B is kept small, then the bootstrap estimator of the mean is a terrible estimator of the mean compared to the very simple mean for it is both less accurate and more expensive to compute. Furthermore it is obvious that in general data expansion does not help making more accurate a classical estimator such as the sample mean.

Now that this is clear, let us have another look at the figures described in section 5.5.2. First of all, we can see on figure 5.2 that in the case of replay, the expansion by a factor K, without bootstrapping increases evaluation accuracy (see the different between replay and BRED with $B = 1$). Moreover, figure 5.3b exhibits further improvements when B increases. Another unsettling fact when talking about bootstrapping can be noticed on figure 5.3b. S-BRED(B=1) is in fact replay played on an dataset expanded K times: no sampling with replacement is going on and all the records are used K times exactly. Therefore this is the estimator actually bootstrapped by BRED. If replay were a classical sample mean, we would witness a convergence of the variance (or MAE) of the bootstrap estimate of the mean toward the variance of the estimation of S-BRED(B=1) which would be equal to the variance of replay. Nevertheless it is clearly not true and we witness increased accuracy with both data expansion and bootstrapping whether they are used conjointly or separately.

Before trying to justify that, let us seek more intuition about what we are after. To do so we measured the accuracy of BRED (regarding the mean g_π) as B evolves for the three policies we considered before (random uniform, optimal and ε-optimal) and that seem representative of the various cases that can arise. While keeping in mind that when bootstrapping a regular estimator, the variance of the bootstrap estimator of the mean tends from above toward that of the simple estimator, let us compare the variance of BRED with a few baselines. These baselines are naturally the following:

1. T interactions with the real world,
2. replay on a dataset of size T,
3. replay on a dataset of size T expanded K times (note that if replay were a classical estimator, this should be equivalent to number 2)

The third baseline is the estimator we are actually bootstrapping, the second the estimator without expansion and the first what we aim at.

As it can be seen on figure 5.6, BRED behaves exactly as the bootstrap of a sample mean when evaluating a deterministic policy (fig. 5.6b). The variance of BRED is higher (thus the accuracy is lower) than the simple version of the estimator being bootstrapped: replay. Also note that the variance of replay does not change when expanding the data, exactly as for the sample mean. For stochastic policies, things are different. For both of them, when $B = 1$, the variance is slightly higher than replay on expanded data. This makes sense for the data is not used uniformly because of the sampling with replacement. Yet it is already way lower than the variance of the simple replay. Then, as B grows, the variance decreases quickly and converges to a value smaller than the variance of an evaluation by replay on an expanded dataset. For the random uniform policy (see fig. 5.6a), this limiting value is the variance of an evaluation in the real world. Nevertheless this is not the case for ε-optimal (fig. 5.6c) which remains significantly larger than the variance of an evaluation in the real world.

The explanation is rather intuitive. When replay is evaluating random policies, data expansion clearly helps gaining accuracy. Indeed, it allows the policy to make different choices on different records and explore more areas of the dataset, letting the evaluation acquire more **knowledge** and thus more precision. This is approximately what happens when bagging regressors or classifiers [119]. Yet what increases accuracy with classical bagging is the fact that classifiers and regressors, in particular those based on trees are not smooth functions of the data. As such one small alteration of the data (or even in the order it is presented) can sometimes lead to big changes in the trained model. This allows big changes from one bootstrap estimate to another which correspond to different interpretations of the data. Thus averaging them is more informative than taking only one of them: bagging trees lets them extract more knowledge from the data. Replay is smooth but the stochasticity of a policy has the same effect with regard to knowledge extraction than non-smoothness here. This is not the point of this work at all but note anyway that the construction of classifiers and regressors can be randomized as well, sometimes leading also to further improvements in the case of bagging. Anyhow, our analysis based on the sample mean only explains why BRED estimates the correct distribution as exhibited empirically in section 5.5.3. Nevertheless it does not explain why the accuracy is so high. Although the reason is intuitive, the fact that it is the bootstrap that enables this accuracy gain is quite strange but more importantly it does not allow a straightforward analysis of the precision of both the mean estimation and the distribution estimation. This is exactly the problem people have with bagging. We want to fix that in what follows. To do so, we will start with quantifying the knowledge contained in a dataset regarding a policy. Then we will derive a simple estimator that allows to extract all of it. Finally we will set our interest back on bootstrapping. In particular we will demonstrate that replay can be bootstrapped and that the Studentized accuracy can be reached. We will conclude by showing how to bootstrap the maximum knowledge estimator we just mentioned with the Studentized accuracy.

5.7.3 Quantifying the extracted knowledge

For a deterministic policy BRED behaves similarly as the bootstrap of a regular sample mean, that is no additional variance gain can be obtained via expansion or increasing the number of bootstrap resamples B. Therefore we can reasonably conclude that BRED does not manage to extract more knowledge from S than replay. Moreover one can assume without too much risk that bootstrapping theory could apply without much change in that case but we leave that for the next section. Things are different for stochastic policies. Yet, leaving bootstrapping aside for a moment, we can make a very simple interpretation. If we manage to reduce the variance of the evaluation further than what replay enables, it means that there is more "knowledge" in the data than what one simple replay can find. In fact it only makes sense: since the policy can make different choices given one context, replaying the data a second time, a third time *etc.* enables each time to drag more knowledge out of the data until everything was extracted. How can we quantify that knowledge we extract in order to study it? Actually we have already more or less done it. Intuitively, the more we reduce the variance, the more knowledge we drag out of the data regarding the evaluation of the considered policy. This is exactly the definition of *Fisher information*, which is a so commonly used measure in information theory that it is sometimes only referred to as just "information". See Lehmann and Casella [132] p.115 for an introduction. Formally, Fisher information is defined as the variance of the score. In the particular case we consider, information about some estimated parameter can be reduced down to the inverse of the variance of its estimation. This yields a very intuitive way of quantifying the amount of knowledge that we manage to drag out of S. To compare different policies in our situation, we can compare the amount of normalized Fisher information γ_π (this is not standard but fits our needs very well), that an estimator manages to acquire from a dataset S:

$$\gamma_\pi = Var\left(\hat{g}_\pi^{(estimator)}(S)\right)^{-1}.Var\left(\vec{r}[\pi(x)]\right).$$

Note that if the estimator is not unbiased, one should consider its expected squared error instead. Indeed otherwise we would we considering information about g_π combined with the bias of the estimator. Note that γ_π is not exactly the Fisher information about g_π extracted by the estimator. We multiply it by the variance of one interaction of the policy with the real world. Doing so removes the dependency of γ_π on the inherent variance of the policy when it interacts with the environment. This also makes γ_π without unit and much more interpretable. Indeed by multiplying by the variance of one interaction with the real world, we in fact divide by the Fisher information contained in one such interaction. Therefore γ_π measures information not as Fisher would but in terms of number of real world interactions (or fully labeled records). For instance we saw in chapter 3 that the variance of replay was in $O\left(\frac{K}{T}Var(\vec{r}[\pi(x)])\right)$. Therefore γ_π is in $O\left(T/K\right)$ interactions. The variance of a real world evaluation is $Var(\vec{r}[\pi(x)])/T$ thus γ_π is obviously equal to T interactions. Although these two examples are policy independent, γ_π quantifies the information contained in S regarding the CTR of a given π. As our previous experiment showed it $\gamma_\pi^{(BRED)}$, the quantity of knowledge extracted by BRED, does depend on the policy. Note that we will see later that this is in fact true for replay as well.

We will not consider γ_π exactly but some quantity computed from it that reflects time acceleration. Since γ_π is simply proportional to the size of the dataset (in expectation for it also depends on the data) given that everything is *i.i.d.*, one can be interested in the per record information contained in S: $\frac{\gamma_\pi}{T}$. Intuitively, $\frac{\gamma_\pi}{T}$ should be between $\frac{1}{K}$ (for a deterministic policy) and 1, the ideal scenario that seems to be reached for the random uniform policy. In general, the value should be between these two values, like what happens for ε-optimal. To match what was said in introduction, we will rather consider the inverse of the per record information:

$$\Gamma_\pi = \frac{\gamma_\pi}{T}^{-1} = \frac{T}{\gamma_\pi} = \frac{T}{Var(\hat{g}_\pi)/Var(\vec{r}[\pi(x)])} = \frac{T.Var(\vec{r}[\pi(x)])}{Var(\hat{g}_\pi^{(estimator)}(S))}.$$

Γ_π measures the loss of information entailed by considering a partially labeled dataset. It tells how many partially labeled records it takes to get as much information about π as from one interaction in the real world. In other words, it is the time acceleration coefficient: K for a deterministic policy, 1 for the random uniform policy and hopefully somewhere in between for the others. With replay in expectation and without more prior information, we have $\Gamma_\pi^{(replay)} = K$ for all policies. Let us denote by $\gamma*_\pi/T$ (resp. $\Gamma*_\pi$) the maximum of information (resp. the minimum of time acceleration) in the sense of pure replay given a dataset S. The optimum value for Γ_π in the sense of replay means that we only allow replay mechanisms to drag information out of the data. For instance BRED, although it expends the data only uses replay on records available in S in order to make its estimation. Thus it fits our requirements here. Other mechanisms could drag even more information but we ignore them for now. Nonetheless we will exhibit one in section 5.10.1.

In what remains of this section, we will be interested in two major points regarding our problem:

1. Now that we have a way to measure information, what estimator drags the amount amount of information $T/\Gamma*_\pi = \gamma*_\pi$ from S with certainty, given that replay and the "K times expanded replay" wrapped inside BRED do not do that? Additionally, how can we characterize $\Gamma*_\pi$?

2. What about the bootstrap? Replay is obviously smooth enough to be bootstrapped successfully. Yet can it be bootstrapped with the Studentized accuracy as the sample mean? And what about our new maximum information estimator? Can it be bootstrapped and with what kind of accuracy?

5.7.4 RED∞: A maximum information estimator

It is in fact very easy to build an estimator that is going to extract all the knowledge embedded in the dataset regarding a given policy that does not need to be bootstrapped or expanded to do so. Such an estimator will thus have a deterministic behavior and as such it will be much easier to analyze regarding both raw precision and its properties when being bootstrapped. For a deterministic policy, one scan by replay is enough for given a dataset, replay is deterministic and BRED does not allow to gain accuracy. Note that we will show how to acquire more knowledge than what replay allows to extract even for deterministic policies but with different mechanisms than those at play here.

For stochastic policies we have noticed that letting the expansion factor or B grow increases the accuracy of BRED, up to a point where no more improvement is possible. Therefore with this in mind, an intuitive "optimal" estimator is an estimator that performs replay infinitely on the data or at least until convergence. We call such an estimator $RED\infty$, which stands for Replay on Expanded Data, plus an infinity symbol to express that data is expanded infinitely. All this does not seem revolutionary. We already knew that to reach the highest possible accuracy, we had to increase the expansion factor or the bootstrap parameter B which for a static policy is more or less the same. Yet there is one crucial difference of concept with BRED. With BRED, we were bootstrapping replay, nothing more. Then, expansion was introduced inside the bootstrap process, without altering the estimator. The goal of doing this is to be able to estimate $CTR_\pi(T)$ (or for any other value T') instead of $CTR_\pi(T/K)$. Note although this is a detail that what the BRED procedure actually estimates is $CTR_\pi^{(replay)}(K.T)$, that is replay on a dataset of size $K.T$ but we proved that it was close enough to $CTR_\pi(T)$ to maintain the same order of convergence. BRED did allow to reach the optimal precision for the mean of the CTR in practice. Yet the fact that BRED does not behave like a regular bootstrapping process (see fig. 5.6) is really disturbing for the overall analysis.

With that regard, $RED\infty$ has the major quality of not needing to be bootstrapped in order to achieve its best accuracy. Such a behavior is much more standard and easy to analyze than what was observed with BRED. In particular it will be easier to exhibit the accuracy of this algorithm depending on the policy (note that this is equivalent to quantifying Γ^*_π) and later to show how the bootstrap works and converges in that case.

Before studying RED∞, let us describe how it works with precision. First of all, it is important to keep in mind that contrarily to BRED, RED∞ is not a bootstrapping algorithm. BRED was bootstrapping replay, using expansion. RED∞ uses expansion on its own. Since π is a simple function, possibly stochastic, there is no need to introduce additional randomness. Thus, RED∞ simply consists in performing replay successively until convergence of the mean. Note that consequently, all the records are considered the same amount of time. Here one can think of many stopping rules. In theory we will consider the expansion factor E to be infinite. In practice, we will simply set it to some high value but we could also monitor the estimate and stop when it stops wavering. This is exactly what is done in general with the bootstrap parameter B [14]. The most important difference with BRED comes when RED∞ is bootstrapped. Simply repeating RED∞ on S or even using an infinity of samples with replacement would lead to a one bar histogram for RED∞ is deterministic even for stochastic policies. In practice the fact of stopping before $E = \infty$ and possible numerical approximations by the computer would leave a little bit of randomness. Nevertheless, the "bootstrap" distribution, if we can still call it that way, would actually be nothing more than an estimation of the noise engendered by performing RED∞. Indeed, for the bootstrap to work, we have to sample from an estimation of F (no matter how many times), which is no longer done here. In fact one has to bootstrap RED∞ exactly as a regular estimator, exactly as replay was bootstrapped. At each bootstrap step, we have to create a bootstrap resample of size T', where T' can be any value and compute RED∞ on the resample. Therefore note something quite interesting: we can in fact expand the bootstrap evaluation of RED∞ in order to estimate $CTR_\pi^{(RED\infty)}(T')$ for an arbitrary T'. The bootstrap expansion is thus independent from the infinite expansion within RED∞. Note also that $RED\infty$ is a different estimator than replay. Therefore if we can say that the distribution estimated by BRED, $CTR_\pi^{(replay)}(K.T)$ is approximately equal to $CTR_\pi(T)$, it may not be the case in general for $CTR_\pi^{(RED\infty)}(T)$. We will give more details when studying the bootstrap properties of this new estimator.

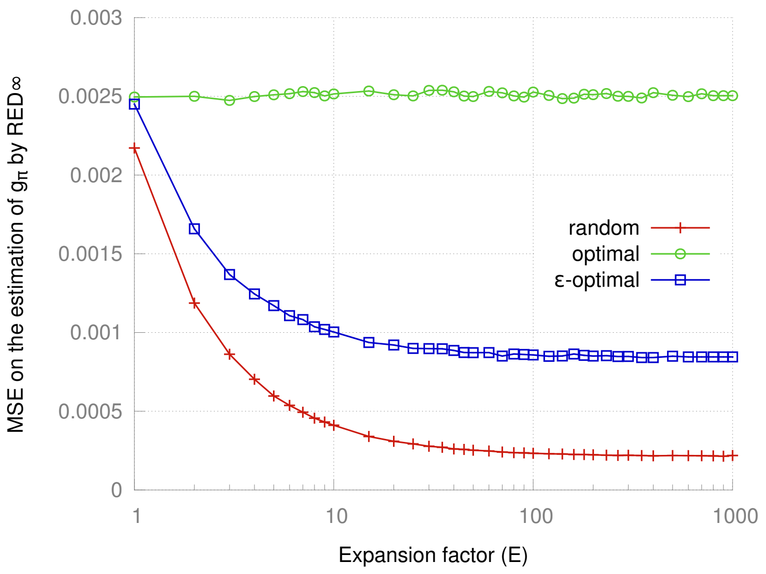


Figure 5.7: Mean Squared Error (MSE) of RED∞ (so without bootstrap whatsoever) on the estimation of g_π (which is the only thing it is meant to estimate) over the expansion factor E. The experiment was run with $T = 1000$ and $50,000$ runs were averaged. This was expected given our previous experiments with BRED since for the estimation of g_π RED∞ with a parameter E is equivalent to S-BRED with a parameter $B = E/K$. Notice simply that $E = 100$ (which is very likely to depend on K so let us say $E = 10.K$) is enough to reach a near optimal precision. After $E = 30.K = 300$, no significant improvement can be measured anymore. Obviously, the deterministic policy does not take advantage of RED∞ (note also that RED∞(E=1) = replay by definition).

Concluding on how RED∞ works, we can see on figure 5.7 the amount of expansion needed to make it work as if the expansion factor E were infinite. More details on the experiment are available on the figure.

Remark When using RED∞ in this setting, it is possible to reduce it to the computation of its limit as we detail it further in this document (section 5.7.7). This way we will really compute RED with $E = +\infty$. Therefore this study of the impact of E may seem irrelevant. Yet we will also see that we can introduce mechanisms such as smoothing in addition to expansion to increase precision (see section 5.10.1). In that case the aforementioned reduction is not valid anymore and we can only increase E until convergence as studied here, hence the significance of what is exhibited on figure 5.7.

5.7.5 Rigorous analysis of the precision of replay's bootstrap

So far we have basically defined a goal for precision of the evaluation of the mean of a policy. This goal, embodied by the time acceleration factor Γ_π^* will be quantified during our variance analysis in the next section. For now what is important to remember is that any algorithm estimating g_π (the mean) by performing replay infinitely (or until convergence in practice), achieves that goal. Now what about the estimation of the distribution of the mean? How accurate is it? Does BRED achieve the Studentized accuracy exhibited in section 5.6.2?

First what is the difference between a sample mean and the simple replay estimator? The real world estimator we want to fake using replay is an actual sample mean. It basically consists in sampling T observations from a distribution F_π, which is the distribution of a random variable r generated by the following process:

1. $x \sim \mathcal{C}$,
2. $a \sim \pi(x)$,
3. $r \sim \mathcal{R}_{x,a}$.

We denote by F_π^{all} the distribution of a tuple (x, a, r). To simplify notations here but more particularly in what follows, note that we stop considering that only one distribution $\mathcal{D}$ generates both the context and the rewards of every actions (the tuple $(x, \vec{r})$). Instead we divide it into two separate entities. We consider the distribution $\mathcal{C}$ that generates a context. We also consider that one reward is generated by a distribution $\mathcal{R}$ parametrized by both one context and one action. This is just a mere technicality and changes nothing about the problem. Nonetheless the fact that the reward and the context are generated separately and the fact that only the considered reward needs to be generated leads to clearer notations in the following arguments.

The dataset S we consider is composed of T tuples (x_t, a_t, r_t) generated by $F_{\pi_{LOG}}^{all}$, where π_{LOG} is the random uniform policy. We know from theorem 2 from chapter 3 that a sequence of L evaluations $(x_0, a_0, r_0), ..., (x_L, a_L, r_L)$ has the same probability to be obtained by replaying a policy π on a dataset S generated by $F_{\pi_{LOG}}^{all}$ than to be obtained by simply sampling from F_π^{all}, provided that S is big enough to replay L simulated interactions. The theorem states that S has to be infinitely big. Indeed otherwise it is simply impossible to ensure that L interactions can be simulated by replay, regardless of the size of S. This statement might cause to question the applicability of this theorem in practice. Yet we can somehow reverse the result and say that given that a sequence of size L comes out of an evaluation via replay, it is impossible to say *a posteriori* whether it was generated by F_π^{all} or by replaying π on a dataset generated by $F_{\pi_{LOG}}^{all}$. Note that all this is also true for any algorithm A, and not only static policies. Consequently let us thus stress again that this is a very strong result, in spite of its being asymptotic.

As a consequence, we can rephrase what replay is doing: it takes a certain number of samples from the real valued distribution F_π (F_π^{all} rid of the contexts and actions) and averages them to produce an estimation of g_π. Provided that at least one sample can be drawn using S, $\hat{g}_\pi^{(replay)}$ is unbiased. Again without conditioning on this very likely event, replay can only be said to be asymptotically unbiased, unbiased with (very) high probability or even endowed with a bias that decreases exponentially. In the common sense, it is biased.

What does it mean as far as bootstrapping is concerned? How is it any different to bootstrap replay or a sample mean? The answer is fairly obvious. When a bootstrap resample is built from S, the size of the sample $T^{(b)}$ of F_π produced by replay is a random variable, of expectation $\frac{T}{K}$. This is problematic when we want to plug that in bootstrapping theory since the existence of an Edgeworth expansion in $\frac{1}{\sqrt{T'}}$ for the bootstrap CDF is conditioned on the fact that it is a sample mean on T' *i.i.d.* observations. Here we only compute the mean of a random number of observations, whose expectation is known but of little help in our context. With stochastic policies, there is not much we can do about that: replay does yield a random number of observations. With deterministic policies however, one can bypass this problem very easily. Let us study how this can be done and what it implicates.

Bootstrapping the replay of a deterministic policy: a simpler case

Bypassing the problem of the number of evaluations $T^{(b)}$ that changes from one bootstrap resample to another is simple when the policy is not stochastic. Indeed one can derive an auxiliary real valued dataset S_π on which computing the sample mean is equivalent to replaying π on S. The procedure is straightforward: since the policy is deterministic only one Kth on average of the records of a dataset are really useful. Furthermore given a dataset, these useful records are always the same. The other records are never used, regardless of the number of bootstrap resamples or the size of the expansion factor. Therefore one can create a dataset S_π, of size T_π that contains all the rewards obtained by π when being replayed on S. Thus $\mathbb{E}(T_\pi) = \frac{T}{K}$ but it can sometimes deviate significantly from its mean. In the worse case, if $T_\pi = 0$, then there is nothing we can do. Indeed, in the sense of replay, S does not contain information regarding the evaluation of π. The rest of the time, the sample mean of S_π yields an unbiased estimate of g_π. Also from theorem 2 we know that it is impossible to distinguish the sequence of S_π rewards obtained here from a sequence of rewards directly drawn from F_π. As a consequence, by bootstrapping the sample mean of S_π, we obtain an unbiased estimator of $CTR_\pi(T_\pi)$. Since bootstrapping replay is here reduced to the bootstrap of a sample mean, which rigorously justifies the fact that it reaches the Studentized accuracy, we refer to this bootstrapping procedure as *Rigorous BRED for Deterministic Policies* (R-BRED-DP). More precisely, the accuracy of R-BRED-DP is as follows:

$$\left|\widehat{CTR}_\pi^{(R-BRED-DP)}(S)(x) - CTR_\pi(T_\pi)(x)\right| = O_p\left(\frac{1}{T_\pi}\right),$$

which simply stems from the accuracy result of the sample mean of section 5.6.2. Furthermore notice that since T_π is the number of records that allows an evaluation of π, $\gamma_\pi^* = T_\pi$ by definition. Therefore
$\Gamma_\pi^* = \frac{T}{T_\pi}$. As a consequence:

$$\left|\widehat{CTR}_\pi^{(R-BRED-DP)}(S)(x) - CTR_\pi(T_\pi)(x)\right| = O_p\left(\frac{\Gamma_\pi^*}{T}\right).$$

We can then introduce expansion to estimate the distribution of the CTR after $E.T_\pi$ interactions for any E and so with a precision in $O_p\left(\frac{1}{T_\pi\sqrt{E}}\right)$ (expansion lemma). In particular we can use expansion to get an estimation of $CTR_\pi(T)$ which yields the following theorem.

Theorem 14. *Under the conditions of the expansion lemma (lemma 1) and for any* ***deterministic*** *policy π, the procedure R-BRED-DP with an expansion parameter T (i.e. that builds bootstrap resamples of size T) produces an estimation of $CTR_\pi(T)$ - the distribution of an estimator of g_π based on the mean of T real world evaluations - that concentrates as follows:*

$$\left|\widehat{CTR}_\pi^{(R-BRED-DP(T))}(S)(x) - CTR_\pi(T)(x)\right| = O_p\left(\frac{\sqrt{\Gamma_\pi^*}}{T}\right).$$

Obviously these concentration results depend on the dataset, and in particular on the knowledge it contains with respect to π. If we want a result that does not depend on the knowledge contained in the data but simply on the quantity of data, we have to take into account the variability on T_π, and possibly include it in a looser Edgeworth expansion that would have to be calculated. Moreover since T_π can be equal to zero, all the estimators will have a bias (although exponentially low). Nevertheless before diving into an analysis blindly, one may first question the practical interest of such a result. Regardless of the value of T, the key variable on which the quality of the estimation provided by replay depends is T_π. This quantity is directly accessible given any dataset. Having access to a theoretical result depending only on T would of course be interesting intellectually but not very useful in practice since once T_π is known. Moreover we get a more precise result than the general one based on T. The only practical case in which such a general result would be interesting is the case of determining when to stop acquiring data in the real world when the policies that will be evaluated are *unknown* at the time. Indeed, if they are known, their respective value for T_π can be easily determined as well as we go. In the case where they are unknown, we could only refer to a general result based on T to decide. Nevertheless, one interested in reaching a minimum precision for any policy can consider a lower bound on T_π with high probability, for any π. This is done easily using Chernoff for instance as in chapter 3, since T_π is nothing more than a Binomial random variable. Then one can plug that lower bound into the result we have for the quality of the estimation of the CDF by R-BRED-DP to get a fair idea of the minimum precision we would reach with high probability. An implementation of the R-BRED-DP procedure is provided in algorithm 14.

Algorithm 14 Rigorous BRED for deterministic policies (R-BRED-DP).

Input

- A dataset $S = \{(x_1, a_1, r_1), ..., (x_T, a_T, r_T)\}$ logged by playing a random uniform policy T times against the real world $\mathcal{D}$
- a deterministic policy π

Output: An estimate of $CTR_\pi(T_\pi)$, or any other T' using expansion.

```
$S_\pi \leftarrow \emptyset$
for $t \in \{1..T\}$ do
    if $\pi(x_t) = a_t$ then
        $S_\pi \leftarrow S_\pi + \{r_t\}$
    end if
end for
return $BootstrapSSM(S_\pi)$
```

A general approach

We know that BRED evaluates $CTR_\pi(T)$ correctly in practice and in theory (theorem 12) for stochastic policies as well but can we show how well? As a matter of fact we most probably could. Indeed to work optimally in theory, the bootstrap needs an Edgeworth expansion. The problem we have is that from one bootstrap resample to another, $T^{(b)}$ changes for it is a random variable. Since it is binomial, $T^{(b)}$ has known concentration properties. Moreover bootstrapping theory is only true with high probability. Therefore we suspect that we could modify the expansion of the sample mean so that it takes into account the variability on $T^{(b)}$. We also suspect that it would also lead to a result in $O_p\left(\frac{K}{T}\right)$ with an almost negligible cost. Although it seems possible we did not do it for it seemed rather complicated whilst much simpler alternatives can be considered. The simplest method uses the expansion lemma (lemma 1) and is as follows. Instead of sampling $K.T$ records with replacement from S, one can sample records with replacement until T evaluations are performed. In some sense it is a way to take advantage of expansion to stabilize $T^{(b)}$. Let us call this procedure Rigorous BRED (R-BRED).

The best way to understand why and how this works is to consider the case of a deterministic policy first. Removing the records that are not useful for the policy as in R-BRED-DP can not hurt. We thus go back to the simple sample mean case as in R-BRED-DP (all the records are useful). Then instead of sampling T_π elements as in R-BRED-DP, we sample T or T/K times from this portion of S. By expansion lemma, it yields an unbiased estimation of $CTR_\pi(T)$ or $CTR_\pi(T/K)$. Therefore when thinking about it, for a deterministic policy, R-BRED is simply R-BRED-DP with the use of expansion, that is the procedure used in theorem 14. The expansion factor E is just calculated given the sample size to reach: T, T/K or any other value T' depending on our need.

With stochastic policies, S contains an amount of information equivalent to γ^*_π fully labeled records. The classical bootstrap, performed on this equivalent fully labeled dataset would thus sample γ^*_π samples per bootstrap resample. By expansion lemma, we can do more, we can do less and still get an unbiased estimation of $CTR_\pi(T)$ for instance. However contrarily to the case of deterministic policies, this equivalent fully labeled dataset is obviously not directly accessible for we can not remove records without reducing the amount of information in S. When sampling T times from S and using replay we only get to observe a realization of $\widehat{CTR}_\pi(T^{(b)})$, with $\mathbb{E}\,T^{(b)} = \frac{T}{K}$ but that can take any value between 0 and T. Nonetheless nothing prevents us from "stabilizing" $T^{(b)}$ by sampling more or less so as to observe a realization of $\widehat{CTR}_\pi(T^{(b)} = T')$, for any T' and so for each bootstrap resample. This yields an unbiased estimate of $CTR_\pi(T')$ for any T'. Note a small detail however. For R-BRED-DP we conditioned the unbiasedness on the extremely likely but yet slightly uncertain fact that $T_\pi > 0$. Here the unbiasedness is conditioned of the even more likely event that $\gamma^*_\pi > 0$. Furthermore, if we evaluate a randomized policy for which $\forall x \in \mathcal{X}, \forall a \in \{1..K\}, p(\pi(x) = a) > 0$, then R-BRED is unbiased since $p(\gamma^*_\pi(S) = 0 | S \sim F) = 0$. Remark that ε-optimal that we considered earlier is such a policy. A detailed implementation of R-BRED is given in algorithm 15.

Algorithm 15 Rigorous BRED (R-BRED).

Input

- A dataset $S = \{(x_1, a_1, r_1), ..., (x_T, a_T, r_T)\}$ logged by playing a random uniform policy T times against the real world $\mathcal{D}$
- a deterministic policy π

Output: An estimate of $CTR_\pi(T')$, for any T'.

```
for $b \in \{1..B\}$ do
    $t \leftarrow 0$
    while $t < T'$ do
        /* If $\gamma_\pi * = 0$, then this is an infinite loop. */
        Sample $(x, a, r)$ with replacement from $S$
        if $\pi(x) = a$ then
            $t \leftarrow t + 1$
            $r_t^{(b)} \leftarrow r$
        end if
    end while
    $g_\pi^{(b)} = \frac{1}{T'} \sum_{t=1}^{T'} r_t^{(b)}$
    /* Note that nothing prevents Studentization here as in algorithm 13. */
end for
return $\left\{ g_\pi^{(1)}, ..., g_\pi^{(B)} \right\}$
```

Finally, what about precision? As far as the estimation of the mean is concerned, if B (the number of bootstrap resamples) is big enough, BRED is more or less equivalent to RED∞. Thus its precision is in $O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right)$ as well. Γ_π^* will be quantified when analyzing the variance of RED∞. Anyway, more formally we have that:

$$\left|g_\pi - \hat{g}_\pi^{(BRED)}\right| = O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right).$$

Note that this is achieved by all the variants of BRED (R-BRED, R-BRED-DP, S-BRED) as long as B goes to infinity. We also recall that it is the best precision that can be achieved in terms of replay. Better can be done with additional mechanisms but we will see how later.

The precision of the distribution estimation is also easy to determine. Indeed, we showed that at the cost of small modifications, BRED could be turned into a pure sample mean of random variables sampled from F_π (R-BRED). The core idea to exhibit the precision of the bootstrapping procedure is the following. We are interested in the mean of T *i.i.d.* samples from F_π. Provided that π_{LOG} is the random uniform policy, theorem 2 basically says that if we have full access to $F_{\pi_{LOG}}^{(all)}$ (this is equivalent to having an infinitely long dataset), then replay can reproduce F_π *exactly.* Unfortunately we only have T records inside S. Yet we have full access to $F_{\pi_{LOG}}^{\hat{(all)}}$, by simply sampling with replacement from S as many times as needed. Therefore we can estimate F_π without bias using replay. By definition S contains T *i.i.d.* records that contain all together $T/\Gamma*_\pi$ units of information as previously defined. We recall that this information is the inverse of the minimum variance that replay can yield when estimating the mean of F_π with S. By Theorem 14.4-1 in Bishop *et al.* [127], the square root of this variance $(\sqrt{\frac{\Gamma*_\pi}{T}})$ is a high probability bound on the absolute error, which is what we are after. Furthermore, replaying S does not only give access to the mean of $\hat{F}_\pi$ but to $\hat{F}_\pi$ itself. Thus we can also bound the minimum absolute error that can be made on the other cumulants by a quantity of the same order as the one shown for the mean. Consequently since $F_{\pi_{LOG}}^{\hat{(all)}}$ is built by sampling records from the *totality* of S (and thus from the totality of its embedded information), $\hat{F}_\pi$ has a mean within distance in $O\left(\sqrt{\frac{\Gamma*_\pi}{T}}\right)$ of the true mean of F_π. The same thing goes for the other cumulants.

As a consequence the fact that R-BRED is a bootstrap of a sample mean and the expansion lemma yields the following theorem.

Theorem 15. *Under the conditions of the expansion lemma (lemma 1), if $\gamma_\pi^* > 0$ and for* ***any*** *policy π, the procedure R-BRED with an expansion parameter T (i.e. that builds bootstrap resamples of size T) produces an estimation of $CTR_\pi(T)$ - the distribution of an estimator of g_π based on the mean of T real world evaluations. Uniformly on x, this estimation concentrates as follows:*

$$\left|\widehat{CTR}_\pi^{(R-BRED(T))}(S)(x) - CTR_\pi(T)(x)\right| = O_p\left(\frac{\sqrt{\Gamma_\pi^*}}{T}\right).$$

This theorem generalizes to the estimation of $CTR_\pi(T')$ for any T'. R-BRED thus concentrates as follows:

$$\left|\widehat{CTR}_\pi^{(R-BRED(T'))}(S)(x) - CTR_\pi(T')(x)\right| = O_p\left(\frac{\sqrt{\Gamma_\pi^*}}{\sqrt{T.T'}}\right).$$

We recall that $\widehat{CTR}_\pi^{(R-BRED(T))}(S)$ is the CDF estimation given S produced by R-BRED (when stopping at T evaluations). R-BRED(T) is a bootstrap procedure on replay, using an expansion factor of K *on average.* Indeed the stopping criterion (expansion parameter in the theorem) is not the number of sampled records from S but the number of successful evaluation by replay. We can obviously do this for any number of interactions T'.

The $\gamma^*_\pi > 0$ condition is there to remind us of the unavoidable bias entailed by the very unlikely fact that a dataset may not contain information with regard to a policy π. We can quantify how unlikely this is. For a deterministic policy, the probability that there is information is the probability that at least one interaction is simulated by replay. It is exponentially close to 1:

$$P(\gamma^*_\pi > 0) = P(T_\pi > 0) = 1 - \left(\frac{K-1}{K}\right)^T.$$

For a stochastic policy, the probability that the dataset contains information with regard to π is even higher:

$$P(\gamma^*_\pi > 0) = 1 - \sum_{x\in\mathcal{X}} p(x) \left(\frac{1}{K}K - \sum_{a=1}^{K} I\left(p\left(\pi(x) = a\right) > 0\right)\right)^T.$$

It simply means that the probability gets higher when the policy can make different choices given a context. It is even equal to 1 if for any context and any action, $p(\pi(x) = a) > 0$ (*e.g.* ε-optimal or the random uniform policy). Anyway, all this means that with almost absolute certainty, $CTR_\pi(T')$ for any T' can be estimated without bias and with a order of convergence superior to the estimation of g_π.

5.7.6 Discussion on what is estimated

The fact that our various methods all estimate g_π, the bagged estimate, with a precision in $O_p\left(\sqrt{\frac{\Gamma^*_\pi}{T}}\right)$ is settled. We will study Γ in the variance analysis that follows and that will complete the analysis on that point. Nevertheless, our various bootstrapping methods estimate different distributions in various ways. This raises a few questions. The first one is that the bootstrap is usually used to determine confidence intervals on an estimation given some data. Do we achieve that here? How can we interpret the estimates we do output? Also we have introduced R-BRED, a bootstrapping procedure that can rigorously be proved to achieve the Studentized accuracy. Does it mean that BRED should be discarded? Finally, our primary concern was to reduce the effect of time acceleration when evaluating learning algorithm. We start by discussing this major concern.

BRED with learning algorithms

One might think that because we estimate $CTR_\pi(T)$, the distribution of $g_\pi(T)$ (and not T/K), the time acceleration issue is completely solved. In a way this is not entirely wrong. Indeed our problem with learning algorithms was that we could not simulate enough interactions which resulted in a severe bias. Here, we basically showed that by sampling with replacement, it was possible to estimate $F_{\pi_{LOG}}$ so as to sample from it as many times as it is needed to cancel the effect of time acceleration that comes along with the use of replay. This allows us to estimate $CTR_\pi(T)$, the distribution of $g_\pi(T)$. In so doing, we exactly solve the problem we had in section 5.2 but for static policies only in theory. From there we can wonder whether it can be extended to learning algorithms. The answer is not really straightforward. A major step forward is that we showed that, thanks to expansion, any dataset (of reasonable length) can be used to simulate any number of real world interactions, at least in the case of stationary policies. Yet the non parametric bootstrap sampling procedure yields unbiased estimators for classical estimators (that consider each record independently, for which the order does not matter *etc.*). It is clearly not the case for "sequential estimators" for which the processing of one record depends on the previously processed records. In particular, by sampling too much from $\hat{F}_{\pi_{LOG}}$, we may end up overfitting when evaluating a learning algorithm and thus overestimate $g_A(T)$. Nonetheless we can first notice that even for learning algorithm, BRED and R-BRED are asymptotically unbiased. Indeed, if S is infinite, then $\hat{F}_{\pi_{LOG}} = F_{\pi_{LOG}}$. Obviously when S is finite, $\hat{F}_{\pi_{LOG}}$ is a discrete and thus biased version of $F_{\pi_{LOG}}$ and a learning algorithm can use this bias to perform better than it would in reality. This is the exact definition of overfitting. As a consequence, to get an idea of whether or not this method can work we simply have to wonder if $\hat{F}_{\pi_{LOG}}$ is a good approximation of $F_{\pi_{LOG}}$. This may be hard to quantify but we could define a very reasonable rule of thumb. Indeed, BRED and R-BRED become unbiased when S is infinite but it can be close to be unbiased before that. We basically need a good approximation of $R_{a,x}$ for each action a and each context x. Let us postulate that N samples from a distribution is enough to approximate it fairly. N obviously depends on the distribution but we could set it to ten or a few tens to be sure under normality assumption or maybe a little more for Bernoulli or for strange distributions. Then if $T \geq N.|\mathcal{X}|.K^2$, BRED should not be biased too much for $\hat{F}_{\pi_{LOG}}$ should be relatively close to $F_{\pi_{LOG}}$. If $\mathcal{X} = \mathbb{R}$, we could use $T \geq N^2.K^2$ as our rule of thumb. Note that K is squared because we need to see each action a sufficient number of time in each context and because of the rejection mechanism of replay. Do the algorithms we evaluate verify this rule of thumb? In the toy example we consider, we have that $\mathcal{X} = \mathbb{R}^{15}$ and $K = 10$. Thus, T has to be bigger than $N^{15+1}.10^2 \approx 6.10^{22}$ if we take $N = 20$. This is completely unrealistic. Note however that since the features are independent and linear, we could reduce that number in this case. Indeed as we see it on figure 5.8, $T = 10^5$ starts decreasing toward acceptable values for the bias of an evaluation. In the Yahoo! R6B dataset [12], $\mathcal{X} = \{0,1\}^{135}$ and $K \in [20, 60]$ so let us take the mean $K = 40$. Therefore to get a fair evaluation, that is free from overfitting, T should be bigger than 10^{45} which is even more preposterous. We will see in a next section how Jittering tackles this issue in general. However, this is not necessary for non contextual algorithm such as UCB. Indeed, since they do not use the context, we can leave them out of the rule of thumb. Consequently a dataset of length $T = N.K^2 = 2,000$ should do the trick with our toy example. With the Yahoo! dataset we would need $T = N.K^2 = 32,000$ records. This is completely reasonable. Moreover a period of time covered by so few records can be considered stationary. This means that successive regions of the data can be sampled with replacement and yield fair evaluations of UCB, Thompson sampling or even Optimistic Greedy, and so without hiding the underlying evolution through time of the problem. As a consequence, we could consider evaluating bandit algorithms that take into consideration the CTR decay of the news, the differences between night and day *etc.* that are exhibited in section 4.2.3 and in Agarwal *et al.* [103]. Doing so would already be a big improvement compared to classical bandit algorithms.

To really exhibit that BRED is biased even for non contextual algorithms when the horizon is relatively small, we studied its bias empirically. The results are displayed on figure 5.8 and the positive bias is clearly due to overfitting. Nonetheless, this bias clearly gets reasonable for $T > 1000$ and almost negligible for $T > 10,000$ when evaluating a non-contextual algorithm (UCB here). Figure 5.8 also displays the bias of the evaluation of LinUCB. In that case, even for $T = 100,000$, the bias is still greater than 0.01 which is still a lot for a CTR evaluation, hence the need for Jittering so as to reduce it faster.

Besides the addition of Jittering that we will comment later, we have gone as far as we could go for non stationary algorithms. Indeed it is showed in section 3.5.3 that concentration results were impossible to derive in general. Note again however that this statement could be significantly soften for learning algorithms, in particular "all time optimal" algorithms such as UCB which exhibit a very smooth behavior. Nonetheless in this work we do not study them but most of the arguments that we will use in what follows for static policies will allow us to acquire intuition as to how the evaluation of such smooth learning processes could concentrate around their means.

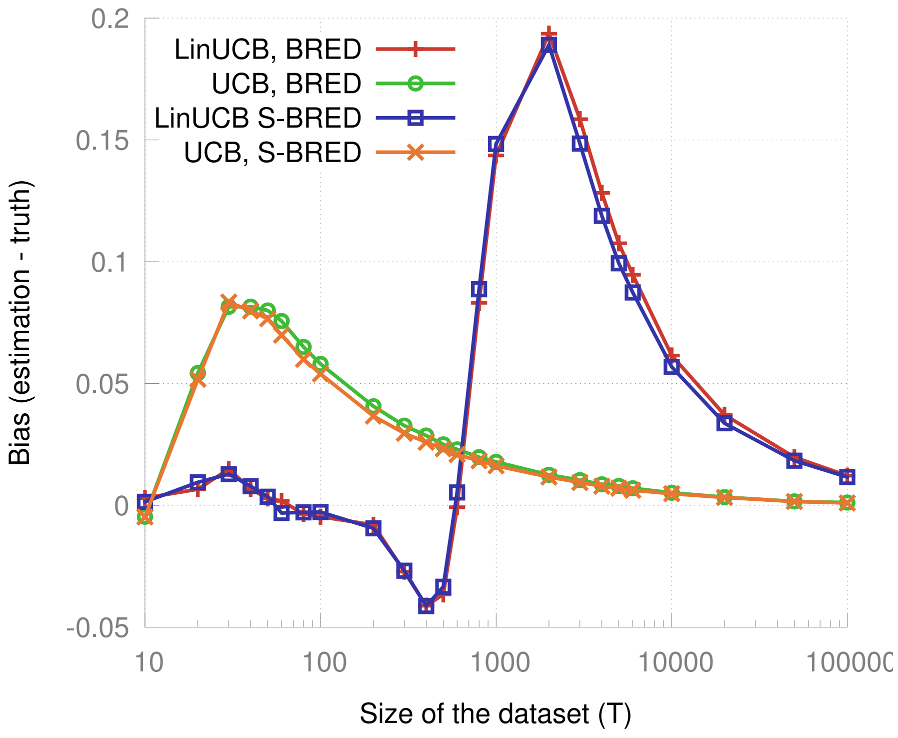


Figure 5.8: Bias of BRED and S-BRED when evaluating UCB and LinUCB over the size of the dataset. Note that this figure gives no information about convergence whatsoever. What is important to keep in mind is that for relatively small datasets, BRED is very positively biased because the evaluated algorithm overfits the data. Then, as more data becomes available the bootstrap estimation of the underlying bandit distribution gets better and better. Consequently the algorithm overfits less and less until having an almost unbiased evaluation. Obviously because what needs to be estimated by bootstrap is much more complex (for non contextual algorithms, the context space does not matter) the process is much slower for contextual algorithms. A less crucial yet mind-tickling detail is that for very small datasets the bias starts by being very low. This is simply because in this case, the algorithm explores a lot and does not take advantage of the fact that it could overfit the data. Then the algorithm explores less and less and its evaluation gets more and more biased. Finally LinUCB's evaluation gets negatively biased right before being affected by overfitting. It is very likely due to the fact that the algorithms learns from duplicates. Indeed it causes the algorithm to believe it has seen slightly less interactions that it really has. This makes LinUCB explore a short while longer with BRED than in reality, hence its underestimation at this moment. We study this further in chapter A.

Consequently, let us focus on static policies again, for which concentration results are available.

Going further with stationary policies

We have already discussed the precision of R-BRED's distribution estimation in $O_p\left(\frac{\sqrt{\Gamma_\pi^*}}{T}\right)$ and nothing more can really be added except for quantifying Γ_π^*. Nonetheless, regardless of the accuracy, what have we estimated with BRED and R-BRED in particular (since we analyzed it more precisely)? We have estimated $CTR_\pi(T)$, the distribution of how π behaves when put online for an amount of time that suits our needs. Note that when talking about distribution, it means that R-BRED allows to compute various properties linked with the distribution: a CDF or more importantly in our context of risk minimization: confidence intervals.

Although all this is very interesting, let us stress one crucial point. The confidence interval estimated by R-BRED is an estimation of how π can deviate from its mean when being put online. This is definitely something that has to be kept in mind when minimizing risk. Nevertheless in no case does this confidence interval computed by R-BRED reflects the error that might have been done on the *estimation* of g_π by R-BRED itself (or by any other means by the way unless the estimation was done online which is precisely what we do not want to need to do). This does not mean that what is computed by BRED is not interesting, on the contrary knowing how much we can deviate from the expectation is highly interesting but we need something else in conjunction. Indeed, usually the bootstrap is about computing confidence intervals on the estimation an estimator produces. Are the distributions outputted by BRED or R-BRED informative with that regard? Actually the answer is clear: no. R-BRED estimates the uncertainty on the online evaluation but says nothing about the uncertainty yielded by the estimation using our dataset S. This is a specificity of our problem. Even if we could estimate g_π with infinite accuracy, we would still be unsure about the result of putting π online since we would only do so once and so for a limited amount of time. Nevertheless, our estimation of g_π is certainly not perfect and having a sense of its uncertainty is at least as important as knowing how it may deviate from its mean when being put online. This is all the more important to be aware of it since a dataset S contains Γ^*_π (with Γ^*_π between 1 and K) less information than T real world interactions. Indeed this causes the expected squared error (variance if it is unbiased) on the estimation of g_π to be Γ^*_π greater than the expected squared deviation of π from its mean when being put online. Thus it would not be ridiculous to say that having an estimation of $CTR^{(estimator)}_\pi(T)$, the distribution of the per trial payoff outputted by an estimator when computed on T records, is Γ^*_π more important than having an estimation of $CTR_\pi(T)$.

Fortunately, we have already partially addressed this issue here and we are on our way to completing the process. Because we stabilize $T^{(b)}$ with R-BRED, it really estimates $CTR_\pi(T)$. Nevertheless, the first version of BRED is not altered in that fashion. Therefore it actually estimates $CTR^{(replay)}_\pi(K.T)$, that is the distribution of replay computed on a dataset that consists in $K.T$ *distinct* records. Indeed, BRED samples $K.T$ independent records from $\hat{F}_{\pi_{LOG}}$ and performs replay on them at each bootstrap resample, it is thus bootstrapping replay. By expansion Lemma, this is an unbiased estimation of the distribution of replay on $K.T$ independent records sampled from $F_{\pi_{LOG}}$. By dropping expansion, we even get an estimation of $CTR^{(replay)}_\pi(T)$. Contrarily to $replay(K.T)$, that we can obviously not compute in practice with a dataset of size T, we can compute $replay(T)$. Therefore the estimation provided by BRED, performed without expansion is informative with regard to the estimation of the uncertainty on our estimation of g_π. Yet, is an estimation of $CTR^{(replay)}_\pi(T)$ really what we are looking for? The answer is no. Indeed, we checked empirically in section 5.5.2 that replay did not always managed to drag the maximum amount of information γ^*_π out of the data. A precise quantification of the amount of information it manages to acquire will be given in section 5.8. We even exhibited that the bagged estimate provided by BRED was better in many cases. Nonetheless, we know intuitively that BRED can not drag less information than replay out of the data. Thus the estimation of uncertainty provided by BRED, which is an estimation of $CTR^{(replay)}_\pi(T)$, can not underestimate the real uncertainty we have on g_π which would be dramatic when minimizing risk. Therefore it can act as a lower bound (in expectation) on that uncertainty which already informative. Nonetheless in no case will this estimation reflect the actual accuracy with which g_π is estimated by BRED's bagged estimate, unless the evaluated policy is deterministic, in which case replay is a good estimator. As a consequence, we need another tool. Before studying it, although bootstrapping replay is not something we recommend in practice for static policies, let us briefly talk about its accuracy regarding the evaluation of $CTR^{(replay)}_\pi(T)$. We will not prove it but we can conjecture that since replay is very close to a sample mean, we have that:

$$\begin{aligned}\widehat{CTR}_\pi^{(BRED,E=K)}(S) &= CTR_\pi^{(replay)}(K.T) + O_p\left(\frac{\sqrt{\Gamma_\pi^*}}{T}\right)\\ \widehat{CTR}_\pi^{(BRED,E=1)}(S) &= CTR_\pi^{(replay)}(T) + O_p\left(\frac{\sqrt{K.\Gamma_\pi^*}}{T}\right).\end{aligned}$$

Notice that Γ_π^* appears instead of K, improving convergence if it is smaller. This is because although replay performed on one individual bootstrap resample is not capable of dragging all the available information, the overall procedure will as B goes to infinity, hence the dependence in Γ_π^*. More formally this is because the sample cumulants of S are within distance in $O\left(\sqrt{\Gamma_\pi^*/T}\right)$ of the cumulants of F_π. Furthermore note that we gave arguments in the previous section supporting that proving this result would not be straightforward. One such argument is that replay is not a classical estimator and can even be assimilated to a mean on a sample of variable size, which is not easy to analyze in this context. However, this is not a big deal since we will not use this estimator in practice for static policies. In addition of not being what we are looking for, that is an estimation of the exact uncertainty there exists on our estimation of g_π, BRED turns out to be complicated to analyze. This is one additional reason not to use it with static policies. Note anyway that since replay is a sum it necessarily verifies the classical smoothness assumptions. Therefore at the very least we have a result that looses a factor $\frac{1}{\sqrt{T}}$:

$$\begin{aligned}\widehat{CTR}_\pi^{(BRED,E=K)}(S) &= CTR_\pi^{(replay)}(K.T) + O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right)\\ \widehat{CTR}_\pi^{(BRED,E=1)}(S) &= CTR_\pi^{(replay)}(T) + O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right).\end{aligned}$$

Again, Γ_π^* appears and not K because B goes to infinity. This is the reason why BRED would still be better than a Normal approximation on the sum resulting from one computation of replay. Indeed, such a sum would consist in T/K elements in average so a normal approximation would be in roughly $O_p\left(\sqrt{\frac{K}{T}}\right)$. Anyway, although it does not really matter, we do believe that BRED could take advantage of Studentization similarly to the sample mean and achieve the first conjectured precision.

Before dealing with the general case, let us focus for now on the simpler case of deterministic policies for which simply performing replay is optimal. We will see later that it is not the case but according to our current definition of Γ_π^*, it is. In this case, we have in fact already managed to estimate a distribution $CTR_\pi^{(estimator-DP^*)}(T)$, where $estimator - DP^*$ is an estimator that acquires an amount of knowledge $\gamma_\pi^* = T_\pi$. Also we proved that we did so with the Studentized accuracy. When did we do so? We saw that BRED, without expansion was meant to estimate $CTR_\pi^{(replay)}(T)$, which is what we are after. Nevertheless we did not prove that we could reach the Studentized accuracy neither for deterministic nor stochastic policies. Indeed in both cases, $T^{(b)}$ is a random variable. R-BRED provably achieves the accuracy we are after. Yet it estimates $CTR_\pi(T')$ for any T', which is not what we want. Our last estimator R-BRED-DP does exactly what we want. This is however not completely straightforward since $CTR_\pi^{(replay)}(T)$, which is what we are after is not equal to $CTR_\pi(T_\pi)$, the distribution estimated by R-BRED-DP with the Studentized accuracy. Indeed, even when $T_\pi = \frac{T}{K}$, replay is a slightly biased estimator of g_π (in $O_p\left(e^{-T}\right)$) and has a slightly higher variance than $g_\pi(T/K)$ (in $O_p\left(T^{-2}\right)$). See chapter 3 for proofs. Knowing this, how can an estimator of $CTR_\pi(T_\pi)$, the distribution of the per trial payoff of π when played in the real world T_π times be relevant with respect to the quality of our offline estimation? Simply because it is also the distribution of replay given that it is played on a dataset containing T_π relevant records, which is exactly what S is. Indeed, given that a certain number of interactions are simulated by replay, their sequence is indistinguishable from a sequence of interaction with the real world. We could consequently write that $CTR_\pi(T_\pi) = CTR_\pi^{(replay)}(info(S) = T_\pi)$. Thus $CTR_\pi(T_\pi)$ is highly relevant to compute confidence intervals on our estimation using replay on S. Notice the difference of philosophy compared to BRED. When using BRED to quantify the uncertainty we have on an estimation of g_π, we consider that we obtained it by computing replay on a dataset of size T. When using R-BRED-DP, we consider that we obtained it by computing replay on a dataset containing T_π relevant records. Both points of view are correct but the latter is definitely more informative with regard to the quantification of the uncertainty on $\hat{g}_\pi$. Therefore R-BRED-DP should yield confidence intervals that reflect more precisely the uncertainty on our estimation of g_π.

Finally, it is clear that for the general case and for stochastic policies in particular, we do not have an estimator of $CTR_\pi^{(estimator^*)}(T)$ such that:

$$Var\left(\hat{g}_\pi^{(estimator^*)}(T)\right) = \frac{\Gamma_\pi^*}{T} Var\left(\vec{r}[\pi(x)]\right) \ .$$

In other words we do not have an estimator of the distribution (*e.g.* via bootstrap) of an estimator such as BRED that drags the quantity of knowledge γ_π^* out of the data. Unfortunately, the technique used for deterministic policies to filter out useless records cannot work since things are not binary anymore. With deterministic policies, each record contains an amount of information equivalent to either 0 or 1 fully labeled records. With stochastic policies, most of the records actually contain information equivalent to a strictly positive fraction of information. Therefore filtering some of them would not help. One could think of designing a technique analogous to R-BRED-DP but that would work for any policy using a remark we made in the previous section. R-BRED-DP estimates $CTR_\pi(T_\pi)$. We also pointed out that R-BRED, when used with a stabilized number of interactions equal to T_π, was estimating the same thing as R-BRED-DP with the same accuracy. For a deterministic policy, we have that $T_\pi = \gamma_\pi^*$. This is not the case for a stochastic policy and we certainly have that $T_\pi < \gamma_\pi^*$. Nonetheless, using R-BRED with a number of interactions equal to γ_π^* would also yield an estimation of $CTR_\pi(\gamma_\pi^*)$. This is however a very bad idea for many reasons.

1. γ_π^* might to be an integer. Rounding it up in order to define a number of interaction to plug into R-BRED would definitely alter the precision of our quantification of the uncertainty on g_π. This is however a minor concern compared to the following ones.

2. The process is easy when π is deterministic since using T_π, an easily accessible quantity, is enough. On the contrary when the policy is stochastic we do not know γ_π^*. We would have to estimate it. It is strongly correlated with the variance of π, which can not be estimated with a precision superior to $O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right)$. Therefore reaching the Studentized accuracy for an estimation of $CTR_\pi(\gamma_\pi^*)$ based on that estimation would be impossible.

3. The last reason is actually the main one. Let us consider that γ_π^* is known and is an integer. We would therefore have that:

$$\widehat{CTR}_\pi^{(R-BRED(\gamma_\pi^*))}(S) - CTR_\pi(\gamma_\pi^*) = O_p\left(\frac{\Gamma_\pi^*}{T}\right) .$$

   Yet we are not in fact interested in the distribution of γ_π^* interactions with the real world but in the distribution of the estimator of g_π, given that S contains information equivalent to γ_π^* (or given a dataset of size T which we already established to be OK but less accurate). If by definition they have a similar mean and variance, nothing ensures that they have a similar distribution. Actually we will show more formally later that they have not. Intuitively it makes sense since instead of using γ_π^* fully labeled records, we only have at hand the same amount of information but spread among T records. However since they are sums of things (which is shown in the next section) and that they have the same mean and variance, their normal limits are equal. Consequently we would have:

$$\widehat{CTR}_\pi^{(R-BRED(\gamma_\pi^*))}(S) - CTR_\pi(\gamma_\pi^*) = O_p\left(\sqrt{\frac{\Gamma_\pi^*}{T}}\right),$$

but nothing better, even though γ_π^* is assumed to be known, which is already pretty unrealistic.

As a conclusion, what we need is to be able to bootstrap an estimator that drags all the knowledge there is. There are no real shortcuts when the policy is not deterministic. Fortunately we exhibited a few of them: BRED, R-BRED, S-BRED and RED∞. The bagged estimator of the first three could be bootstrapped in practice without problem. Yet they might be complicated to analyze. Moreover it would be rather unsettling to bootstrap a bootstrapping procedure, especially considering that we are only interested in a small part of what the procedure outputs: the bagged estimate. This is why in the next section we analyze the bootstrap of RED∞ and show that it enables to reach the Studentized accuracy thanks to another equivalence with a basic sample mean.

5.7.7 Analysis of the Bootstrap of RED∞

As we already mentioned it, RED∞ achieves the same accuracy exhibited for BRED but without needing to introduce bootstrapping which is not usually meant for variance reduction. Thus it enables us to use bootstrapping without expansion to estimate without bias the uncertainty on our estimation of g_π. As a nice side effect, this separation also allows to study how bootstrapping works as a distribution estimation technique more easily. Let us do so here.

At first sight, RED∞ does not seem like the mean of *i.i.d.* random variables which is what we want our estimator to be reduced to, so as to justify that it can be expanded in Edgeworth's sense. Indeed, RED∞ loops over the same records many times and averages them so the elements of the overall sum are definitely not independent. Nevertheless, since the loop is infinite in theory, we can reduce the estimator to its limit in E, the expansion factor. Using $\pi_e(x_t)$ to denote the choice made by π on context x_t at the expansion index e. The reduction simply goes as follows:

$$\begin{aligned}
\hat{g}_\pi^{(RED\infty)} &= \frac{\sum_{e=1}^{E}\sum_{t=1}^{T} I(\pi_e(x_t)=a_t).r_t}{\sum_{e=1}^{E}\sum_{t=1}^{T} I(\pi_e(x_t)=a_t)} = \frac{\sum_{t=1}^{T}\sum_{e=1}^{E} I(\pi_e(x_t)=a_t).r_t}{\sum_{t=1}^{T}\sum_{e=1}^{E} I\left(\pi_e(x_t)=a_t\right)} \\
&= \frac{\sum_{t=1}^{T} E\,\mathbb{E}_S\left(I(\pi(x_t)=a_t).r_t\right)}{\sum_{t=1}^{T} E\,\mathbb{E}_S\left[I\left(\pi\left(x_t\right)=a_t\right)\right]} = \frac{\sum_{t=1}^{T} p\left(\pi\left(x_t\right)=a_t\right).r_t}{\sum_{t=1}^{T} p\left(\pi\left(x_t\right)=a_t\right)}.
\end{aligned}$$

Similarly, we can get the same kind of result for RED*∞, the unbiased version of RED∞ based on replay*:

$$\begin{aligned}
\hat{g}_\pi^{(RED^*\infty)} &= \frac{K}{ET}\sum_{e=1}^{E}\sum_{t=1}^{T} I\left(\pi_e(x_t)=a_t\right).r_t \\
&= \frac{K}{T}\sum_{t=1}^{T} p\left(\pi(x_t)=a_t\right).r_t\ .
\end{aligned}$$

First note that as long as the policy we want to evaluate is fully known, that is we know the probability of each action to be chosen given any context, the reduction we just exhibited can be computed. Therefore the replay procedure is not necessary anymore which can save a lot of resources. This is non negligible especially if we are interested in bootstrapping RED∞ as we exhibited that a value for B of at least $10,000$ is necessary (see fig. 5.5). In some cases however, if the policy is a black box or a non-deterministic model from which we can not easily extract probabilities of performing actions, the replay procedure is the only option. Anyway, regardless of how they are computed, RED∞ and RED*∞ are equivalent to their respective reduced forms if E, the expansion factor, goes to infinity. Therefore we use these reduced forms in our analysis.

What does it mean about bootstrapping RED∞? The reduced RED∞ is some kind of weighted average which may be tricky to analyze in the context of bootstrapping, similarly to the kind of problems we met while analyzing replay's variance in chapter 3. The reduced form of RED*∞, being a weighted sum, is much more friendly as far as bootstrap analysis is concerned. Why is that? Simply because it can be reduced to a sample mean of T independent draws from some distribution. Indeed, let us define the following quantity:

$$s_t \stackrel{\text{def}}{=} K.p\left(\pi\left(x_t\right)=a_t\right).r_t\ .$$

RED*∞ is in fact the mean of a sample $s_1,...,s_T$, computed from all the tuples (x_t,a_t,r_t) contained in S. Therefore RED*∞ is simply the mean of a sample containing T realizations of the random variable s generated as follows:

1. $(x,a,r) \sim F^{(all)}_{\pi_{LOG}}$
2. $s \stackrel{\text{def}}{=} K.r.p(\pi(x)=a)$.

We may denote by $F_{\pi_{LOG},\pi}$ the distribution of s.

Note that γ^*_π is the amount of information extracted by RED∞, the maximum information estimator based on replay. Thus by definition, the variance of RED∞ is equal to $\gamma^*_\pi.Var\left(\vec{r}[\pi(x)]\right)$. The variance of RED*$\infty$ is very likely to be slightly higher. In what follows however we will study γ^+_π, the amount of information extracted by RED*∞. This will yield a much simpler and more intuitive analysis for two reasons: (i) RED*∞ is unbiased and (ii) has a simpler reduced form. Also, although we do recommend using RED∞ in practice, the analysis based on RED*∞ will bring all the necessary intuition to enable the correct use of both estimators in practice.

The knowledge of the variance of RED*∞ provides us with a high probability bound on the estimation of the mean g_π by Theorem 14.4-1 in Bishop *et al.* [127]. This error bound is as follows:

$$\left|\hat{g}^{(RED^*\infty)}_\pi(S)-g_\pi\right| = O_p\left(\sqrt{\gamma^+_\pi.Var\left(\vec{r}[\pi(x)]\right)}\right) = O_p\left(\sqrt{\frac{\Gamma^+_\pi.Var\left(\vec{r}[\pi(x)]\right)}{T}}\right) = O_p\left(\sqrt{\frac{\Gamma^+_\pi}{T}}\right),$$

from which we remove the variance for clarity. Since RED*∞ is a mean of independent random variables (s), error bounds of the same order are necessarily valid for the other sample cumulants provided that they exist. Consequently since bootstrapping RED*∞ is equivalent to bootstrapping a sample mean, we can follow exactly the proof from the literature. This means that we can write the CDF of $CTR^{(RED^*\infty)}_\pi(T)$ (resp. $\widehat{CTR}^{(B-RED^*\infty)}(S)$, where B-RED*$\infty$ is the bootstrap estimator of $CTR^{(RED^*\infty)}_\pi(T)$ based on S) as an Edgeworth expansion in $\frac{1}{\sqrt{T}}$ and the cumulants of $F_{\pi_{LOG},\pi}$ (resp. the cumulants of $\hat{F}_{\pi_{LOG},\pi}$, which are simply the sample cumulants of the sample $s_1,...,s_T$). Because the cumulants and their estimators are within distance in $O\left(\sqrt{\Gamma^+_\pi/T}\right)$, the difference of the two aforementioned expansions simplifies. It yields the following theorem:

Theorem 16. *Under the conditions of the expansion lemma (lemma 1) and for* **any** *policy π, the bootstrap of the estimator RED*∞ concentrates as follows:*

$$\left|CTR_{\pi}^{(RED^*\infty)}(T) - \widehat{CTR}^{(B-RED^*\infty)}(S)\right| = O_p\left(\frac{\sqrt{\Gamma_{\pi}^{+}}}{T}\right)$$

and so uniformly on x.

Note that bootstrap theory would in fact ignore Γ_{π}^{+} the same way as we ignore $Var\left(\vec{r}[\pi(x)]\right)$ that could appear in the square root. It would result in a bound in $O_p(1/T)$. Yet removing it would be like ignoring the effect time acceleration has on the quality of our estimators, which we already know to be really problematic in practice. This result comes with intuition with that regard which is what we wanted in the first place.

Finally remark that this estimated distribution gives no information on $CTR_{\pi}(T')$, and so for any T'. It is rather obvious given the weighted form of s but it can be checked formally that the moments of F_{π} are not equal to the moment of $F_{\pi_{LOG},\pi}$. The moments of F_{π} are simply:

$$\mathop{\mathbb{E}}_{F_\pi} r^{\tau} = \sum_{x\in\mathcal{X}} p(x) \sum_{a=1}^{K} p(\pi(x)=a)\,\mathbb{E}\left(r^{\tau}|r\sim\mathcal{R}_{a,x}\right) \ .$$

The moments of $F_{\pi_{LOG},\pi}$ are as follows:

$$\begin{aligned}\mathop{\mathbb{E}}_{F_{\pi_{LOG},\pi}} s^{\tau} &= \mathbb{E}\left[\left(K.p\left(\pi(x)=a\right).r\right)^{\tau}|(x,a,r)\sim\mathcal{F}_{\pi_{LOG}}^{(all)}\right] \\ &= K^{\tau}\sum_{x\in\mathcal{X}} p(x)\sum_{a=1}^{K} p(\pi_{LOG}(x)=a)\,\mathbb{E}\left[p\left(\pi(x)=a\right)^{\tau} r^{\tau}|r\sim\mathcal{R}_{a,x}\right] \\ &= K^{\tau-1}\sum_{x\in\mathcal{X}} p(x)\sum_{a=1}^{K} p\left(\pi(x)=a\right)^{\tau}\,\mathbb{E}\left[r^{\tau}|r\sim\mathcal{R}_{a,x}\right] \ .\end{aligned}$$

Obviously, the first moments are equal, making RED*∞ an unbiased estimator of g_{π}. Also, because it is how $\gamma_{\pi}+$ is defined, the variance of the mean on a sample of size $\gamma_{\pi}+$ will be equal to the variance of RED*∞. Nevertheless, the third and fourth moments, which are needed to enable the bootstrap to beat the normal approximation, are clearly different.

To conclude on this analysis of the bootstrap of replay, we were able to show two major results. The first one is our capacity to estimate $CTR_{\pi}(T)$, the distribution of the per trial payoff of π after T interactions with the real world, and so for any T using R-BRED. In addition we showed that this could be done with an accuracy superior to a classical normal approximation. Nonetheless this estimation was driven by the need to solve time acceleration for learning algorithms. In the process we lost the essence of bootstrapping that is getting a sense of the uncertainty on the bootstrapped estimator. This was solved by studying the bootstrap of another estimator, RED∞. This bootstrap was also proved to reach an accuracy superior to the one of a normal approximation. All the results from this section and the purposes of the various bootstrap estimators are recapped in table 5.1.

5.8 Variance analysis of RED∞

So far we have analyzed the bootstrap of various estimators of g_{π}. The only missing piece is the quantification of Γ_{π}^{*} (which is slightly lower than Γ_{π}^{+}). We have already pointed out than Γ_{π} could change from one policy to another. In this section we will precisely characterize how

Table 5.1: Recap of the various bootstrapping techniques we developed for stationary policies, what they estimate and with what accuracy. All the results are true with high probability ($O_p(...)$) and for any E such that $E.T$ (or $E.T_\pi$ for R-BRED-DP) is an integer. Note however that expansion is only helpful to extrapolate what would happen in the real world or with datasets of size $E.T$. This was led by the desire to evaluate learning algorithms on more data. With stationary policies it is only interesting with R-BRED that estimates what happens in the real world but is partially hidden by the stabilization step. Indeed with R-BRED, the count of the number of records happens after the use of replay whereas with all the other bootstrap estimators, it happens beforehand. Yet in any case expansion is possible so we put it here for the sake of thoroughness. Furthermore, one should be aware that when bootstrapping RED∞ or RED*∞, the bootstrap expansion quantified by E here is not the same as the infinite expansion performed by the estimator.

Estimator	What is estimated	Accuracy of the estimation	
		Not Stu.	Studentized
BRED	$CTR_\pi^{(replay)}(E.T)$	$\sqrt{\frac{\Gamma_\pi^*}{T}}$	**Conjecture:** $\frac{\sqrt{\Gamma_\pi^*}}{T\sqrt{E}}$ *Very likely because replay≈sample mean*
R-BRED-DP (deterministic π only)	$CTR_\pi(E.T_\pi) =$	$\sqrt{\frac{\Gamma_\pi^*}{T}}$	$\frac{1}{T_\pi\sqrt{E}} = \frac{\Gamma_\pi^*}{T\sqrt{E}}$
R-BRED	$CTR_\pi(E.T)$	$\sqrt{\frac{\Gamma_\pi^*}{T}}$	$\frac{\sqrt{\Gamma_\pi^*}}{T\sqrt{E}}$
S-BRED	Nothing. *It is meant to estimate g_A for learning algorithms.*	-	-
Bootstrap of RED*∞	$CTR_\pi^{(RED^*\infty)}(E.T)$	$\sqrt{\frac{\Gamma_\pi^+}{T}}$	$\frac{\sqrt{\Gamma_\pi^+}}{T\sqrt{E}}$
Bootstrap of RED∞	$\begin{aligned} & CTR_\pi^{(RED\infty)}(E.T) \\ = \ & CTR_\pi^{(BRED)}(E.T) \\ = \ & CTR_\pi^{(S-BRED)}(E.T) \\ = \ & ... \end{aligned}$	$\sqrt{\frac{\Gamma_\pi^*}{T}}$	**Conjecture:** $\frac{\sqrt{\Gamma_\pi^*}}{T\sqrt{E}}$ *Very likely given the similarity with the estimator RED*∞.*

and why this happens. To do so, let us recall that we defined the information contained in a dataset as the inverse of the variance:

$$\gamma_\pi^* = Var(\hat{g}_\pi^{(RED\infty)}|S)^{-1}.Var\left(\vec{r}[\pi(x)]\right).$$

Indeed, we highlighted that RED∞ was capable of dragging all the knowledge there is from S (in the sense of replay). The information characterized by γ_π^* can be seen as a number of fully labeled records. Also the time acceleration factor that appears in all the previous results simply depends on the information acquired by RED∞:

$$\Gamma_\pi^* = T/\gamma_\pi^*.$$

Anyway, the key of this analysis is in fact the variance of the RED∞ algorithm given a dataset S. Consequently it is what we will be interested in this section. Although bootstrapping is not going to be studied any further in itself here, this analysis will complete our understanding of its behavior.

In addition, we will take advantage of this analysis to go much further than simply completing the previous one. Besides the variance analysis of RED∞, this section carries two other contributions:

1. The first one is one additional step toward our primary purpose: the evaluation of learning algorithms. We gave intuition in the previous section justifying that without exhibiting variance reduction properties, estimating $F_{\pi_{LOG}}$ combined with expansion was a way to simulate more than T/K interactions and consequently reduce time acceleration. This section will also justify that expansion can also yield a reduction of variance. Although we will only prove things for static policies, we will provide intuition justifying that learning algorithms are able to take advantage of the same mechanisms.

2. The second contribution will be the application of the evaluation of static policies using replay techniques to recommendation in general, and not only dynamic recommendation. Indeed we will see that using a procedure that we call *iterative improvements*, we should be able to evaluate a broad variety of recommender systems using replay. After the study provided in this section, we will see that we actually have all the tools to do so. Such an achievement would be very interesting given that the literature as well as basically everyone in the community assumes that evaluating recommendation in general can only be done via indirect measures of performance such as rating prediction, ranking and so on. Such metrics only partially reflect the performance of the system and can be misleading. This is discussed in chapter 1 and by Herlocker *et al.* [8] for instance. Here we provide a global procedure to design a recommender system, either from scratch or from an existing system that is directed by the maximization of a real world metric: the CTR or any other online measurable metric (scrolling, time spent on a page or seeing a video, conversion rates *etc.*). We will see however that the major drawback of this technique is that it requires data from the actual system. Moreover it cannot be performed on the standard datasets available online. As such this method is not directed toward academic researchers but toward people who are willing to improve an application they own and can tamper with.

This section is organized as follows. We first describe the procedure called *iterative improvements* rather briefly, simply to have in mind what we are interested in while studying the variance of RED. We will see that a setting slightly more general than what was considered so far is needed. Then we study the properties of RED in this setting (variance and bias) and compare it with replay's. Based on this analysis, we then study a simple case to see how iterative improvement could be performed. Finally, we draw a larger picture in which the iterative improvements procedure can be used to build a recommender system in many contexts and to optimize it via an accurate estimation of the CTR. The complete procedure is rather complex, allows to minimize risk easily but is not really studied in itself here. Yet we argue that this chapter already offers solutions to almost all the problem it raises and a first working solution could be implemented from there. Finally, we go back to learning algorithms and justify thanks to intuition based on the variance analysis how BRED in fact brings both bias and variance reduction for their evaluation.

5.8.1 Iterative improvements

It is more or less admitted in the community that CTR metrics and replay methodologies in particular such as the ones we are studying here are not usable with classical recommendation. Indeed, it is clear that with a million items, online learning methods which replay was meant to evaluate are not really an option. Moreover even a static recommender system would be impossible to evaluate accurately at a convergence rate in $O\left(\frac{K}{T}\right)$, K being the number of items. Nonetheless, we argue that if the process of evaluating and building a recommender system is considered as parts of a sort of iterative optimization process, then this is not true any longer. Replay methodologies actually become a credible option. In this work we only lay the foundations of a general procedure, without really investigating its practical behavior. We however exhibit almost all the tools needed to make it work.

This methods targets recommendation applications in which learning online what to recommend is considered too difficult. Several factors can be the cause of that such as a great number of items, very sparse rewards or rewards coming from a very skewed distribution. Rather at time t we will try to build the best recommender system we can with the information we have. Then we will use the data it logs to try to improve it using replay methodologies. By doing so we can reach a much better convergence than $O\left(\sqrt{K/T}\right)$.

Indeed it is intuitive that without any assumption about the evaluated policy π nor the data, the best logging policy π_{LOG} we can find is the uniformly random one. This is the kind of data that the academic world is after so as to produce broad experiments on many policies/algorithms with the same ensured accuracy. Yet in practice, people who are trying to improve their own recommender system may have different requirements. They may want to improve it step by step, making sure (or with minimum risk) that the next version put online is better than the former. In such a process of iterative improvements, mentioned for instance by Bottou *et al.* [98] (Microsoft), we do have strong assumptions on π: we know that it will be somewhat similar to the former policy. In this case a logging policy that is close to π will allow much better converge rates, close to the optimal rate in $O(1/\sqrt{T})$. Thus one can use the current policy as the logging policy of the replay evaluations of the next one and expect very nice results. Nonetheless, to enable evaluation, one has to randomize the current policy so that it allows future replay. This can be done by either introducing random exploration with small probability or using a *softmax* function. As a reminder, a softmax function draws an action with a probability proportional to its estimated click probability. Bottou *et al.* [98] exhibited that using the first simple technique, the CTR of the current policy was not reduced very much and allowed fair evaluations in the context of ad display within the search engine Bing.

One can actually take a very simple example from the Yahoo! R6B dataset [12] used for the challenge we organized and studied in chapter 4. The best policy we found has a CTR of 9%. The random policy has a CTR of 3%. Consequently, a policy choosing the best action with probability $1-\varepsilon$ and an action at random with probability ε, with $\varepsilon = 1\%$ has a CTR of $0.99 * 9 + 0.01 * 3 = 8.94\%$. With $\varepsilon = 5\%$, which may already be quite a lot, we get a policy of CTR equal to 8.7%. Obviously the right amount of "exploration" depends on K (the number of items) and T (the traffic). In any case, given the small gap between a good policy and a random recommender system, the cost of exploration is not as huge as it might have seemed.

We had mentioned in section 3.6.1 that replay could be performed when the logging policy was stochastic and had a non-zero probability of choosing each action given any context. It is also possible when the logging policy is deterministic, non-contextual and performs each action a sufficient number of times. For simplicity we will consider the first case but it is interesting to be aware that exploration can be done in many ways and does not need to be random. To allow an unbiased estimation, for each record, one has to re-normalize the reward by the inverse of probability that this record was produced by the logging policy. More formally we multiply r_t by $p\left(\pi_{LOG}(x_t) = a_t\right)^{-1}$. That being said, the reason why the logging policy has to be known is obvious (it can also be inferred [97], but this unavoidably reduces the accuracy of the evaluation). Moreover given any context, the logging policy has to have a non zero probability of performing any action our evaluated policy may perform. Indeed we divide be this quantity to make our estimation. Yet note that if we know in advance that the policies we aim at evaluating will not perform a given action in a given context, there is no need to explore it while logging the data.

It is with this particular procedure in mind that we also consider a logging policy different from the uniform one in the following analysis. This will not prevent us from discussing the random uniform case while being more general. More importantly, this will allow us to discuss the iterative improvements procedure.

5.8.2 Theoretical analysis

What we want to exhibit formally is that RED∞ does provide a significant improvement in terms of accuracy compared to replay for static policies (that are not deterministic). We also want to quantify this improvement with respect to the difference between the logging policy and the evaluated policy for this is what is interesting in the context of iterative improvements of a recommender system. To simplify and therefore clarify the analysis, we will consider the replay* method and RED*∞. We will do so for two reasons. First because we studied the bootstrap of RED*∞ and highlighted that provided that it was Studentized, it was as accurate as it can get. The main reason however is that as this was exhibited in chapter 3 and section 5.7.7, the unbiased estimators marked by a star are simpler to analyze. One may argue that they are also less accurate. Yet the difference is minor, especially when T is relatively big. Also they work similarly and will be affected by the same variables. Consequently, although the results for the biased but less variate estimators would be slightly different (slightly better convergence, different constants, more complicated formulas...), all the *intuition* that will come out of this analysis, which is after all what we are really after, will be directly applicable to RED∞ and replay.

Unbiasedness

First of all, the following preliminary results assert the unbiasedness of both replay* and RED*∞ when the logging policy is not randomly uniform as long as it respects the aforementioned conditions. Obviously replay and RED∞ are biased. Nonetheless the expression of that bias is less friendly than when the logging policy is random uniform. Indeed, it depends on the probability of the fact that no interactions are simulated by replay. RED∞ intuitively have a smaller bias for stochastic policy for expanding the data infinitely reduces the probability of not simulating a single interaction. Furthermore, the RED∞ can even be unbiased if for any context, all actions have a non zero probability of being played. Yet we are here interested in unbiased estimators: replay* and RED*∞.

Theorem 17. *For any contextual bandit distribution $\mathcal{D}$, for any policy π, for any stochastic logging policy π_{LOG} such that for any context x and any action a, we have that $p(\pi(x) = a) > 0 \Rightarrow p(\pi_{LOG}(x) = a) > 0$, for any dataset S built by T interactions of π_{LOG} with $\mathcal{D}$ we have that the estimator provided by the replay* method:*

$$\hat{g}_\pi^{(replay*)} = \frac{1}{T}\sum_{t=1}^{T} V_t.\frac{r_t}{p(\pi_{LOG}(x_t) = a_t)}$$

is an unbiased estimator of the expected per-trial reward of π: g_π.

Note as a reminder that V_t, introduced in chapter 3 indicates whether π chooses the same action as π_{LOG} at step t. The expectation of V_t is $p(\pi_{LOG}(x_t) = a_t)$, hence the normalization factor that was set to K with a uniform logging policy.

This result is fairly intuitive and is proved in appendix (G.2). From there it is straightforward to show that RED*∞ is unbiased, even if it is not expanded infinitely.

Corollary 18. *Under the same conditions as theorem 17, the estimator of g_π provided by RED*∞:*

$$\hat{g}_\pi^{(RED^*\infty)} = \frac{1}{T.E}\sum_{e=1}^{E}\sum_{t=1}^{T} V_{t,e}.\frac{r_t}{p(\pi_{LOG}(x_t) = a_t)}$$

is unbiased for any expansion factor E, finite or infinite.

Proof. Indeed, RED*∞ can be seen as averaging E independent realizations of the estimator replay* given the data, which is a random variable in the case where π is stochastic. In the simpler case where π, is deterministic, then $\hat{g}_\pi^{(RED^*\infty)} = \hat{g}_\pi^{(replay*)}$ for any dataset S. □

Here we proved that the estimation of g_π was unbiased. We recall that for replay, when the logging policy was random uniform we had another, stronger result stating that given that a sequence of rewards was simulated by replay, it was indistinguishable from a sequence acquired by playing π in the real world theorem 2. It is straightforward that by weighting the rewards here, we lose that property. This can be proved by a very simple example. Let us consider a problem with two actions, yielding Bernoulli rewards. Let us consider a logging policy that plays action 1 with probability 0.2 and action 2 with probability 0.8. While evaluating any policy, each simulated action 1 will yield a reward of either 0 or $\frac{1}{0.2} = 5$. Each simulated action 2 will yield a reward of either 0 or $\frac{1}{0.8} = 1.25$. This is necessary to obtain an unbiased estimator of g_π. Yet a sequence of reward simulated by replay will not be a sequence of 0 and 1 as in the real world. Obviously this involves a bias in all the other moments, and in particular in the variance. This bias can be computed analytically, similarly to what was done for RED*inf in the previous section. Also, similarly to the correction made to get an unbiased estimator of the mean, we can make corrections to get unbiased estimators of all the moments. This is however much more problematic for learning algorithms. Indeed although the corrections would be made during the evaluation, the algorithm would learn on a biased version of the process. For instance, notice that UCB uses an estimation of a confidence interval (based on the variance) to perform exploration. Consequently, it would not behave the same way when replayed than in the real world. It would be even worse for contextual algorithms that would learn a biased representation of the world. Dealing with this problem is not straightforward and would need further investigation. Nevertheless we do not believe that this is a big deal. Indeed we only consider using data with various kinds of logging policies for static policies while performing iterative improvements so as to reduce the variance of the estimations. In this context, only the mean matters. It would be preposterous to want to perform iterative improvements with a learning algorithm since by definition it is able to learn by itself.

Let us make a remark on what the two methods (online learning and iterative improvements) are implicitly assuming so as to understand better when they can be used. With iterative improvements, we only play static policies. Therefore we consider that the environment is not evolving very fast. Also the changes from one policy to the next one can not be dramatic since we need to change to something close so as to maintain a good accuracy in the evaluation. There are two ways to look at the problem. The first one is to consider the environment to be completely static. Therefore the iterative improvement procedure can be seen as a big optimization problem in which the goal is to converge toward the best policy. Depending on the problem, this may be considered realistic or not. For instance, the tastes of users in movies or books are known to be rather consistent and not to change much over time. On the contrary assuming such a thing for clothing or worse, new recommendation would be ridiculous. The second way to look at it is to consider that the environment is slowly shifting within a huge space and that iterative improvements are a way to "follow" the environment. In that case, a better suited name would be "iterative adjustments". Anyway, iterative improvements are nor made for an environment capable of sudden and abrupt changes such as what happens with news recommendation, or what we generally defined as dynamic recommendation. It makes sense for we already mentioned that this procedure was an alternative to the classical recommendation methods, based on predicting user ratings on past data. This procedure that we propose seemed to be flexible for it is adaptive by nature. In classical recommendation, taking into account the evolution over time is challenging. This was not really the first purpose of this thesis work but it would have been a shame not to talk about it, especially given that using replay for classical recommendation is (almost [98]) unanimously considered impossible. We show here that it is not the case.

Paradoxically, to evaluate a recommender system meant for dynamic recommendation, that is that changes really fast we do not need to continuously acquire data so as to let the system adapt. Only one dataset is enough. Indeed, with iterative improvements, we assume that recent past data is informative regarding what will happen in the future. With news recommendation this is not the case. Indeed, the news of yesterday are not the news of today. What we need to evaluate a dynamic recommender system is one dataset, long enough and relatively general to evaluate the capacity of the system to learn continuously. To do that effectively, we need the environment to be completely explored to let the learning algorithm perform its exploration. There is actually no logging policy that looks like a learning algorithm. Moreover we do not necessarily need recent data since we are not interested in what the algorithm learns but rather *how* and how fast it learns. This is the reason why someone interested in performing online learning to deal with its recommendation application should really sacrifice a day or two, maybe more to acquire a random uniform dataset so as to design an effective learning algorithm. This algorithm will then be able to deal with the recommendations on its own, to adapt to the changes in the environment without needing much updates. The major difference in spirit with iterative improvement is in fact the following. To design a static policy via iterative improvements, we need to continuously explore, but only a little. To design a learning algorithm, we only need to explore once, but a lot. Indeed, too little data would not enable to design an algorithm. Obviously exploration will keep being done but not by hand. It will only be performed by the designed algorithm, using advanced statistical methods so as to do only what is needed.

Finally there is a last consequence, although relatively minor, of the bias on the simulated process by replay. It affects the fact that R-BRED can estimate $CTR_{\pi}(T)$. Obviously when the policy is not uniform, this fact is not true anymore since the simulated interactions are weighted and therefore not distributed the same way. Yet the purpose of showing that we could do that was to highlight the fact that T interactions could be simulated via resampling, even though we only have T partially labeled records at hand. Consequently, regardless of the variance which can not be reduced in general to the level of T interactions with the real world, the bias in the evaluation of an evolving algorithm due to time acceleration can *a priori* be erased or at least dealt with. The very point of proving that was to show that it was possible to simulate T real world interactions with T records, in order to evaluate learning algorithms. As such the goal of that part of the analysis was to understand what we were doing mainly in the context of learning algorithms. The main part of the analysis of the bootstrap however was about showing that we could get a very accurate (better than any analytic approximation) of $CTR_{\pi}^{(estimator)}(T)$, and thus about the uncertainty on our estimation of g_{π}. This part obviously remains valid since the weighting does not affect the fact that we are dealing with independent samples that are identically distributed. It only changes the nature of the distribution we want and manage to estimate.

Variance

Now that the unbiasedness of the two estimators has been established, we can assert that their expected squared error (ESE) is equal to their variance. We recall that the inverse of that variance is our definition of information. Note that had we considered replay and RED∞, we would have had to study the inverse of the ESE, and not the variance for they are biased. Anyway, let us state the results for the variance of the unbiased estimators (the proof, which are tightly connected, are provided in appendix G.3 and G.4).

Theorem 19. *For any contextual bandit distribution $\mathcal{D}$, for any policy π, for any stochastic logging policy π_{LOG} such that for any context x and any action a, $p(\pi(x) = a) > 0 \Rightarrow$*

$p\left(\pi_{LOG}(x)=a\right)>0$, for any dataset S built by T interactions of π_{LOG} with $\mathcal{D}$ we have that the estimator provided by the replay method has the following variance:*

$$Var\left(\hat{g}_\pi^{(replay^*)}\right)=\frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}}\frac{p\left(\pi(x)=a\right)}{p\left(\pi_{LOG}(x)=a\right)}\mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[r^2\right]-g_\pi^2\right).$$

Theorem 20. *For any contextual bandit distribution $\mathcal{D}$, for any policy π, for any stochastic logging policy π_{LOG} such that for any context x and any action a, $p\left(\pi(x)=a\right)>0\Rightarrow p\left(\pi_{LOG}(x)=a\right)>0$, for any dataset S built by T interactions of π_{LOG} with $\mathcal{D}$ we have that for any B and when E goes to infinity, the estimator provided by the RED*∞ method has the following variance:*

$$Var\left(\hat{g}_\pi^{(RED^*\infty)}\right)=\frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}}\frac{p\left(\pi(x)=a\right)^2}{p\left(\pi_{LOG}(x)=a\right)}\mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[r^2\right]-g_\pi^2\right).$$

Before actually commenting on these results, a few details should be added. In the expression of the variance, the condition $p\left(\pi(x)=a\right)>0$ that is added when summing over all actions is here to allow us to be slightly more general and write $p\left(\pi(x)=a\right)>0\Rightarrow p\left(\pi_{LOG}(x)=a\right)>0$ instead of $\forall x,\forall a,p\left(\pi_{LOG}(x)=a\right)>0$. Note also that we use the alternative notations with:

$$\mathcal{D}=\{\mathcal{C},\{x\in\mathcal{C},a\in\{1,K\}\mid R_{x,a}\}\}\ .$$

This allows a simpler notation for the expectation inside the sum. Nonetheless, when talking about the expectation of the per trial payoff of a policy, dividing $\mathcal{D}$ this way yields a funny notation: $\mathbb{E}(r|x\sim\mathcal{C},a\sim\pi(x),r\sim R_{a,x})$. Consequently as a shortcut we will keep the previous notations when talking about the whole process: $\mathbb{E}_{\mathcal{D},\pi}(\vec{r}[\pi(x)])$, or even simpler since there is no ambiguity: $\mathbb{E}(\vec{r}[\pi(x)])$.

Furthermore note that $p(x)$ simply denotes the probability that x is drawn from $\mathcal{C}$. Finally, the sum implicitly assumes $\mathcal{X}$ to be discrete. This is obviously not necessary. We could also consider integrating over a continuous context space $\mathcal{X}$. Yet these results are meant to give intuition on the difference of convergence between RED∞ and replay so the simpler the better.

Now let us give more details about these results. First of all, notice that with the random uniform logging policy, that is to say if $p\left(\pi_{LOG}(x)=a\right)=\frac{1}{K}$ as in the rest of the thesis, we get the same result for replay* as in section 3.5.6:

$$\begin{aligned}Var\left(\hat{g}_\pi^{(replay^*)}\right) &= \frac{1}{T}\left(K.\sum_{x\in\mathcal{X}}p(x)\sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}}p\left(\pi(x)=a\right)\mathop{\mathbb{E}}_{\vec{r}\sim R_{x,a}}\left[r^2\right]-g_\pi^2\right)\\ &= \frac{K}{T}\mathbb{E}\left[\vec{r}[\pi(x)]^2\right]-\frac{g_\pi^2}{T}+\left(\frac{K}{T}.g_\pi^2-\frac{K}{T}g_\pi^2\right)\\ &= \frac{K}{T}Var\left(\vec{r}[\pi(x)]\right)+\frac{(K-1)g_\pi^2}{T}\ =\ \frac{K}{T}Var\left(\vec{r}[\pi(x)]\right)+O\left(\frac{1}{T}\right)\ .\end{aligned}$$

Now what can we deduce from these two results? The only difference between the variances of RED*∞ and replay* is the square on $p\left(\pi(x)=a\right)$. This is very nice for it confirms all of our intuitions. First if the policy under evaluation is deterministic, RED*∞ and replay* have strictly the same variance since $p\left(\pi(x)=a\right)$ can only be equal to 0 or 1 and thus is equal to its square. Therefore RED*∞ and replay* are strictly equivalent for deterministic policies.

Nonetheless, as soon as there is one tiny little bit of stochasticity, RED*∞ gets better than replay*. Indeed, for any $p\left(\pi(x)=a\right) \in]0,1[$, $p\left(\pi(x)=a\right)^2 < p\left(\pi(x)=a\right)$. Then the more stochastic the policy is, the greater is the gain of variance that RED*∞ yields. This fact is independent from the logging policy so it is also valid for the random uniform policy considered in most of this thesis work. Second it also confirms that with RED*∞, evaluating a policy close to the logging policy yields a convergence around $O\left(\sqrt{\frac{1}{T}}\right)$.

Indeed if $\pi = \pi_{LOG}$ exactly, the probabilities simplify and, we have that:

$$\begin{aligned}
Var\left(\hat{g}_\pi^{(RED^*\infty)}\right) &= \frac{1}{T}\left(\sum_{x\in\mathcal{X}} p(x) \sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}} p\left(\pi(x)=a\right) \mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[r^2\right] - g_\pi^2\right)\\
&= \frac{1}{T}\mathbb{E}\left[\vec{r}[\pi(x)]^2\right] - g_\pi^2 \;=\; \frac{1}{T}Var\left(\vec{r}[\pi(x)]\right) \;,
\end{aligned}$$

which is the best we can hope to achieve without knowing $\mathcal{D}$. Furthermore, since the inverse function is continuous, a logging policy in the neighborhood of π will allow to achieve results that are very close to $O\left(\sqrt{\frac{1}{T}}\right)$.

Note that this is **not true** for replay* (or replay). When $\pi = \pi_{LOG}$, replay* can only achieve a variance equal to $\frac{1}{T}Var\left(\vec{r}[\pi(x)]\right)$ if π is deterministic. Indeed, the probabilities also simplify but we get:

$$Var\left(\hat{g}_\pi^{(replay^*)}\right) = \frac{1}{T}\left(\sum_{x\in\mathcal{X}} p(x) \sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}} \mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[r^2\right] - g_\pi^2\right),$$

which is strictly greater than $\frac{1}{T}Var\left(\vec{r}[\pi(x)]\right)$ unless $p\left(\pi(x)=a\right)$ is equal to either 0 or 1 for all contexts and actions (see the first step in the calculation of the variance of RED*∞ when $\pi = \pi_{LOG}$). The gap between the variances of RED*∞ and replay* in this case is obviously problem dependent and cannot be analyzed much further.

In fact what does this variance reduction means? It means that expanding a dataset, combined with the replay method, is actually useful to drag more knowledge out the data than with replay alone. Therefore if it was really Utopian to think that we could acquire as much knowledge as T interactions using a dataset of size T in general, it is still possible to do much better than $\frac{T}{K}$, as long as the policy is able to make different choices given one context. This additional knowledge will be helpful when evaluating learning algorithms (this is discussed in more depth in section 5.8.4). To quantify this additional amount of knowledge used in the evaluation, we can just have a look at the formula of the variance of replay* and RED*∞. When we calculate the variance of replay* (which is equivalent to RED*∞ if the evaluated policy is deterministic) in the case of a random uniform logging policy, the double sum (that calculates the expectation of the square of the estimator on one record) simplifies to $K\,\mathbb{E}\left[\vec{r}[\pi(x)]\right]$. It yields a variance in $O\left(\frac{K}{T}Var\left(\vec{r}[\pi(x)]\right)\right)$, and thus a convergence in more or less $O_p\left(\sqrt{\frac{K}{T}}\right)$. In the case where $\pi = \pi_{LOG}$, the double sum in the variance of RED*∞ simplifies to $1.\,\mathbb{E}\left[\vec{r}[\pi(x)]\right]$ yielding a variance equal to $\frac{1}{T}Var\left(\vec{r}[\pi(x)]\right)$. In general RED*∞ yields a variance which is between these two values. In fact by calculating the variance, we have actually quantified Γ_π^+ which was mentioned in the previous section. Besides being very close to Γ_π^*, Γ_π^+ is affected the same way by the parameters of the problem. To be more specific, by definition, the variance of RED*∞ is equal to $\frac{\Gamma_\pi^+}{T}Var\left(\vec{r}[\pi(x)]\right)$, where:

$$\Gamma_\pi^+ \;=\; \frac{\displaystyle\sum_{x\in\mathcal{X}} p(x) \sum_{\substack{a\in\{1,K\},\\ p(\pi(x)=a)>0}} \frac{p(\pi(x)=a)^2}{p(\pi_{LOG}(x)=a)} \mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[r^2\right] - g_\pi^2}{Var\left(\vec{r}[\pi(x)]\right)}.$$

As a conclusion on this analysis, it was Utopian to believe that although BRED can *simulate* T interactions via resampling, it could always converge in $O\left(\frac{1}{\sqrt{T}}\right)$. Notice however that it is true or almost true in the case where $\pi \approx \pi_{LOG}$. We showed however that in most cases, BRED's bagged estimate converges in $O\left(\sqrt{\frac{\Gamma^*_\pi}{T}}\right)$, where $\Gamma^*_\pi \in [1, K]$ depends on the problem, π and π_{LOG}. More importantly we proved that in the case of the random uniform logging policy we considered throughout this work, Γ^*_π is almost always smaller than K, contrarily to replay for which it is always more or less equal to K. Consequently, BRED allows to reduce the variance of the evaluation of a policy. We discuss how this affects the evaluation of learning algorithms in section 5.8.4.

In a context of iterative improvements, this analysis shows that it can be done for very big values of K. Obviously this remains to be verified in practice. This is only a proof of concept. More importantly this analysis can be used to design a complete procedure of iterative improvements. Indeed it can be used to define the kind of randomization we want to apply to the current policy so as to minimize the variance of the evaluation of the next one. Indeed, a simple way to randomize the policy is to add random exploration. Yet, since we approximately know the kind of policy that is going to be evaluated next, we can use the results we presented here in order to minimize the variance of these future evaluations. More precisely, we actually want to minimize Γ^+_π as a function of π_{LOG} (pi_{LOG} is the policy that is going to be put online, π is the one we plan on evaluating after that). It is straightforward to notice a very interesting property: Γ^+_π is a convex function of π_{LOG} which makes it very easy to minimize. We will only prove this result for $K = 2$ in section 5.8.3 while studying a toy example. Yet we will see then that the generalization seems only natural. Note also that the actual way to design the current π so as to optimize the evaluation of the next one is not investigated here but would be a very interesting line of future research. Indeed, Γ^+_π's convexity may seem to make things easy, we can obviously not choose any logging policy. We can only randomize the current one a little. Thus we would have to solve a constrained convex optimization problem. Also this optimization problem is relatively noisy since we obviously do not have access to the true values of the $\mathbb{E}_{r\sim R_{x,a}}\, r^2$ for every contexts and actions. Nonetheless the ones we are interested in are known with the best precision since they are the ones which are explored the most by the policy. Anyway all this seems feasible and promising but should definitely be investigated further.

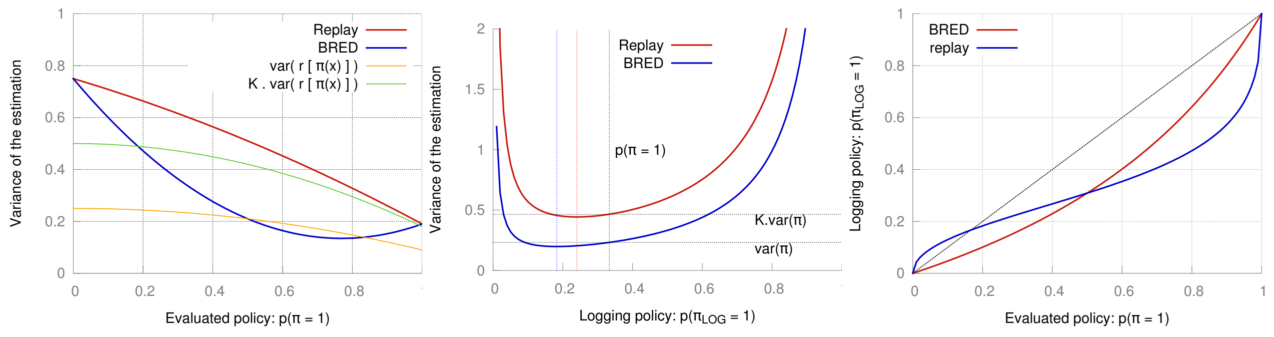
Finally note that given a dataset, computing Γ^+_π is rather useless. The only purpose of the formula should be to optimize the next randomization of a policy in a context of iterative improvements. Indeed, Γ^+_π is not easy to compute accurately given its expression. Note however that this does not prevent us from minimizing it using approximate dynamic programming [19] or other robust optimization tools. Furthermore it may slightly underestimate how accurate RED∞ really is (this should be checked more precisely) for it is based on RED*∞ which is more variate. For instance we can notice on figure 5.6 that Γ^*_π is equal to 3.5, that is RED∞ is able to extract 3.5 less knowledge than T interactions with the real world, or is 3.5 times more variate. As expected, we notice that for replay, $\Gamma^{(replay)}_\pi = 10$. When using the formulas of RED* and replay* we find significantly different results for ε-optimal for instance. Although this policy is contextual, it can be considered as a two actions, non contextual policy that plays the best action with probability $1-\varepsilon$ and an action at random with probability ε. Consequently we have that:

$$\begin{aligned}
\Gamma^+_{\varepsilon-opt} &= \frac{0.5^2 * g_{opt}/0.1 + 0.5^2 * g_{rand}/0.1 - g_\pi^2}{g_\pi(1-g_\pi)} \\
&\approx 7.990.427 * 0.427 \\
\Gamma^{(replay*)}_{\varepsilon-opt} &\approx 16.7.
\end{aligned}$$

The last line is obtained by the same formula after dropping the squares. Note the significance difference with replay and RED∞ for both estimators. Nonetheless using Γ^+_π to optimize a future evaluation with RED∞, which is more accurate than RED*∞ is probably the way to go. Indeed, we highlighted in chapter 3 that the accurate replay was not that easy to analyze. As a consequence, an optimization based on replay may not be possible given the complexity of the formulas about it. Also, since both estimators are summing over the same set of variables, but somehow average the result differently, they behave very similarly with respect to the logging policy, especially given that they use the same regularization factor $(p(\pi_{LOG}(x) = a)^{-1})$. Consequently we most likely do not lose anything by minimizing Γ_π^+ since we are only interested in the logging policy minimizing it and not in the value of Γ_π^* itself.

Furthermore note that the technique used here to compute Γ for ε-optimal is not considering all the context. On the contrary we simplified the computation by thinking about how the policy modeled the data. Indeed it would be foolish to try to estimate $\mathbb{E}(r|r \sim R_{x,a})$ for all actions and context. Instead, we can group contexts and actions without introducing any bias as long as the evaluated policy does not make differences within a group. To illustrate this, let us consider a simple example. If the context space is continuous but discretized into only three pieces by a model used by a policy. Then Γ_π can be computed without bias by only summing the expectations given the belonging to these three individual groups. Similarly, if a policy considers a linear correlation between a real valued context and the rewards, one should integrate using the same smoothness assumption so as to simplifying the Γ_π's computation.

5.8.3 Study of a simple case



(a) Theoretical variance of RED*∞ and replay* (for one record) over the probability that the evaluated policy π chooses action 1 for a random uniform logging policy π_{LOG}. Two baselines are plotted: the variance of a reward yielded from a real world interaction (orange) and K times that variance (green).

(b) Theoretical variance of RED*∞ and replay* (for one record) over the probability that the logging policy π_{LOG} chooses action 1 for a policy π such that $p(\pi = 1) = 1/3$. The blue (resp. red) vertical line indicates for which π_{LOG} RED*∞ (resp. replay*) reaches its minimum variance. Remark that for both methods, this minimum is not achieved for $\pi = \pi_{LOG}$ (black vertical line).

(c) Optimal (*i.e.* that yields the minimum variance) logging policy - characterized by its probability to choose action 1 - over the probability that the evaluated policy π chooses action 1 for both BRED and replay.

Figure 5.9: Study of a simple case with no contextual information and two actions (1 and 2) yielding Bernouilli rewards of respective probability of success $\mu_1 = 0.1$ and $\mu_2 = 0.5$.

So far our work on the respective convergence rates of RED∞ and replay has been very theoretical although it was somehow predicted by experiments. Our analysis outputted very intuitive results to understand why RED∞, but also the bagged estimates of all of BRED's variants are more accurate than replay. We also provided a way to quantify the difference.

Yet the way this difference evolves with the parameters (the model, the logging and evaluated policies) is less intuitive. If anything, we do know that it is equal to zero for deterministic evaluated policies. Nonetheless for other cases, studying this difference further than just saying it exists would be interesting. This is why we study a very simple but already very informative case in which $K = 2$ and no contextual information is available. This simple case will allow easy graphical visualizations of the results. We will consider two actions 1 and 2 that yield Bernoulli rewards of respective expectation $\mu_1 = 0.1$ and $\mu_2 = 0.5$. We will consider various logging and evaluated policies. They will be characterized by their probability to choose action 1: $p(\pi = 1)$. Obviously, $p(\pi = 2) = 1 - p(\pi = 1)$. Note that in this subsection, we do not run experiments and only plot curves based on formulas we derived in this work.

We are first interested in the improvement in terms of raw accuracy that is brought by using BRED instead of replay, when the logging policy is random uniform. Indeed, we know that if $p(\pi = 1) = 0$ or $p(\pi = 1) = 1$, there is no difference. What happens in between these values? Figure 5.9a answers this question quite clearly. First we can see that replay* has a variance that follows the curve of $K = 2$ times the variance of π, while remaining significantly above it. It just makes sense given the formula of the variance of replay*. We would notice the same thing with less difference for replay (see table 3.1 for a reminder of the formulas). This is not what is really important on this plot. RED*∞ is much more accurate than replay* as soon as π gets stochastic. For some value of $p(\pi = 1)$, it is even three times less variate which is quite remarkable given that $K = 2$. Notice a few other interesting features on figure 5.9a. The variance of RED*∞ is almost always lower than $K.Var\left(\vec{r}\left[\pi(x)\right]\right)$ and reaches $Var\left(\vec{r}\left[\pi(x)\right]\right)$ when $\pi \approx \pi_{LOG}$. It is also very interesting to point out that the variance of RED*∞ is even sometimes smaller than what we considered to be the best we could do. In fact this is because the two actions have different expected rewards. As displayed on the figure, a policy playing action 2 more often is more variate than a policy playing 1 more often. This is simply because the variance of a Bernoulli variable is maximized when its probability of success is 0.5. Then it decreases symmetrically whether this probability increases of decreases. In the context of news recommendation, the CTR is always less than 0.5 so we can just simplify and say that good news yield higher variance. Therefore here, the policy that yields the lowest variance is not π_{LOG} but a policy that plays 1 with probability approximately 0.8. It is the policy minimizing Γ_{π}^{+} for fixed π_{LOG}, μ_1 and μ_2. The mechanism is very interesting, and figure 5.9a exhibits it perfectly. We would get exactly the same type of behavior with more actions and contexts, although it would be much harder to plot. Yet we do not investigate it further since we are not interested in finding the policy that minimizes the variance given π_{LOG} but rather the other way around. Indeed what one may be after is the best way to randomize the current policy π (and therefore the future π_{LOG}) so as to minimize the variance of the next policies that will be evaluated. These curves were here to understand and quantify what is happening in general and they did just that.

So now let us consider the following question. How does the variance of RED∞ and replay* evolve relatively and absolutely for a fixed evaluated policy π when π_{LOG} changes? Here we consider the policy π such that $p\left(\pi = 1\right) = \frac{1}{3}$. The curves representing the variance against π_{LOG} are displayed on figure 5.9b. First notice the convexity of the two curves and the obvious fact that when the logging policy gets deterministic, the variance goes to infinity. Also notice as we explained it above that the minimum of variance if not reached when $\pi = \pi_{LOG}$. Since playing action 2 yields more variance, the best logging policy plays it a little bit more than the evaluated policy. This allows replay* (resp. RED*∞) to achieve a variance slightly lower than $K.Var\left(\vec{r}\left[\pi(x)\right]\right)$ (resp. $Var\left(\vec{r}\left[\pi(x)\right]\right)$) when the right logging policy is picked. Note that these exact values are reached when $\pi = \pi_{LOG}$ exactly. As an intuitive consequence of this observation, when randomizing π in a context of iterative improvements, one should try to play the best actions with higher probability than the others so as to minimize the evaluation of future policies. A very nice side effect of this is that it may allow to reduce the cost of randomization or to perform more randomization for the same cost as before, since playing good actions is less expansive in terms of performance.

The last problem we consider here is how to randomize π precisely so as to minimize the variance of the next evaluated policy in the context of iterative improvements. We gave intuitions so far but we can go further. We mentioned that Γ_π^+ looks convex (figure 5.9b). The following result (proved in appendix G.5) makes things clearer.

Theorem 21. *For any non contextual bandit problem $\mathcal{D}$ with two actions (of respective reward expectations μ_1 and μ_2) and any fixed policy π characterized by its probability to choose action 1: $p(\pi = 1) \in]0,1[$, the variance of its evaluation by either replay* or RED*∞ (as characterized in theorems 19 and 20) on any T records acquired by a logging policy π_{LOG} (characterized by $p(\pi_{LOG} = 1) \in]0,1[$) is a* ***convex function*** *of $p(\pi_{LOG} = 1)$.*

Using the following notations:

- *$a = p(\pi = 1)^2.\mu_1$ for RED*∞, $a = p(\pi = 1).\mu_1$ for replay*,*
- *$b = (1 - p(\pi = 1))^2.\mu_2$ for RED*∞, $b = (1 - p(\pi = 1)).\mu_2$ for replay*,*
- *$c = \frac{b}{a}$.*

The logging policy π_{min} that ***minimizes the variance*** *of an evaluation of π is then characterized as follows:*

$$p(\pi_{min} = 1) = \frac{1}{2}, \qquad if\ a = b$$
$$p(\pi_{min} = 1) = \frac{1 - \sqrt{c}}{1 - c}, \qquad if\ a \neq b$$

We strongly suspect this theorem to be true for any bandit problem, although we may need to use convex optimization instead of a closed form solution to find π_{min} in more complex cases. We leave the proof of such a result for a future work.

Note that in general we do not have full liberty of choosing π_{LOG}. We can only decide how to randomize the current policy without hurting the performance too much. Therefore we may have to perform convex optimization with respect to some constraints even though a formula might exist. Nonetheless we plotted this optimal logging policy in a minimum variance sense for all the possible evaluated policies on figure 5.9c. We can clearly see that for RED*∞, it is always best to play the news with higher expected reward a little bit more. With replay, the behavior is a little bit more erratic. Indeed, in theory RED∞ takes all the knowledge there is to take from the dataset. That is not what replay does. As a consequence when the probability that π chooses 1 (the "bad" action) is very low, we need that π_{LOG} chooses it slightly more often to limit the high variance that arises when action 1 is never or almost never replayed.

5.8.4 Discussion: why does it matter to evaluate stochastic policies?

Although we already gave a few hints supporting the contrary, all this may still seem like a lot of fuss about nothing. Indeed, in practice one may not be very interested in evaluating stochastic policies. Therefore a method that is only better for that kind of policies may seem useless. However, in a context of iterative improvements of a recommender system, the successive policies put online need to be stochastic so as to allow the evaluation of the future ones. We do need to evaluate these policies (in order to determine an amount of randomization that does not deteriorate the performance too much for instance) and BRED would yield better precision in that case. See [98] for more arguments supporting this claim in this context.

Obviously BRED will enable to get a slightly better estimation but the difference is not going to be huge. Fortunately however, this is not the main goal of this analysis. The first point, which we already discussed is that given that the design of a recommender system is seen as an iterative process, it can be evaluated with a reasonable accuracy by replay methodologies. This is true for both replay and RED∞ with stochastic or deterministic policies. Indeed, it is not because a policy is deterministic that it does not take advantage of a logging policy close to it. On the contrary this is the very thing we aim at doing with iterative improvements. This justifies the variance analysis but not the introduction of BRED as way to reduce the variance of evaluation methods. Let us justify that as well.

The second interesting idea to get out of all this is that if a policy (or an algorithm) is able to make different choices given one single context, then we are able to drag more information out of the data than what replay enables. We saw that in the context of iterative improvements, besides being able to quantify more precisely the cost of randomization, this was not crucial. Nonetheless, what is a learning algorithm if not an algorithm that by essence makes different choices over time. For each record, a learning algorithm can make a different choice whether it sees it at the beginning of the learning process or at the end. Moreover even it if it sees it twice roughly at the same time and especially at the beginning of the learning process, exploration is about trying other options so the algorithm will again necessarily make different choices and consequently use a lot more information than T/K records. Finally, exploration in itself allows different choices but that is not it. An algorithm typically goes back and forth between exploration and exploitation, yielding again possibilities of different choices in a given context. To sum up, this analysis on stochastic policies gives a lot of intuition as to why expansion but also bootstrapping would reduce variance in the evaluation of learning algorithm in addition to dealing with time acceleration. Indeed we recall that one expanded replay does not drag all the information there is within the data. It is preposterous to expand the data infinitely when evaluating a learning algorithm or we would completely hide the cost of exploration. Therefore bootstrapping will act as a variance reduction tool exactly as when bagging strong learners. Remark that this variance analysis is a formal proof justifying why bagging reduces variance in our special case.

Remark also that obviously the variance results obtained here do not apply to algorithms in general. Yet a learning algorithm is a very particular algorithm and has nothing to do with the strange algorithm used to prove that no convergence results could be achieved in section 3.5.3. Somehow, an algorithm such as UCB is a stochastic policy (exploration) that slowly and more importantly smoothly converges to a deterministic policy (pure exploitation). For most algorithms, it is even proved that they consistently converge. As a consequence, we can very reasonably assume that the evaluation of a learning algorithm with BRED will have convergence properties that are better than those of a deterministic policy (given the exploration) but not as good as a fully random policy (convergence to a stationary regime). Anyway, expansion definitely enables variance reduction in addition of bias reduction for learning algorithms, whose evaluation was the first goal of this thesis work. In this we can really say that we achieved even more than expected.

There is a third reason why exhibiting improvement for stochastic policies is important. Paradoxically this has to do with the evaluation of deterministic policies. Indeed, we demonstrate empirically in an upcoming section that we can somehow force a deterministic policy to make difference choices when being confronted to the same record. This surprisingly yields increased accuracy in the evaluation. This analysis on stochastic policies provides theoretical ground to justify that.

To conclude on this, this analysis shows two major things.

1. We are able to evaluate classical recommendation with replay.

2. In addition to bias reduction, BRED will most likely allow to reduce as well the variance

of the evaluation of learning algorithms. Since, as convincing as it is, only intuition is provided with that regard, this is verified in this chapter in an empirical study (see section 5.10).

We did not push our investigation about the first point any further. Nonetheless we believe that if any research is worth being conducted following this thesis work, it should be about that. Figure 5.10 is a diagram developing a little bit more how iterative improvements could be performed and even completely automatized to form one and only learning process. In a nutshell, one could very well see the entire procedure as an optimization problem over π in which one has to minimize risk of putting a new policy online based on the bootstrapping techniques we introduced. Moreover one would have to balance between exploration (randomization of the policy) and exploitation (current best policy). Note that we already mentioned that the optimal exploration could be found by convex optimization. A way to define the extent of that exploration (handling the exploration/exploitation dilemma), is still to be thought of. It seems highly feasible and would definitely bring another look at how recommender systems are built and evaluated. The second point is what we were after in the first place and even more (we did not really expect to reduce variance). Nevertheless, an issue remains unsolved. The bias reduction was conditioned on the fact that the resampling procedure induced by the bootstrap was a fine estimation of $\mathcal{D}$ the bandit distribution. Figure 5.8 showed that if it was good enough for non contextual algorithm, it was not the case for contextual algorithms such as LinUCB, or at least not without a lot of data. Indeed, in that case the negative bias due to time acceleration is simply replaced by a more dangerous positive bias caused by overfitting. This is why we will now talk about how this estimation can actually be improved using Jittering. Before moving on, note a last funny fact. Jittering is also the mean used to make the evaluation of deterministic policies more accurate by making them do different choices. The same thing is potentially true for learning algorithms, whether they are exploring or exploiting. As a consequence this is one more argument indicating that BRED, and in particular when it is Jittered, should reduce the variance of the estimation.

5.9 On Jittering and bootstrapping

5.9.1 Justification of Jittering

So far Jittering, that we propose to evaluate contextual bandit algorithms with BRED, may seem like a patch-up job to avoid overfitting. In the neural network field, from which we took the idea, it is mostly justified by the same intuitive and practical arguments that we already gave. Yet we will see in this subsection that Jittering is completely justified from a theoretical point of view and far from being novel in the bootstrap field although known under a different name. Note however that the purpose for which it is used in classical bootstrapping is very different.

Let us go back to classical bootstrapping techniques. We consider that we have a dataset S generated via T samples from a real valued distribution F. The bootstrap is about approximating F by $\hat{F}(S)$, a distribution from which we sample by sampling with replacement from S. We recall that this is the non parametric bootstrap. We could obviously assume that F belongs to a family of distributions. It would thus be parametrized by a vector θ of parameters ($F = F_\theta$). In this case, we would estimate θ from S, and get $\hat{\theta}(S)$. This would yield a parametric estimation of F: $F_{\hat{\theta}(S)}$ (see Efron and Tibshirani [109] for more details).

The problem we have when using BRED is that the quality of $\hat{F}_{\pi_{LOG}}$, the bootstrap distribution is not good enough which causes bias in the evaluation of contextual bandit algorithms. In other words, the distribution is a somehow too discrete version of reality and repetitively sampling from it allows the algorithm to overfit the data.

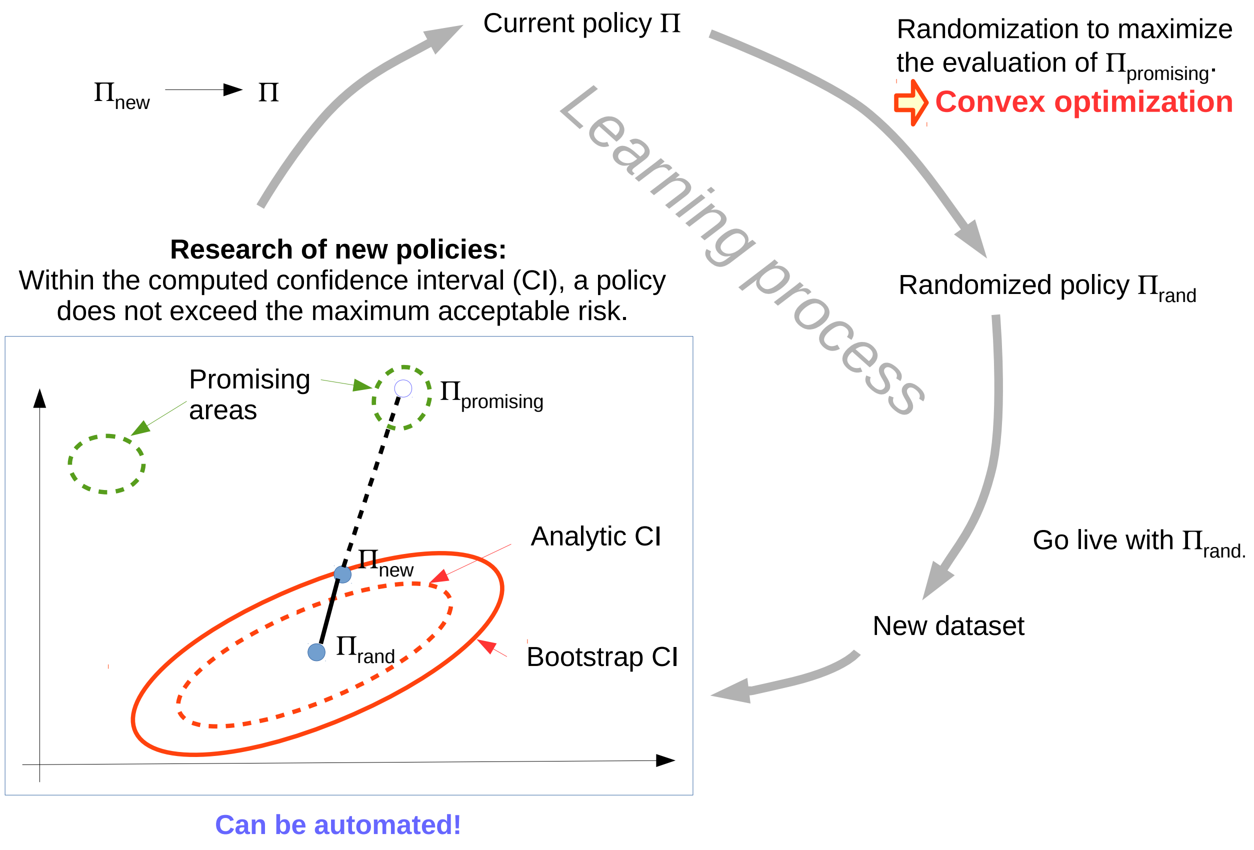


Figure 5.10: Enhanced "iterative improvements". In this work we improve the accuracy of the confidence intervals (CI) thanks to the bootstrap and BRED in particular. A direct and highly desirable consequence is the expansion of the range of choice for a new policy to put online at each iteration. We also show the choice of the randomization to maximize the evaluation of a future policy to be a convex optimization problem and thus easy to deal with optimally. Finally a very exciting perspective is that this entire loop is in fact a learning process of a policy that could clearly be automated.

A parametric estimation would definitely solve this issue. Indeed, once θ is estimated, we have access to a full continuous distribution $F_{\hat{\theta}}$. Yet doing so is a terrible idea. Indeed notice that a parametric method requires assumption on the data. It is equivalent to training a model of the environment on some data. Such an approach was disproved in chapter 1 and is generally considered to be a bad idea by the community except to find bugs or evaluate very specific behaviors (see chapter 7 of *Recommendation: an introduction* [24] and chapter 8 of *Recommender Systems Handbook* [3] for much more details). Among other things, it usually causes the design of algorithms that are biased toward the model and not very effective in the real life.

Unsurprisingly, this problem of bad estimation by the non parametric bootstrap is not new. To tackle this issue in the context of bootstrapping there exists a small variant called the smoothed bootstrap, introduced by Silverman and Young [15]. The idea is that the bootstrapped estimator may be inaccurate and very variate for discrete data or when too little data is available. To overcome this issue, Silverman and Young propose to smooth the data in order to, in a way, fill in the blanks. Intuitively the idea is simple: Silverman and Young propose to introduce bias in order to reduce the variance of the estimation. Note that in special cases, when too little data is available, smoothing may also reduce the bias. Indeed, although smoothing is equivalent to adding bias, a smoothed distribution might still be a less biased approximation than a very discrete one based on sampling with replacement from very few records. We will not go into the details of when to smooth the data and when not to. The topic is addressed with plethora of details in Silverman and Young's work [15]. Smoothing is also well studied in data analysis in general. It is generally performed via a kernel density estimation (KDE) of F using the data. Therefore smoothed bootstrapping is just a slightly more elaborate non-parametric bootstrapping technique that samples from a KDE computed on the data instead of doing it directly from the data.

5.9.2 Kernel Density Estimation (KDE)

If we denote by $S = \{s_1, s_2, ...s_T\}$ a (univariate and real valued) dataset, a kernel density estimator is simply defined as follows:

$$\hat{F}(x) = \frac{1}{T.h} \sum_{i=i}^{T} Ker\left(\frac{x - s_i}{h}\right),$$

where $Ker(\dot{})$ is the Kernel (positive-valued, symmetric function that integrates to one). h is a parameter called the bandwidth. It accounts for the amount of smoothness that we want the estimation of F to have. In other words if h is too small, the estimate will be very close to the empirical distribution of the data and will thus remain too discrete. If it is too big, the estimate will be very smooth and we may lose information by hiding the underlying structure of the data. Therefore there is a clear trade-off between variance and bias when computing a KDE.

Many kernel functions $Ker(\dot{})$ are possible. Fortunately KDE has been extensively studied in the literature. First the Epanechnikov kernel was proved to be optimal in a minimum variance sense [133]. Nevertheless it was also exhibited that the choice of the Kernel is of little importance in practice compared to a careful choice for h [134]. Therefore the Gaussian kernel is the most common choice for people are used to it and because it offers nice mathematical properties. Since it was highlighted to be the key parameter of the procedure, the choice of the bandwidth was also subject to intense scrutinizing. It is important to have in mind that in general it is not possible to derive an analytic formula for the best h. Yet in a very particular case, that is under normality assumption and for univariate data, it can be proved that the optimal bandwidth h is:

$$h = \hat{\sigma}\left(\frac{4}{3T}\right)^{\frac{1}{5}} \approx 1.06\hat{\sigma}T^{-\frac{1}{5}},$$

where $\hat{\sigma}$ is the standard deviation of the data. This value is usually referred to as Silverman's rule of thumb and it was exhibited to be a correct value in a wide range of settings [135]. In general h is chosen in a data-driven fashion. There is no consensus on the method to use but the methods based on cross validation or plug-in selectors seem to work well in practice. See Jones *et al.* [136] for a comparative study of various methods and more details. A lot of practical work on the choice of the kernel (and the associated bandwidth) in the context of pure KDE or smoothed bootstrap considers univariate data. Most of it can be easily adapted to multivariate data but for more details, the reader is referred to Scott [137].

5.9.3 When Jittering meets Smoothing

That being said, what does this have to do with Jittering? When bootstrapping, we need to be able to sample from $\hat{F}$. A very nice property of KDE is that if we use the Gaussian kernel, sampling from the KDE given S is equivalent to sampling a record uniformly from S and to apply some normally distributed noise of mean 0 and standard deviation h [15]. Note that sampling from a KDE built with another kernel can be achieved in a similar way, using different noise distributions. Yet the reader should keep in mind that the choice of the kernel was exhibited in the literature to be of little importance. Therefore, what we called Jittering is exactly the same thing as sampling from a Gaussian kernel density estimation of the true generating distribution F given S.

There is one difference with a classical smoothed bootstrap though. Our distribution F does not simply generates real numbers. It is actually composed of three sub distributions ($\mathcal{C}$, $\mathcal{U}_{\{1..K\}}$, R). We only make a KDE on $\mathcal{C}$, the other ones being left unchanged when sampling so that their density is estimated with the classical non parametric bootstrap. Thus $\hat{F}$ is only partially smoothed. When we introduced the concept of Jittering here, it seemed rather obvious to introduce noise in the context so as to avoid overfitting. In the light of what has been said about the smoothed bootstrap, we may be able to do more. Indeed, nothing prevents us from smoothing the rewards in addition to smoothing the contexts. Since our estimator is in the end based on these rewards, we would achieve a very similar purpose as what is classically done with the smoothed bootstrap. Smoothing the rewards would not really prevent the algorithm from overfitting. Yet we may end up with a smoother version of $CTR_{\pi}^{(estimator)}(T)$. In that case we could benefit from the smoothed bootstrap the same way as statistician do, that is by possibly getting estimates with less variance but more bias. It might be of use with very little data. We do not study this in this work because it is far from being our major concern. Moreover we exhibited rather satisfying experimental results as far as the estimation of the distribution is concerned. In practice we do believe that we have enough data not to need to do that, although this could be worth further investigation. For now let us just keep in mind that it is an option when using BRED when the reward functions are continuous and whose estimations could be hurt by using too little data (which would yield a too discrete approximation). Here we recall that the Yahoo! Today dataset [12] is composed of Bernoulli rewards so we do not experience this issue. Also smoothing the rewards would prevent the evaluation of any algorithm that considers the rewards as what they are, that is binary outcomes.

Finally smoothing the distribution that generates the actions would not be possible since although it is fully known, it is discrete. Nevertheless, given precisely that it is fully known, we could resample from it. What would it implicate in the approximation of F, that generates all the data? We would first sample a context at random from the data or the KDE of $\mathcal{C}$ given the data. Then, instead of taking the associated action, we would take one at random, simulating π_{LOG} perfectly. In order to generate an associated reward, we would have to approximate $R_{a,x}$ for all action a and context x. Were $|\mathcal{X}|$ quite small, we could consider all the rewards associated to a and x in S and sample with replacement, possibly adding some Jittering. We would then perform a complete non parametric approximation of F given S. The problem we have is that $|\mathcal{X}|$ is generally huge (2^{135} in the Yahoo! Today dataset [12]) or worse, continuous. Thus it is simply impossible to approximate $R_{a,x}$ in a non-parametric fashion. The only option to do it would be to introduce assumptions and build a parametric estimation. Yet again this is equivalent to building a reward model which we already argued against at length in chapter 1 and in this one. This is basically the reason why we only end up performing partially smoothed bootstrapping to evaluate learning algorithms.

Remark Note that all the work on bandwidth selection and KDE in general aims at building an estimate which is as close as possible from the true distribution. To do so, one faces a bias/variance dilemma. In our situation, not smoothing the data enough leads to overfitting by the learning algorithm and therefore bias. Smoothing it too much leads to a loss of information which in turn also causes bias. Thus we have to make a trade-off between two sources of bias and the optimal bandwidth for a KDE may not be the one solving our problem optimally, although it would certainly be informative to know it. This is problematic. Finding a good bandwidth in the KDE sense would be possible using methods from the literature (see Jones *et al.* [136]). Nevertheless we already have to expand our dataset in order to make accurate enough evaluations. Therefore classical cross validation would be very difficult with such a limited amount of data. We will come to this problem in chapter A.

Note finally that although we are basically performing (partially) smoothed bootstrapping, we kept the word Jittering since it was used in the literature to facilitate a learning process, which is what we aim at doing here.

In the set of experiments we ran, we made very classical choices with respect to Jittering for we just wanted a proof of concept for BRED. We used a Gaussian kernel for it is the most commonly used in the literature. For h, we took values in $O\left(\frac{1}{\sqrt{T}}\right)$ for they seemed to work equally well for any value of T. Furthermore $\frac{1}{\sqrt{T}}$ is a common value used as a first approximation in various textbooks on statistics. See Liebke [138] for a very simple working example using this value for h. The author uses smoothed bootstrapping to improve the estimation of the median of the speed of light measured during Simon Newcomb's famous experiment in 1882. Note that this example is also very interesting to see how smoothing helps with the bootstrapped estimation of the bias of a median. Finally $\sigma/\sqrt{T}$ is the standard deviation of the posterior distribution of the data when no prior is taken (σ being the standard deviation of the data). It seems like a reasonable thing to smooth some data with a value proportional to its uncertainty.

5.10 Experiments

This section is about what happens in practice when evaluating contextual bandit algorithms. We already argued that although they do not meet the requirements of the analysis, they can be considered as a stochastic policy smoothly and consistently converging toward a deterministic policy. For a given algorithm, although the path toward their stationary regime can be more or less chaotic, most of them are proved to be consistent and always end up learning the same (optimal) policy. The only issue is in fact the learning process which removes independence between the processing of each records and voids the reasons why both expansion and the bootstrap works in general. Yet we argued that if $F_{\pi_{LOG}}$'s estimation by the bootstrap procedure is good enough, then it does not matter. Furthermore one may consider that Jittering or smoothing introduces bias and thus the bootstrap as well as expansion should not work either. Nevertheless, it was proved that in an extremely wide range of settings (basically smoothness or sometimes differentiability assumptions that are needed anyway for the bootstrap) it works anyway because the quality of estimation of the cumulants of F by $\hat{F}$, the smoothed estimation of F given S remains similar [15]. In some case, especially when F is strange or S is small, smoothing was exhibited to increase the empirical accuracy of estimation and its robustness.

In this section we first are interested in what Jittering can bring to stationary policies empirically. Then we see if everything works as expected with learning algorithms although in that case, none of our assumptions hold.

5.10.1 Stationary policies and Jittering

We proved analytically that RED∞ and consequently all of BRED's variants are of no help when it comes to improving the evaluation of a deterministic policy. Yet, in the context of classical recommendation, this is the kind of policy that a data scientist would be the more interested in evaluating. Indeed, even in the context of iterative improvements, the policy we evaluate are only slightly randomized and the part we care the most about is its deterministic part. Also a learning algorithm converges toward a deterministic policy so improving the accuracy of their evaluation should improve the accuracy of evaluation of learning algorithms.

We argued that introducing Jittering helped improving the quality of our estimation of $F_{\pi_{LOG}}$. Can we use that to our advantage in order to improve the evaluation of a stationary policy as well? Let us consider the following idea. A deterministic policy π is just a partition of the context space to which actions are assigned. Obviously a record with action a performed in an area of the context space very close to its assigned area would be informative with regard

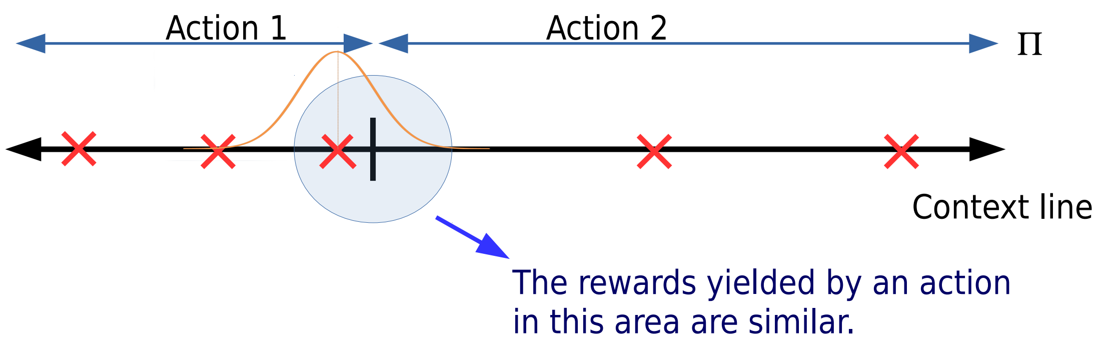
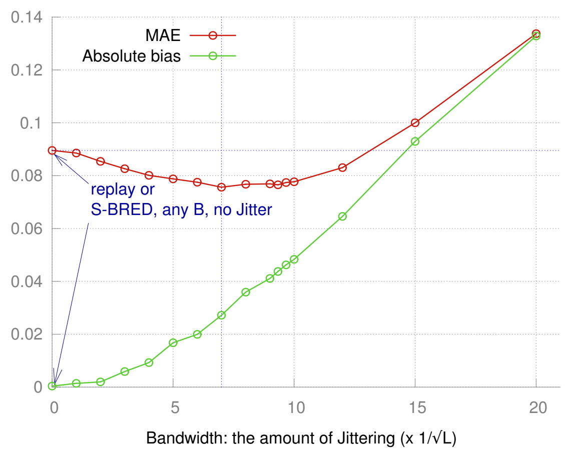


Figure 5.11: This figures illustrates at least one reason why Jittering improves the evaluation of deterministic policies. We consider a real valued context space (the context line) and two actions. A policy π is simply a partition of the context space, each part being assigned to one and only one action. The red crosses represent data points (context, action, reward). Now let us consider a frontier between two actions on the context line. The result of action 1 is very likely to be similar on either side of the frontier and the same thing goes for action 2. Note that this does not mean that action 1 and 2 have similar results. We know nothing about that. Anyway let us consider the cross in the blue circle. If it is associated to action 1, then RED∞ or replay are able to use that information. On the contrary, if it is associated with action 2, the point is just ignored by both estimators even though we know given how close it is from the border, that the point is informative with regard to π's evaluation. Adding Jitter will allow that point to sometimes be on the right side and let RED∞ use the information it contains. Also the randomness combined with the large number of iterations will act as a weighting system, giving more weight to records that are naturally on the right side of the border, less weight to records on the wrong side but not too far and even less to records farther away. Note that only one Jittered pass through the data (which is what replay does) would not enable this natural weighting scheme to occur and would just deteriorate the performance of the estimator by moving the data points around.

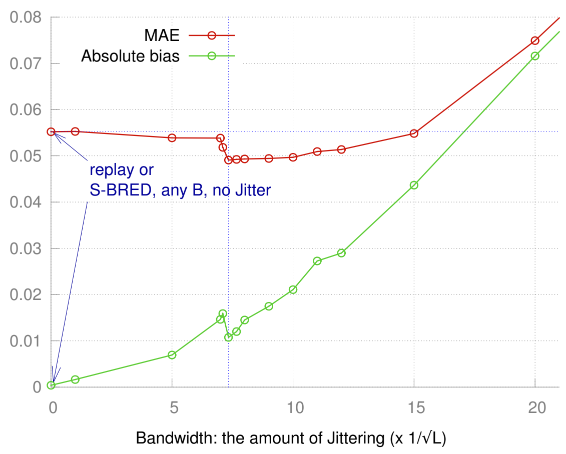
to the evaluation of π. Yet both replay and RED∞ would simply ignored that record. By introducing Jittering, we allow the estimator to use the information contained in that record. Replay's estimation would just be hurt by the introduced noise and thus bias. Indeed replay only considers each record once so if noise is added, it may prevent it from using all the information it contains. On the contrary, RED∞ considers each record an infinite number of time (at least in theory). Therefore each record will in expectation remain at its place. Nevertheless a point in the context space will not be considered as a point anymore but as an infinite Gaussian hypersphere. It means that the further away they are from the center, the less the data points will have weight in the evaluation. This will allow to drag more knowledge out of the data by also considering the neighborhood of each context point and naturally weighting the results by letting a random Gaussian process converge to its expectation. Figure 5.11 illustrates this idea with a sketch and a one dimension context space. Consequently the Gaussian hypersphere is actually an infinite Gaussian line, that is a bell curve.

To sum up, if the reward functions are not too crazy, that is that we can assume two neighboring contexts to yield similarly distributed rewards given an action, then Jittering is not only a regularization tool meant to avoid overfitting. It is also a means to allow replay methodologies to be less blinkered somehow in their choices to ignore records or not. Few people would argue that the addition of noise adds knowledge. This is obviously not the case, it simply enables to drag more information out of the data.

The experiment we consider here aims at verifying that all this is true when evaluating the optimal static policy for the synthetic model. Note that we will only be interested in the



(a) $T = 200$



(b) $T = 500$

Figure 5.12: Evaluation of the optimal policy of the synthetic model using S-BRED, B=10. We plot the Mean Absolute Error (MAE) and the Absolute Mean Error which is in fact the absolute bias. When Jittering is equal to 0, this is equivalent to performing replay. There is an exponentially small bias in that case (≈ 0). Then as we introduce Jitter, the bias increases roughly linearly. On the contrary the average error decreases slowly before reaching an optimal value and going up again. This is a typical variance *vs.* bias dilemma. This proves that Jittering does help improving the accuracy of BRED's bagged estimate. Notice that when $T = 200$ (fig. a) the plot is an archetype of the variance/bias trade-off. When $T = 500$ (fig. b), strangely, Jittering makes the error drop suddenly, so suddenly that even the bias decreases. We have no clear explanation for this phenomenon. It is possible that only a few areas of the context/action space that were insufficiently exploited by replay were interesting. The error drop happens when the sufficient amount of noise that lets the policy explore these particular areas is reached.

estimated mean of the CTR and not by the full distribution for the sake of simplicity. Indeed, since we manage to drag more knowledge than $\gamma_\pi*$, let us say $\gamma_\pi^{(jit)}$, the estimation of the mean is by definition in $O_p\left(\frac{1}{\sqrt{\gamma_\pi^{(jit)}}} = \sqrt{\frac{\Gamma_\pi^{(jit)}}{T}}\right)$, and so will necessarily be the other sample moments. Consequently it yields an estimation of the Studentized CDF in $O_p\left(\frac{\sqrt{\Gamma_\pi^{(jit)}}}{T}\right)$. We evaluated this policy using various Jittering parameters that are proportional to $\sqrt{1T}$. We tried two different values for T: 200 and 500 and averaged the results of 5,000 runs. Before having a look at the results, let us remind that without Jitter, S-BRED, RED∞ and the simple replay method would output the same estimate all the time. Increasing B would not change a thing. Note that it is only true for BRED when B is big enough because of the bootstrap resamples that may not take into account all the records. Figure 5.12 displays the results which are pretty positive. BRED does manage to improve the estimation of a static policy by adding Jittering. Note also the very nice curves, typical of a bias/variance trade-off. More comments and details are available directly on the figure.

This experiment although quite successful only shows that it works for our model. Nonetheless, the intuition behind the mechanism which is at play seems rather universal: we only need the rewards to be similar in neighboring contexts for one given action. This is the main intuition behind LinUCB for instance which is used in practice at Yahoo! [7]. Consequently, Jittering should help improving the evaluation of policies in a wide range of contexts. Notice that we are actually making an assumption here, although very weak and more or less universal. In a way this is a step, although very tiny, toward parametrizing the resampling process. Obviously, any additional assumption that can be made on some dataset may introduce bias (similarly to Jittering, see figure 5.12), but also help reducing significantly the variance of the evaluation by introducing parameters that may be learned from the data. Nevertheless, one should be very careful for although we have guarantees when we use RED∞ or replay on raw data, they are all lost as soon as a bias that cannot be quantified is introduced. Slightly smoothing by introducing Gaussian noise is hardly risky, and it is possible that theoretical guarantees might be derived in a wide range of settings using error bounds from the KDE literature. See for instance Wand and Jones [134]. For any other assumptions, the risk to introduce uncontrollable bias would be much greater, so doing so should be done with great care.

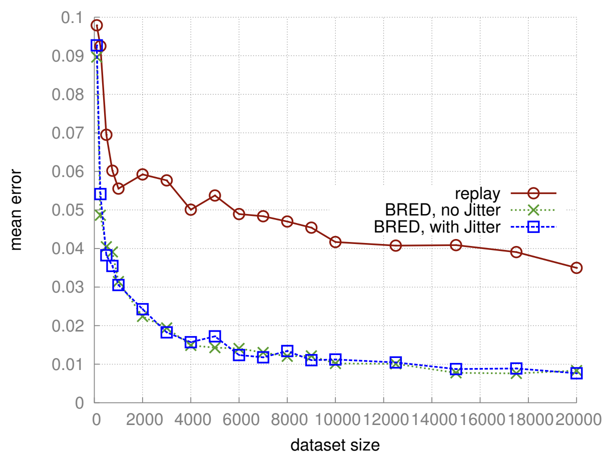
5.10.2 Experiments on learning algorithms

With learning algorithms choosing how to use BRED is much more simple. Indeed, we studied that for stationary policies, many options were possible, to estimate different things with a varying accuracy. Here, in order to minimize the bias we have to simulate T interactions and thus use BRED (or one of its variants) with an expansion factor equal to K. Our possible goals with stationary policies were to estimate the per trial payoff of the policy g_π, the distribution when it is played online $CTR_\pi(T)$ and the distribution of g_π's estimator (BRED or RED∞). With learning algorithms, we recall that replay only outputs an estimation of $g_\pi(T/K)$ and estimating $g_\pi(T)$ is already challenging enough. Indeed, contrarily to the evaluation of stationary policies, the number of records matters for otherwise some bias, referred to as time acceleration is introduced. Consequently, our first set of experiments is about the estimation of $g_\pi(T)$ using BRED

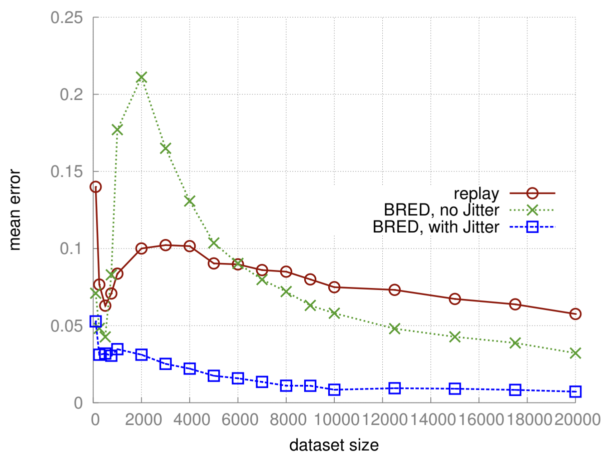
We already displayed the bias of the bagged estimate of BRED (see figure 5.8) but this was mainly to justify the use of Jittering. It said nothing about accuracy. This is what we investigate here in the first place. Then we will talk a little bit about how we can quantify the accuracy we achieve. Finally we will also see if we manage to estimate the distribution of both $CTR_\pi(T)$ (which is not very useful) and $CTR_\pi^{(estimator)}(T)$, that can be used to estimate the uncertainty on our estimation on $g_\pi(T)$ using the estimator BRED or S-BRED.

Simple error rate

The first experiment we run here is very simple: we evaluate both UCB and LinUCB parametrized as in section 5.2 using replay, S-BRED without Jitter and S-BRED with Jitter. We took a Jitter parameter or bandwidth $h = \frac{75}{\sqrt{T}}$ simply because it seemed to work fine when trying it for a few iterations and kept $B = 1$ to save computational resources. It is possible that the results could be improved further by tweaking h more carefully and it is sure that we would get more accuracy with $B > 1$. The model is generated as in section 5.2. The results are averaged over 200 generated datasets and are displayed on figure 5.13. The curves are eloquent enough not to need much commenting. In a few words, the improvement due to the use of BRED is tremendous. UCB obviously does not need Jittering as it is non-contextual (see figure 5.13a). On the contrary LinUCB is clearly wrongly evaluated when no Jitter is introduced (see figure 5.13b). This is because it clearly overfits the data. Note that when the horizon grows, as expected Jittering becomes less and less necessary. The Jitter-free BRED even ends up being much better than Replay. There is one little surprising detail on this figure that only arises when evaluating LinUCB. The error rate for very small horizons is quite low and increases afterward before decreasing slowly again as it is expected. This is probably due to the fact that LinUCB, in order to capture the full contextual structure of the data makes a lot of exploration at the beginning. Therefore it has very low CTR at that time which is easily evaluated. Nonetheless, as soon as the horizon is big enough to let LinUCB start exploiting things it has learned (when $T > 1,000$ on the figure), time acceleration does its job and the error rate for replay takes off.



(a) Evaluation of *UCB*: Jittering is useless. Indeed *UCB* does not use the contextual information so tampering with it does not change a thing.



(b) Evaluation of *LinUCB*: Jittering helps a lot. Low error values when the horizon is small are explained by the fact that *LinUCB* performs very poorly and is therefore easy to evaluate.

Figure 5.13: Mean absolute error (MAE) of the evaluation of LinUCB and UCB using replay and S-BRED (B=1) with or without Jittering over the size of the dataset which is also the horizon of the bandit game. Lower is better.

Note that the error rate of the Jitter-free BRED increases much more abruptly than replay's since LinUCB starts overfitting the data. To sum up, replay has a negative bias due to time acceleration. BRED without Jitter has a positive bias because of overfitting. BRED with Jitter manages to minimize both sources of bias.

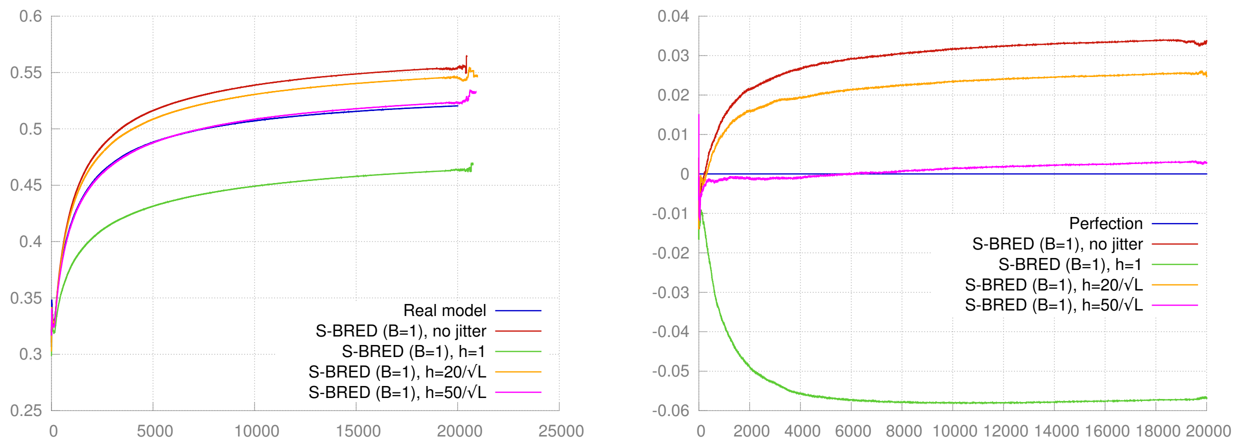
Learning curves

In the previous experiments we only studied the error of the final estimator outputted by BRED compared to the one outputted by Replay. We know from theorem 2 that the learning process is not altered when evaluated by Replay on a static environment, it is only shortened. Yet we do not have such guarantee for BRED. It is obvious that the learning process would be the same for UCB since it does take into account the contexts. We could even justify this theoretically with the bootstrap framework. Indeed we mentioned in section 5.9 that bootstrapping is about estimating the process that generates the data. In the non-parametric way that we use for BRED, this is done by sampling with replacement. When evaluating a non-contextual algorithm, the contexts can be ignored. Therefore we only have to estimate a reward function parametrized by K possible actions. Thus we can consider that the samples with replacement are an excellent approximation of the real generating process which leads to excellent evaluations. In fact this could be quantified as a function of T at the cost of a few additional assumptions. In this case note that performing a smoothed bootstrap via a KDE of the K reward functions would even be a option to increase efficiency. Therefore with a dataset of sufficient size T, we could evaluate fairly any non-contextual algorithm for any horizon. The smoothed or not bootstrap procedure would look like this:

1. ask the algorithm to choose an action,
2. sample a reward with replacement from those of the dataset that are associated with the chosen action,
3. optionally Jitter this reward (with Gaussian noise or random switches depending on their type),

4. let the algorithm learn from the reward.

and repeat the procedure to reach the chosen horizon. Note that the rejection procedure that goes along side the replay methods is in fact not necessary for non-contextual algorithms. Yet proving error/bias bounds that will obviously get better when T increases would probably require more assumptions on the reward distributions. Anyway we can be sure that with BRED, UCB has a learning curve very close to the one it would have in reality due to the excellent approximation of the reward functions made by the bootstrap resampling.



(a) Average CTR of LinUCB over time when evaluated by BRED with different Jittering parameters and on the real world. The closer to the "real world" curve, the better. The plot on the right may be better for a raw analysis of the difference but this one emphasizes the similarity of the learning curves when the right amount of Jittering is introduced.

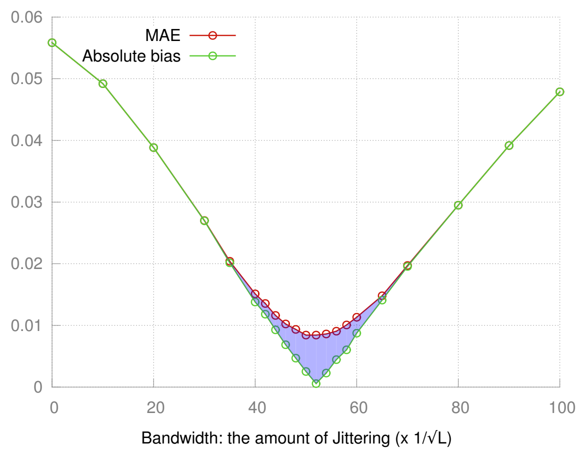
(b) Difference between the CTR of LinUCB when evaluated on the real model and when evaluated with BRED (Bias of BRED). Note that when the best Jittering is selected, the bias is very small but a little bit remains, which was to be expected considering the evaluation method based on expanded data.

Figure 5.14: Study of the similarity of the learning of LinUCB when played against the real model and evaluated with BRED. The noise when we go beyond the horizon on the first plot is due to the fact that $T^{(b)}$, the number of times LinUCB makes a recommendation when evaluated by BRED is a random variable. If we consider it as a binomial distribution, its standard deviation is 134. Therefore going as far away from it as on the plot is not common and thus, badly averaged even with as many as 1,000 runs.

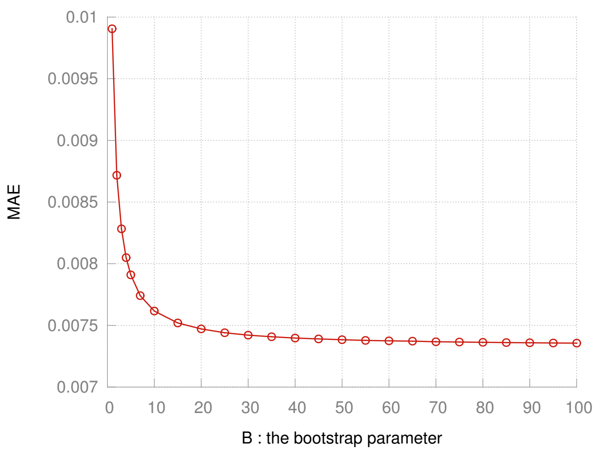
Nonetheless, when evaluating a contextual bandit algorithm such as LinUCB, we have no *a priori* guarantee. We exhibited that Jittering allows to obtain good final estimates but for all we know, the learning curve could be completely different from what happens in the real world. As an example, LinUCB could simply be learning much faster than in reality before reaching a plateau. Let us verify this with an experiment. To do so we simply repeated the previous experiment with $T = 20,000$ and various Jittering parameters. Nevertheless we plot the average CTR obtained by the algorithm all the way of an evaluation. To keep things simple, we used S-BRED with B=1 and averaged the results of evaluations on 1,000 datasets of size T generated by the synthetic model. As presented on figure 5.14, the learning curve of a LinUCB evaluated by BRED with the right amount of Jitter is very similar to the one of LinUCB evaluated on the real model. Therefore in addition to saying that BRED is a good estimator of the performance of a contextual bandit algorithm, we can go as far as to say that is also approximate rather fairly the learning process of an algorithm such as LinUCB.

Trading-off bias and... bias

We mentioned before that when BRED introduces Jitter in the sampling process, we do not trade-off bias and variance as statisticians usually do when they perform the smoothed boot-



(a) Bias (Absolute mean error) and MAE (Mean absolute error) of S-BRED (B=2) over the amount of Jittering (bandwidth). The results are averaged over 1,000 generated datasets. For information, in the same conditions the MAE and bias of the replay method on this problem is 0.090, that is 150% of the height of this plot.



(b) MAE of S-BRED when the bandwidth achieves the optimal bias ($h = \frac{52}{\sqrt{T}}$) over the bootstrap parameter B. Note that the bias obviously remains unchanged. Increasing B lets us reduce the MAE of S-BRED by approximately one quarter. Note that for reasons of limited resources, we only averaged the results on 100 generated datasets.

Figure 5.15: Bias and error of BRED on the synthetic model (dataset of size $T = 10,000$). Figure (a) clearly displays the trade-off between the two sources of bias: Jittering and overfitting. The variability of the evaluation process, is only visible through the average error when Jittering uses a bandwidth in $[\frac{35}{\sqrt{T}}, \frac{70}{\sqrt{T}}]$. This is represented by the blue area on plot (a). The rest of the time, the bias is too important and therefore trying to reduce the variance is pointless. Note that as it can be seen on figure (b), increasing B when the bandwidth is "optimal" lets us reduce the variability by a small factor. This allow us to reduce the MAE a little bit further.

strap or a kernel density estimation is general. Although BRED can be used for this purpose for static, deterministic policies (see section 5.10.1) or even when smoothing the rewards (section 5.9.3), this is not what happens with learning algorithms. Jittering is actually meant to prevent the algorithm from overfitting the data. Therefore we introduce bias, by the mean of data smoothing in order to limit another source of bias: overfitting. This process can be seen very clearly on figure 5.15 as well as the positive effect of increasing the value of B when a good bandwidth is used. As a conclusion on this set experiments on synthetic data, with the right amount of Jittering, BRED allows us to get very accurate estimations of the CTR of a contextual algorithm even with small values for B. Then, the bootstrapping procedure, if enough computational resources are available, allows us to reduce the error rate even further.

Extracted knowledge

It was proved that it is not possible to obtain convergence results for the evaluation of algorithms in general. Yet we also argued that learning algorithms are a very special kind of algorithms since they generally converge smoothly and consistently from more or less a random uniform policy toward a deterministic policy. We will not prove anything here. Nonetheless we will use the same notion of information to measure the level of knowledge contained in a dataset of size T, that is the inverse of the variance of the evaluation.

What could we expect? With static policies, we proved that the amount of information for a deterministic policy was T/K on average. Then, every stochastic choice, every possibility given a context to choose more than one action increases this amount with respect to the evaluated policy. This amount of information goes up to T or maybe sometimes even more when the evaluated policy is close enough to the logging one. It is clear that a learning algorithm can make different choices given one context. This is basically the definition of exploration. There is one additional feature that has to be taken into account in this case. Contrarily to when we evaluate stationary policies, the order of the records matters. Therefore we may assume that replaying an algorithm such as UCB twice on the same dataset but ordered differently and averaging the result will be a less variate estimator of $g_\pi(T)$ than replaying that algorithm only once. Therefore doing so - which is naturally done with all of BRED's variants via either resampling or shuffling - will allow to drag more knowledge out of the data. Given all these "sources" of information, we may even wonder if BRED, by replaying many times on one dataset, might not be able to acquire more information than T interactions with the real world.

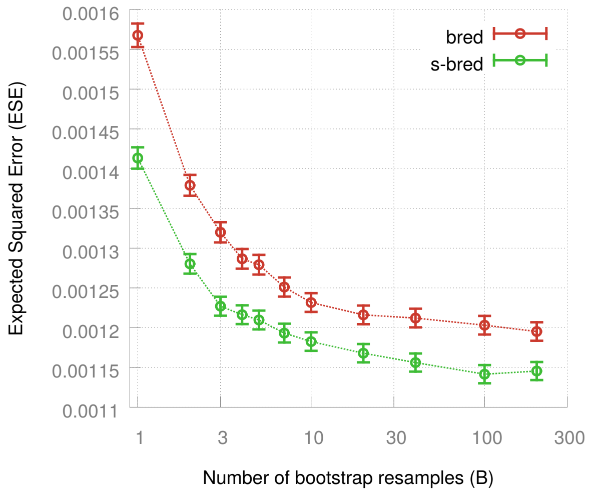
Obviously given that the evaluation can be biased (see figure 5.8) we will not consider the inverse of the variance but the inverse of the ESE as a measure of information. We computed the ESE of BRED and S-BRED as B grows so as to know the maximum amount of information extracted. We took $T = 1,000$. Remark that this experiment is rather computationally expensive because we have to perform replay B times on a dataset of size $K.T = 10,000$, and repeat the process many times to average the results. Consequently, we give on that plot confidence intervals after $100,000$ runs. As it can be seen on figure 5.16a and contrarily to what happened with stationary policies, S-BRED is clearly better than BRED even when B grows. This can be explained by the fact that it limits the number of duplicates to K which may reduce overfitting. Moreover each bootstrap resample, when replayed might be a better approximation of the real world given that the dataset is used evenly.

Finally, this experiment also teaches us that there is less knowledge in a dataset of size T than in T interactions with the world, even though we shuffle the dataset and replay it many times. Numerically, the variance of S-BRED is around $1.1\ 10^{-3}$ whereas the variance of an evaluation online is around $3.2\ 10^{-4}$ or approximately 3 times lower. It was reasonable to think that since only little data is necessary to get a good estimation of $\mathcal{D}$ when the contexts are ignored, the fact that S could be replayed in all possible orders would lead to a very precise estimation of the CTR of UCB. The estimation is precise indeed. We do manage to do much better than what is done with only one expanded replay (S-BRED, B=1). Yet it is does not get as informative as only one real world evaluation. The reason is actually very intuitive. We already argued that a learning algorithm was a stochastic policy, converging consistently and smoothly toward a deterministic policy. UCB learns very fast so most of its evaluation is the evaluation of a deterministic policy. This is why we do not do better than only one real world evaluation. Things might be different for an algorithm that takes most of its time learning. Yet such an algorithm would not be very efficient. Note however that contrarily to the evaluation of a purely deterministic policy, we manage to do much better than what is achieved by only one expanded replay, much better also that what the small level of exploration might let us hope to achieve. This is obviously due to the fact that changing the order of the records helps acquiring more knowledge about UCB's learning process.

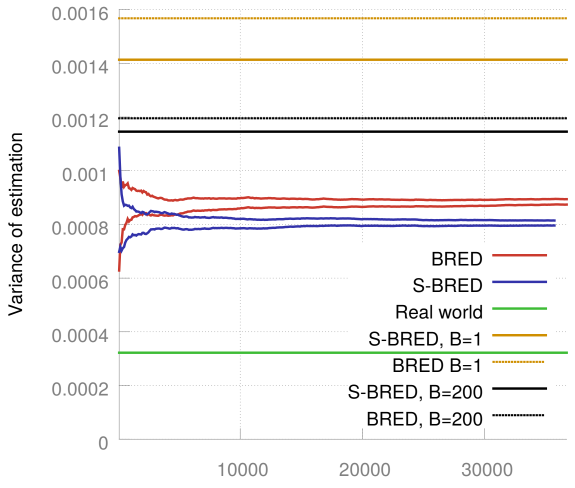
Remark Replay's expected squared error is around 6.10^{-3} or six times the variance of S-BRED.

Bootstrap

It is also interesting to wonder whether we manage to estimate $CTR_A(T)$, the distribution of $g_A(T)$ when A is a learning algorithm and not a stationary policy. We may also be interested in $CTR_A^{(S-BRED)}(T)$ since S-BRED seems to be the best estimator in that case. We kept the very same settings as the previous experiment and computed a confidence interval of the variance of the distribution outputted by both BRED and S-BRED. If learning algorithms were strictly similar to stationary policies, BRED would be estimating the distribution of replay played on a



(a) ESE of BRED and S-BRED's bagged estimate over the number of bootstrap resamples. Note that the variance of a real world evaluation is around 3.10^{-4}, which is much lower than what we achieve here.



(b) 95% confidence interval on the variance of the distribution estimated by both BRED and S-BRED, and a few baselines. Replay is not displayed for it has a variance of 6.10^{-3} which is much higher that what we get here. To our knowledge, the estimated distribution does not have a relevant meaning.

Figure 5.16: On variance and knowledge. Note that the evaluated algorithm is UCB and that $T = 1,000$.

dataset of size $K.T$, which is extremely close to the distribution of T interactions with the real world (note that R-BRED would estimate that distribution without theoretical bias). S-BRED is not really a bootstrap procedure and does not consequently estimate a distribution.

Figure 5.16b shows that BRED fails to estimate what is expected. The figure also shows that BRED does not seem to be estimating anything relevant. Actually it makes a lot of sense. Indeed, the key assumption of the bootstrap, and which is even more important with expansion is that the estimator treats the records independently. Indeed, if it is not the case, $\hat{F}(S)$ is not a very good estimator of F since it does not approximate the independence between all the records. We necessarily get duplicates. Therefore for an evaluation of a learning algorithm, it can not work.

Nevertheless, it is important to understand that this is not crucial. Indeed, the key problem that we wanted to solve was time acceleration. We wanted to be able to simulate more interactions than what replay allows so as to compare more fairly various algorithms that handle the exploration *vs.* exploitation dilemma. In fact $g_A(T)$ is only important relatively to other algorithms on the dataset. Consequently its distribution in the real world in not important either. This is even more true in applications such as news recommendation in which the CTR of the best news can change a lot from one day to another. Therefore the raw value of g_A is irrelevant. What matters however is the behavior of the algorithm: how fast it learns, if it can follow a sub-optimal path, does it explore too much... All these things can be monitored and although the various replays that can be computed by reordering will not approximate perfectly the distribution of live evaluations, we may observe these behaviors. Finally it seems possible to compute $CTR_{\pi}^{(S-BRED)}(T)$. To do so it would be enough to bootstrap S-BRED, that is resampling datasets S_i of size T and computing S-BRED on each of them. Contrarily to RED∞, that could be reduced this procedure needs to let the algorithm interact all the way with the data. Consequently such a procedure would be quite computationally expensive (many bootstrap resamples times the B parameter of S-BRED times $K.T$ records). In addition this may not be very useful since we are after comparing algorithms and it is very likely that reasonable algorithms would have a similar variance of estimation. The only reason to do it would be to quantify how sure we are that an algorithm is better than another.

Remark that all this completely differs to what was wanted with iterative improvements of a stationary policy in classical recommendation. In this case, we are interested in the raw value of g_π and that is it. A confidence interval on a raw value is critical.

5.10.3 Bandits on real data

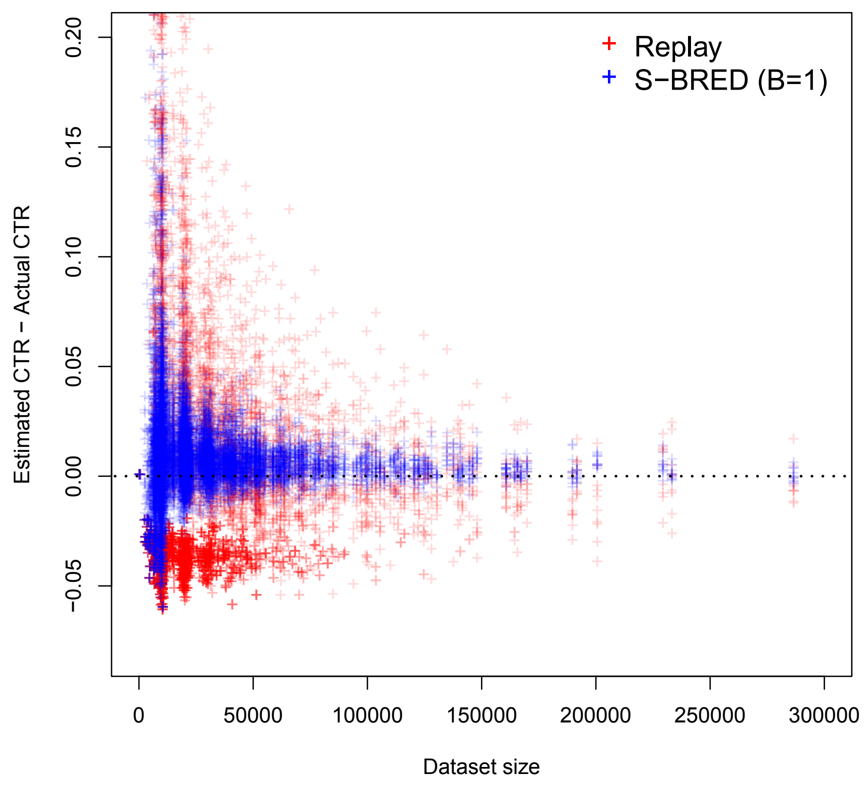


Figure 5.17: (Non-absolute) Error of S-BRED (B=1) on the various zones of the Yahoo! R6B dataset [12] as described on figure 4.16. Closer to zero is better. This plot clearly shows that S-BRED produces results that are much more concentrated around zero. Yet for more evidence supporting this claim, see figure 5.18. There is another important detail in this cloud of points: when T grows larger, the estimates produced by replay tend to be fairly centered around zero. Nevertheless, when T is small, there is a great concentration of red points (corresponding to replay) way bellow zero. In fact this means that as we argued it all along this chapter, replay underestimates the CTR of learning algorithms because of time acceleration. This is not the case with S-BRED. It is not that accurate for very small horizons (although still better than replay) but it is always more or less centered which is a sign of a very low bias.

The practical use of BRED as well as its superiority compared to the state-of-the-art approach was clearly demonstrated using synthetic data. Yet one may argue that this can not be completely satisfying until nice performance are also exhibited on real data. Nevertheless achieving this is complicated for several reasons, especially without having access to a real application. The main reason is that it is impossible to know the ground truth and therefore evaluating a evaluation method is complicated. The fact that a real dataset was not generated by static processes (the environment is dynamic) is also problematic for a bootstrapping procedure for we can only consider small parts of datasets.

Nonetheless we pulled off a small trick to manage such an evaluation using the Yahoo! R6B dataset [12]. This trick allows us to run an experiment very similar to one that was run in one of the articles about the replay method which was about evaluating UCB [11]. What they did is that they measured the error of the estimated CTR of UCB ($\alpha = 1$) by the replay method on datasets of various sizes relatively to what they call the ground truth: an evaluation of the same algorithm on a real fraction of the audience. As we obviously cannot do that, we used a simple trick: we divided the Yahoo! R6B dataset [12] into the minimum number of batches such that within a batch, the action set is static. These batches have a maximum size of 300,000 which corresponds to 3 or 4 hours. Therefore we can reasonably assume stationarity. See figure 4.16 for a more precise reminder of how these datasets were built.

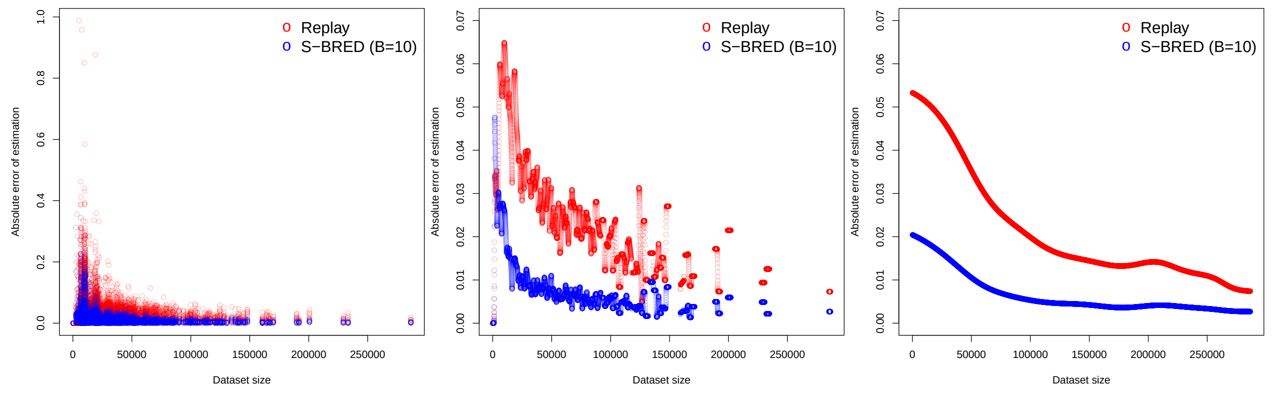
For each batch, we computed a ground truth by averaging the estimated CTR of the algorithm using the replay method on 1,000 random permutations of the data of the batch. Note that the CTR of an algorithm estimated via the replay method on a dataset of size T is an almost unbiased estimate of the CTR of this algorithm over T/K real time steps (K being the size of the action set). This is why for each batch, we subsampled uniformly T/K events and evaluated the algorithm using the replay method and BRED on this smaller dataset of size T/K and for which a ground truth is available. Note that we used $B = 10$ and no Jitter as this is useless for non-contextual algorithms. For each dataset we also repeated the subsampling followed by the evaluation using S-BRED and replay ten times. Indeed, various subsamples of T/K events out of the same T events can be very different from one another.

Remark Note that we evaluated a non-contextual algorithm in this section which may seem counter-intuitive given our primary purpose. We do agree that this is not perfect. Yet we highlighted in chapter 4 that the evaluation of contextual algorithms and LinUCB in particular was very biased on the R6B dataset due to time acceleration. Therefore since this experimental protocol makes us use even less data than in the challenge, trying to evaluate LinUCB was hopeless.

We plotted the results corresponding to this experiment under different perspectives. Figure 5.17 shows the raw results, that is a cloud of points corresponding to the (non-absolute) error of estimation made by both methods. We find this figure very representative of what is happening and highly informative. More details are available on the figure. Yet one may argue such a cloud not to be conclusive enough. Thus as it can be seen on figure 5.18, we also plotted the absolute error and applied different Gaussian convolutions in order to average the results locally. Although the result provides a smaller range of information, the fact the S-BRED outperforms Replay becomes crystal clear.

5.11 Towards using BRED on real datasets

This section tackles two issues that arise with real data. First, when simplifying the problem of time acceleration of a dynamic process to a static bandit distribution with a fixed horizon, we argued that all the solutions to the simple problem would apply to the real one. In section 5.11.1, we show that this is as straightforward as announced and describe the a simple modification of BRED that allows to handle dynamic processes. The second issue is the choice of the bandwidth of the Jittering Kernel which is not as simple to solve as the first one. Nevertheless, we describe in section 5.11.2 a brand new method called *entangled validation* that bypasses the problem. More details and experiments on this extremely promising technique are also provided in appendix A. Although very little is said in the main body of this document about entangled validation, we do believe this technique to be a major contribution. Moreover, the perspective to both study this technique further, apply it to other problems and use it on real data is a very exciting perspective that unfortunately, we have not had time to include in this



(a) No convolution. (b) Convolution with a Gaussian Kernel of bandwidth 1,000. (c) Convolution with a Gaussian Kernel of bandwidth 50,000.

Figure 5.18: Absolute error of S-BRED (B=10) and replay on the various zones of the Yahoo! R6B dataset [12] as described on figure 4.16. The results are averaged (convoluted) locally using a Gaussian Kernel. Lower is better. The curves speak for themselves but one little additional detail is worth discussing. When the bandwidth is equal to 1,000, we see that there is no error with very small datasets. Then the error made by S-BRED increases faster than the one made by replay which sounds surprising. This is because when the dataset is very small, the CTR found by a replay method including the one found by the ground truth replay is small or even zero. Therefore an evaluation method that always outputs zero has low error. At some point when there is a little bit more data, the method becomes very variate as one click found completely changes the value of the estimate. Then where more data starts becoming available, the error rate drops again progressively. S-BRED only follows this process much faster than replay (see that the error then goes down faster too) as it tries to make the most with the data at hand.

thesis work. We end this section by remarks on BRED's complexity (sec. 5.11.3) and on the use of expansion in real life situations (sec. 5.11.4).

5.11.1 Handling datasets generated from dynamic processes

As we mentioned it in the previous section, running BRED on an entire dataset from a real application is not as straightforward as running replay. The fact that the dataset depicts a dynamic process, with items that come and go and possibly other sources of evolution makes seem the use of bootstrapping a little odd. Indeed, bootstrapping makes the natural assumption that the generating process is stationary and approximate it non-parametrically using bootstrap resamples. Note that so many things can evolve in a recommendation application that using BRED on a full dataset is doomed to fail. See Agarwal *et al.* [114] or the description of a few features of the Yahoo! R6B dataset [12] in section 4.2.3 for more details. The ideal way to solve this would be to be able to estimate the process that generated each single event. This is obviously impossible and approximation are necessary.

Two ideas can be envisioned. To work, BRED needs an approximation of the process that generates the data. It is done via sampling from a Kernel Density Estimation, or in other word via resampling from the data and adding Jitter. Kernel density estimation was also studied for time series [139]. Such ideas could be used to smooth the dataset and allow BRED to work on a big dataset exactly as described in this chapter. A simple way to sample from a time series density estimation at step t of a BRED evaluation would be to sample an event from a discrete distribution, centered around the t^{th} record of the dataset and then add Jitter. One would certainly be willing to give bigger chances of being drawn to records that are closer to the t^{th} one. This is the simplest way. Obviously many more method can be envisioned depending on the assumptions that seem reasonable for a given dataset. Studying this is left as future work. Yet the interested reader could refer to Robinson [139] for leads on the subject that are based on theoretically justified KDE.

The other idea to deal with dynamic datasets is much coarser but happens to have a nice property. The basic idea is simply to say that the environment does not change that much within one or two hours. Therefore one can perform BRED on consecutive windows of the dataset, assuming BRED to be dealing with a static process in each window which makes the bootstrap resample a nice approximation of the real world. From there the bootstrap aggregation, parametrized by the number of bootstrap resamples B, can be done in two ways:

1. either use BRED with a bootstrap parameter equal to 1 on each window and repeat the process B times

2. or use BRED with a bootstrap parameter equal to B on each window.

The first method gives a bagged estimate of the CTR of the algorithm on the entire dataset as well as an estimation of its density (if the algorithm is stationary). Note that this is also possible when approximating the dataset with KDE for time series. The second method allow us to have bagged estimates and estimations of their densities all along the evaluation of the dataset. This can provide very helpful information. Nonetheless note that this method requires a little bit more engineering effort. Indeed, when bootstrapping the w^{th} window, it is necessary to copy the state of the algorithm at the beginning of the window and use that state at the beginning of each bootstrap evaluation. Moreover it is necessary to define a way to average all the different states in which the algorithm ends the bootstrap evaluations of the data. In general algorithms are based on an estimation of a parameter vector that describes the environment. Averaging them is usually fairly easy but the process may turn out to be difficult depending on the type of algorithm used. Algorithm 16 is an implementation of this second methods that provides density estimations of the CTR all along the dataset. Remark however that this technique could bias the evaluation of the algorithm as it is common knowledge that bagged predictors tend to be more accurate when randomness is involved [119] and therefore the learning algorithm may achieve better performance when bootstrapped on each window.

Note that with all these methods, a key parameter has to be chosen. We can call it the bandwidth of the data as it determines how much it will be smoothed. For the time series kernel approach, it is the bandwidth of the kernel, that is the width of the discrete distribution from which the records are sampled at each time step. For the window approach, the bandwidth is simply the size of the windows. In any case, this parameter is of utmost importance. If it is too big the data will be very smoothed and all the dynamic characteristics will be erased, introducing bias. If the bandwidth is too small, the approximation will be very noisy as it will use too little data, introducing a lot of variance. Thus choosing that parameter is another bias/variance dilemma.

5.11.2 Entangled validation

Let us now discuss how to tackle the issue of the bandwidth of the Jittering Kernel. We argued that the right amount of Jittering should be picked for BRED to work properly. Consequently, finding a way to do so is crucial. Here, with synthetic data, we had access to the ground truth so it was easy to know when we reached it. How can we do it in general? Many data-driven ways exist in the literature to select the bandwidth in the context of Kernel Density Estimation. A consensus exists on the fact that plug-in selectors [140] but mainly cross-validation techniques [141, 142] are the most reliable way to do so. When evaluating stationary policies, it is very

Algorithm 16 BRED for a dataset generated by a dynamic process.

Input

- A (contextual) bandit algorithm A. Here we consider that A is no longer a function of the history but a function of a real vector that represents this history.
- A function Θ that maps an event (x, a, r) and a prior vector θ to a new vector θ modeling the history.
- θ_0, θ's initial value.
- A set $\mathcal{S}$ of T triplets (x, a, r), divided in W windows $w_1, w_2..., w_W$ of size T_{w_i}.
- An integer B.

Output: A bagged estimate $\widehat{g_A}$ of the performance of A and a list of bagged estimate $\widehat{g_{A,1}}...\widehat{g_{A,W}}$ of the performance of A on each window.

```
$\widehat{G}_{0,A} \leftarrow 0$, $T_0 \leftarrow 0$
for $i \in \{1..W\}$ do
    for $b \in \{1..B\}$ do
        $T_i^{(b)} \leftarrow 0$, $G_i^{(b)} \leftarrow 0$
        /* Here we want the CTR of $A$ on each window so we completely reset the estimator.
        By setting $T_i^{(b)}$ and $G_i^{(b)}$ to $T_{i-1}^{(b)}$ and $G_{i-1}^{(b)}$ instead, we would get the evolution of the
        estimation of $g_A(T)$, window after window. Given the dynamicity of the problem,
        this would diminish our ability to measure how well we adapt to changes. */
        $\theta_i^{(b)} \leftarrow \theta_{i-1}$
        for $i \in \{1..T_{w_i} * K\}$ do
            Sample with replacement an element $(x, a, r)$ of $w_i$
            $x \leftarrow \text{JITTER}\left(x, h = O\left(\frac{1}{\sqrt{T_{W_i}}}\right)\right)$
            $\pi \leftarrow A(\theta_i^{(b)})$
            if $\pi(x) = a$ then
                $\theta_i^{(b)} \leftarrow \Theta\left(\theta_i^{(b)}, (x, a, r)\right)$
                $G_i^{(b)} \leftarrow G_i^{(b)} + r$
                $T_i^{(b)} \leftarrow T_i^{(b)} + 1$
            else
                /* Do nothing. */
            end if
        end for
    end for
    $\theta_i \leftarrow \frac{1}{B}\sum_{b=1}^{B}\theta_i^{(b)}$
    $\widehat{g_{A,i}} \leftarrow \frac{\sum_{b=1}^{B} G_i^{(b)}}{\sum_{b=1}^{B} T_i^{(b)}}$
end for
$\widehat{g_A} \leftarrow \frac{\sum_{i=1}^{W}\sum_{b=1}^{B} G_i^{(b)}}{\sum_{i=1}^{W}\sum_{b=1}^{B} T_i^{(b)}}$
return $\widehat{g_A}$ and $\widehat{g_{A,1}}, ...\widehat{g_{A,W}}$
```

likely that we could take advantage of such techniques. Nevertheless, when evaluating a learning algorithm, BRED is not only about approximating the data but aims at limiting overfitting. Also, as exhibited in section 5.10.2, the optimal Jittering parameter is a decreasing function of the size of the dataset; most likely in $\Theta\left(\frac{1}{\sqrt{T}}\right)$.

One could think of using cross validation to select the Jittering parameter, the same way as in the literature. Yet, the fact that the optimal parameter is an unknown function of T, and potentially different for every problem makes this impossible. Indeed we could divide the dataset into two part and perform replay on a part K times bigger than the other. Then we could use the remaining data to adjust the parameter so that BRED outputs the same thing as replay (the validation value). Yet we would obtain the optimal Jittering parameter for a dataset of size $\frac{T}{K+1}$ which is clearly not what we want. In appendix A.2, we show that the cross validation method does not provide satisfying results even when using a rough knowledge (from experiments on the synthetic data) of the function that maps T to the optimal Jittering parameter in order to refine the parameter we find with the previous method so that it corresponds to a dataset of size T and not $\frac{T}{K+1}$. Yet this method has an advantage. We know that the optimal Jittering parameter for a bigger dataset has to be smaller (if we do not try to infer it with an estimation of the aforementioned function). Therefore using this overestimation of the parameter prevents any positive bias (overfitting). Indeed, let us recall that we want to minimize the risk of deteriorating the system when using a new algorithm. A non-quantifiable positive bias is therefore unacceptable for it would make the risk explode. Although a negative bias is not desirable either, it prevents us from being fooled by overly optimistic evaluations. Yet a too big negative bias such as the one of this method would probably prevent us from detecting any interesting improvement.

The solution we propose is to use a completely novel (to the best of our knowledge) and much more subtle approach that we call *entangled validation*. The idea is similar to what we just exposed: prohibiting any positive bias via a validation mechanism. Yet this technique is very likely to yield a much smaller negative bias. The approach is quite simple and resembles cross-validation in spirit. At each iteration, we divide the dataset randomly into two datasets: S_{test} of size $\tau.T$ and S_{train} of size $(1-\tau).T$. Typically, S_{train} is much larger than the other one (*e.g.* $\tau = 0.1$). In a nutshell, we then expand S_{train} but not S_{test}, entangle the generated records and perform replay on the result by only taking into account the records from S_{test} to compute the estimator. Thus the algorithm learns all the way, can overfit depending on the Jittering parameter but only on the records that are not taken into account by the estimator. More specifically, we generate a set I_{test} by sampling without replacement $\tau.T$ integers from $\{1, 2, ..., K.T\}$. Then we perform replay on $K.T$ records such that at iteration i, if $i \in I_{test}$, the replayed record is sampled *without* replacement from S_{test} (without Jitter because the records are only used once). Otherwise it is sampled *with* replacement from S_{train} (with Jitter). The resulting estimator, that we call Tested-BRED (or T-BRED for short) is finally computed without taking into account the training data that enables overfitting:

$$\hat{g}_A^{T-BRED}(S) = \frac{\sum_{i=1}^{K.T} I(i \in I_{test}) I\left(a_i = A(x_i)\right) r_i}{\sum_{i=1}^{K.T} I(i \in I_{test}) I\left(a_i = A(x_i)\right)} .$$

Obviously, because the number of records taken into account is reduced, one should average several realizations of the validation process to obtain a precise estimation. Notice that contrarily to the cross-validation method, we do not prevent overfitting. We simply prevent the estimator from being fooled by it.

As detailed and shown in appendix A, T-BRED has several very nice properties. First it very simple to implement and to understand. Second it cannot have a positive bias which is what we want to avoid the most and even allows to quantify overfitting by comparing how the algorithm behave on duplicated data compared to the validation data. Consequently, the optimal amount of Jitter is simply the parameter that allows the most optimistic estimation. Third, it does not introduce additional variance (compared to S-BRED) as long as we average realizations until convergence. Finally it allows to enrich the estimation of the bandit distribution with models without fear of positive bias as long as the validation data is not altered.

We believe entangled validation to have the potential to be a major breakthrough in the context of online learning evaluation (and not only for contextual bandits). Much more on the subject (experiments, detailed implementation, arguments justifying the aforementioned properties, discussions, applications *etc.*) is available in appendix A. Nevertheless, although this work is much more advanced than a mere perspective that can be put at the end of a chapter, it is still quite immature. Indeed although it can be used in practice just the way it is, a lot of work remains to perfect the method and study its empirical properties. For instance we do not have yet results of the evaluation of contextual bandit algorithms on the Yahoo data [12] used during the challenge we organized. Also the fact that the estimated bandit distribution (by bootstrap), is less informative than the real one is problematic fir the algorithm's, learning process. We provide many interesting leads to tackle the issue in appendix A.5 but we did not study those leads both theoretically and empirically. This is why the main part of this early work is only in appendix. Nonetheless, the reader interested in building a dynamic recommender system should definitely go through the entire appendix A which is without doubt the most important one. Indeed despite the necessary research that remains to be done, this appendix already contains ready-to-use method that seems very promising in the view of the experiments we were able to run.

5.11.3 Remark on complexity

This is probably obvious but let us discuss BRED's time complexity. For a dataset of size T, replay requires the algorithm to make T choices and $\frac{T}{K}$ updates (in average). BRED requires the algorithm to make KT choices and T updates (in average) per bootstrap evaluation. Therefore these values have to be multiplied by B. This may be prohibitive in some cases. As an example, when studying the impact of B on the evaluation of LinUCB on datasets of size $10,000$ with $K = 10$, we first tried to average $1,000$ runs (see figure 5.15b). For $B = 20$ it took 10 hours. In order to be able to try greater values for B, we went down to only 100 runs.

For stationary policies only, a trick can be used to significantly reduce that complexity. The idea is to use a Poisson distribution to approximate subsampling with replacement, so that one only needs to scan through the dataset *once* to compute all bootstrap resamples. The trick is used in the BLB article [113] and implemented in Vowpal Wabbit (VW) [143]. See Hanley and MacGibbon [144] for a full introduction to the method.

5.11.4 Remark on data expansion

The remark on time complexity leads to another one. In this chapter we assumed that we needed to expand the data by a factor K in order to achieve minimum bias. Indeed, this way the algorithm roughly sees the same amount of events as it would in reality. Yet if the algorithm does not need that long to reach a stationary regime, it could be envisioned to expand the data by a smaller factor E without introducing too much bias. This would allow us to save computational resources that could be used to increase B and reduce the variance of BRED's evaluation. This other bias/variance dilemma may be less crucial than the others we presented in this chapter but it exists and taking an interest in it may improve the evaluation if the computational resources are limited. Studying this is also left as future work.

5.12 What to remember in a few words

To wrap up, here is what to remember from this work. The contribution of this chapter is a set of methods (in particular BRED and RED∞) that significantly increase the accuracy of the offline evaluation of both learning algorithms and stationary policies.

For stationary policies, accuracy is increased by three factors:

1. considering a confidence interval (which is more informative anyway) instead of the mean allows to gain an order of convergence: from $O\left(\frac{1}{T}\right)$ to $O\left(\frac{1}{T}\right)$ (see table 5.1 for a recap of the various methods and their respective range of utility),

2. random policies are more accurately evaluated (see variance analysis: section 5.8),

3. Jittering allows to evaluate more accurately both random and deterministic policies because it introduces the possibility to make different choices given one context (see section 5.9 but more importantly section 5.10.1).

A last factor which that could be considered is the use of iterative improvements to improve the accuracy of the entire process of construction of a recommender system.

For learning algorithms, we are not able to derive confidence interval (at least for now), yet we improve their accuracy by reducing both the bias and the variance of the evaluation.

Bias reduction: Expansion reduces time acceleration, a great bias exhibited in chapter 4. It introduces another bias due to overfitting which is controlled by the introduction of Jittering. This bias reduction was exhibited empirically on both real and synthetic data (section 5.10).

Variance reduction: Similarly to what happens with stationary policies, whenever several choices are possible given one context, BRED provides accuracy lifts compared to classical replay methods. Multiple factors make this happen systematically:

1. Jittering, for the same reason as with stationary policies,

2. the exploration process can typically make different choices in one context and take advantage of them,

3. time is contextual: from one expanded replay to another, the order of the records changes. Meeting a record, the algorithm might be at any stage of its learning process and make different choices.

Note that for learning algorithms, S-BRED is the preferred method for it seems to give better results. Also evaluating learning algorithms with non uniform logging policies is challenging at best so we strongly encourage people who want to do that to make the effort and try to acquire uniform data. For stationary policies non uniform logging policies are perfectly fine. The reader should refer to table 5.2 that recapitulates the four methods we introduced - adjusted to handle any non-uniform logging policy - and keep in mind that if RED*∞ is the methods we analyzed in greater depth, RED∞ is generally more accurate although the study of its variance is left for future work.

5.13 Conclusion and perspectives

In this chapter, we presented the main contribution of this thesis work. This contribution is a data-driven method, BRED, based on what is called bootstrapping in statistics and bagging in machine learning. This new method allows to evaluate online learning algorithms with a much

Table 5.2: Table of the estimators of g_π when π is stationary (for any arbitrary stationary logging policy π_{LOG} that, for each context, plays all the actions π plays) with the notations $w_t = p(\pi_{LOG}(x_t) = a_t)$ and $p_t = p(\pi(x_t) = a_t)$. Note that the compressed form as proposed here does not allow to introduce Jittering and thus to take advantage of the variance reduction. Nonetheless a specific compressed form could be computed analytically for any Kernel. The raw form of the estimator works without modification regardless of the Jittering technique.

Method	Estimator	Compressed form	Properties
Replay	$\frac{\sum_{t=1}^{T} V_t.r_t.w_t^{-1}}{\sum_{t=1}^{T} V_t.w_t^{-1}}$	-	Original estimator [11].
Replay*	$\frac{1}{T}\sum_{t=1}^{T} V_t.r_t.w_t^{-1}$	-	Unbiased version of replay. Slightly more variate with reasonable values of K and T. Better when $K \approx T$.
RED*∞	$\frac{1}{T.E}\sum_{e=1}^{E}\sum_{t=1}^{T} V_{t,e}.r_t.w_t^{-1}$	$\frac{1}{T}\sum_{t=1}^{T} p_t.r_t.w_t^{-1}$	Unbiased. Always less or equally variate as Replay*: better with stochastic policies and/or with Jittering.
RED∞, the best method in general	$\frac{\sum_{e=1}^{E}\sum_{t=1}^{T} V_{t,e}.r_t.w_t^{-1}}{\sum_{e=1}^{E}\sum_{t=1}^{T} V_{t,e}.w_t^{-1}}$	$\frac{\sum_{t=1}^{T} p_t.r_t.w_t^{-1}}{\sum_{t=1}^{T} p_t.w_t^{-1}}$	Biased version of RED*∞. Always less or equally variate as replay. Variance and bias not analyzed (future work) but intuitively less variate than RED*∞ for reasonable K and T.

higher accuracy than anything that was previously proposed in the literature, by reducing the effect of time acceleration and dragging more knowledge out of the data. As a side effect of using bootstrapping, we even get an estimation of the distribution of the CTR when played against the real world. We may not have stressed it enough in the body of this chapter but this property is very nice when the goal is to minimize (or at least quantify) the risk of putting online a new algorithm. Our main goal at the beginning of this work was to reduce time acceleration for learning algorithm. In addition to that, we made several interesting contributions with regard to the evaluation of static policies. This is something that should not be overlooked since in recommendation, many sources of knowledge are available such as experts or content based techniques (see section 1.3.2). Such approaches are usually evaluated via prediction accuracy measures. Yet, if $\frac{T}{K}$ is big enough, replay methodologies could be a very interesting and different perspective to measure performance. In the context of pure evaluation, we exhibited the improvements yielded by BRED when evaluating deterministic policies, thanks to smoothing. Also, even when $\frac{T}{K}$ may seem relatively small, BRED comes with a very accurate estimation of the distribution of the estimator (that converges faster than the estimation of the mean). Therefore if the evaluation in inaccurate, one is at least warned.

The main contribution of BRED regarding static policies is how it can be used to perform iterative improvements of a recommender system. By doing so, we showed that we could get convergence rates that are very close to $\sqrt{1/T}$. Because it was not the first goal of this work, we did not give a lot of details. Yet we do believe that this procedure could be used in a broad range of recommendation applications. It would allow to introduce an unbiased CTR metric, and to try to make the system maximize its result. Notice that achieving such a feat would be quite interesting given that recommendation is usually evaluated via indirect metrics such as rating prediction metrics, ranking metrics *etc.*. A metric based on CTR (or any other online metric) is a direct measure of something really representative of the performance of a recommender system. Moreover, it would be very interesting to investigate the matter much further. As it was shown on figure 5.10, iterative improvements are in fact a learning process that could be entirely automatized so as to maximize the CTR, while minimizing the risk of deteriorating user experience too much along the way using the bootstrap. Indeed we proved two key features of the process: (i) the tremendous accuracy that can be achieved thanks to BRED and the fact that the evaluated policy is close to the logging one, (ii) the fact that choosing the logging policy when we know the policy to evaluate is a convex (thus simple) optimization problem. We could therefore envision an algorithm, deciding at each improvement step how it would tweak the current policy to either try to improve it or explore different areas so as to make future improvements. This algorithm could plan the next step while playing the current policy using the output it receives and tweak online the randomization of the live policy so as to acquire more knowledge on the future policy it will consider putting live. The only drawback about this approach if there is one is that it can only be used by companies that own a recommender system. Indeed, it is completely unthinkable to the use state-of-the-art datasets that are used in the literature to try this iterative procedure. The main reason for that is obviously that we could only get precise estimates for policies close to the logging one, which is hardly interesting. Yet one may think of plenty other reasons such as the fact that the logging policy may not be known (although this is addressed by Strehl *et al.* [97] but further reduces accuracy), the fact that the iterative process could not be faked *etc.* A first step would be to try this procedure on a synthetic model, to at least see how things work. In particular, given that the approach is greedy by nature (we improve by going to the most appealing neighboring area), it would be interesting to consider how we could avoid local optima. An interesting thought with that regard is that although the procedure prevents to change the current policy to another that is not in its neighborhood, nothing prevents to look further. We can then explore that direction over several steps, getting closer and closer if the promises are confirmed, or on the contrary, dropping the idea if we were mistaken. Nevertheless one should always keep in mind that a study based on synthetic data could only be a solution to address general problems such as the one we mentioned. Model based evaluations are rightfully not trusted in the recommendation field because they are generally way too biased. Therefore they should not be used (at least by themselves) to design a recommender system meant for a real application.

Nevertheless, our first goal was to make things better for online learning algorithms. Although the theoretical framework does not allow to give formal guarantees in this context, we provided plethora of intuition justifying the use of BRED with learning algorithms. Our empirical study then demonstrated that our methods works pretty well for at least the most wide spread type of algorithm: UCB-like algorithms (UCB and LinUCB in particular). In particular, BRED does not only solve the time acceleration issue, it also enables a very significant variance reduction of the evaluations.

Finally this new method will probably allow much more people to work on dynamic recommendation and evaluate algorithms. In spite of the direct applicability of the methods we propose, this work still offers many perspectives of exploration and improvements. Considering non-contextual bandit algorithms, we justified that it was almost possible to expand the dataset as much as we want so as to let it learn for as long as necessary and get really accurate estimations in any situation. It is so because there is generally enough data to infer an almost unbiased model if the contexts are ignored. For contextual algorithms, things are obviously a little bit more constrained and more work would be necessary to fully understand what is possible and what is not. Here we showed that it works, given that the right amount of Jittering is added, for LinUCB, a smooth learner based on least square regression. What would happen with algorithms that explore and then exploit without smooth transition? What would happen with algorithms based on trees or neural networks? Would random switches be an efficient way to perform Jittering when the context is discrete? All these questions are left open and would be very interesting to investigate.

Two other distinct perspectives of investigation are worth being mentioned. The first one is purely theoretical. It is about smoothly learning algorithms such as LinUCB or even the ones we derived, LogR-UCB (section 4.4.1) and LAP (algorithm 8). We may also consider non-contextual algorithms such as UCB or Thompson sampling. Their behavior can be mathematically characterized as smooth [7,13]. We argued that although deriving convergence results in the general case is impossible, learning algorithms are a particular case for which it would be possible. Li *et al.* [11] made a similar argument. It would be very interesting to study the additional assumptions that are necessary and the effect on the bounds. The second line of future work is highly practical and is about the iterative improvement procedure. It would be very interesting to try to design a more complete procedure that chooses the next policy to put online given the previous evaluation. Also, the randomization could be tweaked online if interesting areas to explore with more care are identified. Obviously this would require more thought and a lot of work but the idea is very tempting. This idea also offers theoretical perspectives. Indeed the problem could easily be modeled mathematically and studied so as to find optimal solutions in various settings.

Before moving on to the perspective we actually study in this thesis work, let us talk about more distant applications. Clearly, we introduced BRED to improve the evaluation contextual bandit algorithms, which it does nicely. Then we realized that it could apply to the design of regular recommender systems. The perspectives and applications we mentioned are all related to these two contexts. It is very likely that BRED could be used in different contexts. Let us mention two of them. For instance BRED could simply improve online evaluation. Replaying an algorithm/policy put online would allow it to make different choices and improve the evaluation. Consequently, less time would be needed to get a fair estimation of the performance of a new approach. We could also think about AB testing [72] that we already mentioned. As a reminder, the idea is to divide the audience randomly and to assign different values of a parameter in order to achieve unbiased comparisons. Let us consider the simple case of two distinct values: A and B. We could consider using BRED to evaluate the policy parametrized by A on the data logged by the policy parametrized by B (and vice versa). This would definitely increase the accuracy.

Finally, with regard to learning algorithms many directions remain unexplored. The main one is to pursue our work on entangled validation (appendix A) which is already advanced enough to be used in practice but that could be studied further. In particular studying how T-BRED behaves on real data would be very interesting. In any case, we strongly encourage the reader to have a look at that particular appendix for the results and promises it displays. Anyhow, although there is still work to do, we have provided a set of methods that is already fully functional for both stationary policies and learning algorithms. Furthermore, we justified significant improvements in terms of both bias *and* variance with respect to the state of the art and so for both cases. As a consequence, our conviction is that the main and most exciting perspective to this work is simply to use it to design recommender systems that are better at providing you with the content you need at the time you need it.

Conclusion

Abstract *This section is the general conclusion of this thesis work. It articulates around three main components:*

1. *the aftermath, that is a summary of the main contributions whose significance is highlighted with respect to the state of the art,*
2. *a few horizontal themes that we came across along the course of this research work,*
3. *the perspectives for both future work and applications.*

1. Aftermath

In summation, this thesis work is about the use of contextual bandit algorithms in the context of a specific range of recommendation problems: dynamic recommendation. In particular, we mainly considered an issue which is universally recognized to be a critical issue: the offline evaluation of such recommender systems, before putting them online. Dynamic recommendation is in our opinion the main application of contextual bandits nowadays and the one that provides the most data. Moreover it is also probably the most meaningful to many readers, hence our choice to use that application to illustrate this work all along this document. Nonetheless, throughout the totality of this work, we paid attention to maintain a certain level of generality so that all that is said in this document can be applied to any other application of (contextual) bandits the reader might think of. Furthermore, we showed at length (and in particular in chapter 5) that the scope of this work ended up going way beyond dynamic recommendation. By studying the evaluation of stationary policies as well, we provided methods that allow to measure the ultimate metric of any recommender system: the CTR (or any immediate reward) as well as solid arguments proving there usability in practice. This contrasts with the current trend in the field which is to systematically consider indirect metrics such as the MAE on rating prediction and could be the contributions that are going to have the greatest impact in the community.

A first contribution was made during the background review. Online learning and its formalization as contextual bandits are not considered as standard techniques to deal with dynamic recommendation. Therefore the first contribution of this work was to motivate that it should definitely be otherwise. Indeed, we exhibited that online learning techniques are definitely helpful and deal successfully with unresolved issues in the context of dynamic recommendation. Then the various contributions of this work are divided into three sets. The first one is about contextual bandit algorithms themselves, that are studied in general and in the context of dynamic recommendation. The second set is the thorough empirical and theoretical study of the replay method, a technique to evaluate such algorithms on real data. Finally the last set of contributions tackles the main flaw of replay: time acceleration. The last set gathered the main and most original contributions of this thesis work that are based on data expansion, bootstrapping, information quantification and entangled validation. Let us come back on these various contributions in more details.

The first set of contributions, which is split between chapter 2 and 4, improves the state of the art with regard to bandit algorithms themselves. We first motivated their use, in particular in the context of recommendation. Such approaches obviously cannot replace classical methods such as collaborative filtering. Yet, as justified in section 1.4.1 they definitely have their place in the recommendation literature for being a very natural way to tackle dynamic recommendation. The first contextual bandit algorithm we studied was built upon Microsoft's Ad predictor [6]. This new algorithm, that we called *Generalized Ad Predictor (GAP)*, has the major advantage of being able to deal with both discrete and continuous (linear) contexts, as more complex functions that are approximated linearly and by parts. Note that in the literature, the two main contextual bandit algorithms could only deal with either linear (LinUCB [7]) or discrete features (Ad Predictor). The significance of this new algorithm was established by its victory in the Exploration *vs.* Exploitation challenge 2 (2011). Its other strength, especially considering responsiveness requirements of web applications is its rapidity. Its time complexity is linear whereas LinUCB's is cubic! The intuition behind this huge difference is that GAP updates its model using approximations whereas LinUCB simply recomputes everything exactly from scratch at each time step. We finally ran experiments that exhibit that GAP performs better than both a straightforward way to approximate model updates based on the Perceptron [88] and the straightforward way to reduce LinUCB's time complexity proposed by its authors. The last set of contributions regarding bandit algorithms stems from an empirical study based on the Yahoo! Today dataset [12]. This study only considered non-contextual algorithms for two main reasons.

1. They are the most well studied theoretically. Yet such a comparison was not possible before the release of the aforementioned dataset and is thus very interesting to all the bandit community.

2. Contextual bandit algorithms are more tricky to evaluate as explained all along this document.

The first lesson that can be drawn from this study is the following. In machine learning, it is common to find that algorithms, for which no proofs of convergence, consistency, *etc.* perform better that theoretically justified algorithms. For a long time Thompson sampling was not theoretically justified (this is not true anymore [55]) and it is very likely that it led people to think that it would perform better than theoretically proven algorithms such as UCB for instance (see Chapelle and Li [56] for an example). Our study sets things right by exhibiting that UCB, although it is not aware of the nature of the reward functions (Bernoulli) does perform better than Thompson sampling. Intuitively this makes a lot of sense. Indeed Thompson uses a very elegant method of exploration based on randomness. This method also comes with native robustness to technical difficulties such are delayed or lost rewards. Nevertheless, compared to a deterministic approach, all this has a cost that we quantified on real data. This study also yielded a very interesting surprise. The simplest algorithm that one can think of to deal with the exploration/exploitation trade-off, *Optimistic Greedy*, outperforms all its competitors.

As detailed in the introduction of this document, the design of new approaches to deal with dynamic recommendation in particular and online learning problems that evolve over time in general was our first purpose. Yet as interesting as the contributions we made in that area may be, they are clearly not the most significant work of this thesis. The main contributions that this document accounts for are serious improvements of the techniques of **evaluation** of contextual bandit algorithms based on past data. In particular, we exhibited how the bootstrap and other techniques we derived from it are helpful with regard to this purpose. Furthermore, not only did we improve state-of-the-art evaluation methods, we also significantly sharpen their theoretical analysis and thus the understanding of their behavior.

Starting this thesis on evaluation with the description of our participation to a recommendation challenge might have seemed surprising at first. Nonetheless the reader must have noticed that it was both a good introduction of how difficult evaluation is, and an Epiphany to us for we realized that this is what needed to be studied and improved. At this point it is also rather clear that the part of this work on contextual bandit evaluation was an iterative research process. Indeed, each new step we introduced was motivated by the shortcomings exhibited in the previous one. A funny detail is that even the original motivation of this work on the evaluation of contextual bandit algorithms stemmed from conclusions drawn after the challenge we won (chapter 2). More specifically, showing how biased an "home brewed", indirect evaluation method - namely the one of the challenge based on display choosing - could be, led us to work on the subject and propose two distinct contributions. The first one is a minor contribution discussed in appendix B. We proposed a method based on classification that, quite interestingly, allows to use the plethora of work in classification for profiling purposes in dynamic recommendation. Yet the method is offline by nature so we quickly discarded it as far as bandit evaluation is concerned. The second contribution the conclusions on the challenge led to is actually the second major set of contributions we already mentioned. This first major set of contributions with regard to offline bandit evaluation is an extensive study of recent, direct evaluation method that has become quite popular. We simply refer to this method, introduced by Langford *et al.* [10] and analyzed in our setting by Li *et al.* [11], as *replay.* This method has acquired a relative popularity in the recent years, probably due to its advertised unbiasedness. Furthermore a fair amount of work has been done to extend this work so as to relax some assumptions on the logged data. Nonetheless, even the simplest version of replay lacked both a proper theoretical analysis and a **reproducible** empirical study. Indeed the state-of-the-art analysis only provides a rather coarse concentration bound and the existing experiments are based on private data. Consequently they could only be trusted blindly which is not very desirable. Our study can therefore be divided into two parts, theoretical and empirical, that both produced very significant results. More interestingly, these two approaches to study replay enabled us to acquire very distinct intuitions on its behavior that are all important in practice.

The first part is a theoretical analysis in a simplified setting that only considers static policy. Indeed, if it was proved that replay is asymptotically unbiased for any algorithm (as long as the data is sampled uniformly), not much more can be done in this general case. Li *et al.* [11] have however defined a simpler setting in which finite time results can be derived. Yet the literature only contained a naive concentration bound [11] in this setting. We significantly enhanced the state of the art by:

- analyzing replay's *finite time* bias (and proved in the meantime that even in the simplest setting, it is not unbiased contrarily to what was advertised),
- providing a new method, *replay** that is truly unbiased,
- refining replay's concentration bound by using Chernoff inequality in a less naive fashion,
- providing a concentration bound for *replay*.*

That should have been it but surprisingly, *replay**'s concentration bound was tighter than *replay*'s. Even more surprisingly, although it was very intuitive that replay's bias was removed at the cost of some additional variance, we exhibited that replay* met the optimal bound that could be derived with the tools we were using. This is why after concluding that such concentration inequalities may not be the best tool to analyze these methods with precision, we proposed a variance analysis. We have not been able to derive a close form formula for replay's variance (although it seems that a very tight upper bound would be a much more achievable goal). Yet, a subtle approximation allowed us to demonstrate that replay* (for which the variance analysis is straightforward due to its unbiasedness), has a larger variance than replay. Besides confirming our intuition, this result is relatively important. Indeed, replay has become quite popular these days. Furthermore, it is generally used in a context of risk minimization. Consequently, having a precise sense of the form of its variance is crucial and can help computing much more accurate confidence intervals. Note that it is intuitive that replay concentrates faster that replay*, contrarily to what the theory seemed to indicate. Yet one may argue that the variance and a high probability bound are not similar ways to quantify the uncertainty of replay which could explain the different results. Therefore one may consider our approach, that bypasses the problem, to be unfair. This is not the case for large datasets. Indeed, for a relatively large dataset (100 times K), replay is going to be very symmetric as well as thin tailed. Consequently the two ways of quantifying uncertainty are bound to be correlated, which justifies the significance of our results. A last theoretical contribution with respect to replay is the study of non uniform logging policies. This problem was extensively studied in the literature for stationary policies [10, 97, 98] but nothing was said about learning algorithm, leaving a blur. Indeed if the bias and variance results can definitely apply, to some extent, to learning algorithms, it is not the case of all the results. In particular, we proved that replay is not asymptotically unbiased for learning algorithms when the logging policy is not uniform, regardless of the corrections that are made.

Small empirical studies had already been presented by the authors of the replay method [7, 11]. Yet the set of experiments we designed and conducted in this thesis work is remarkable by its unprecedented scale, the way it was meant to push the method to its limits and its repeatability.

1. **Repeatability:** This is clearly a very important quality for some piece of research that, due to the lack of public datasets, the previous work lacked [10, 11]. With the release of the Yahoo! R6B dataset [12], we were the first to provide repeatable evaluations of classical bandit algorithms on a real and meaningful application (news recommendation).

2. **Scale:** We also used this dataset to organize the first challenge of his kind, based on *replay* and real data acquired the proper way. We recorded more than 5,000 submissions from around 50 participants, making this challenge an unprecedented experiment with respect to both the number of tested algorithms and the number of minds put to the task.

3. **Fairness:** Along with repeatability, fairness is another very desirable property within a piece of research. In the past, replay had only been tested by its authors to check that it was working as expected. While such an experience is always biased by nature by the researcher's assumptions and preconceived ideas, a challenge is relatively free from all this. The goal of the dozens of participants is not to verify whether the evaluation method is working or not. Their only purpose is to score as high as possible, using every possible flaw they can discover. By allowing this we let replay being pushed to its limits, even those that were unexpected.

Indeed, some limitations when evaluating real world learning algorithms were clearly expected given the nature of the theoretical analysis based on static policies only. We detailed them in chapter 4. In particular we expected two factors to tamper with the quality of the evaluations.

1. Nothing prevents non supervised learning from being performed in the *choose* function which is called even for rejected records (the reward and the action are unknown in that case). See sec. 4.2.1 and sec. 4.4.2 for more details on the subject.

2. Convergence can not be assured for non static policies (sec 3.5.3).

Yet it turned out that these are not the limitations we reached in this challenge. Indeed, although convergence cannot be assured for non static policies, it could be done for consistent learning algorithms. As a matter of fact, we even noticed a great stability in the evaluations of such algorithms so we can conclude that this issue is really not a problem with that kind of data. Moreover it seemed that no one in the challenge has been able to successfully apply non supervised techniques on the contextual information to improve their performance.

As already mentioned, limitations of replay lie elsewhere. The amount of overfitting which is inherent to such a challenge and that we necessarily noticed was relatively small compared to the huge number of submissions of some participants (up to a thousand). What was more troubling is that performance gains due to overfitting set aside, no one outperformed a simple non-contextual algorithm although evidence justifying that it should have been possible have been given. We came to the conclusion that it was almost surely due to a phenomenon that was never studied before (at least in this context) and that we referred to as *time acceleration*. Indeed, in a nutshell replay is unbiased in some sense because it does not alter the process generating the data. In a static world, given the usual size of the datasets we use, unbiasedness is total, even for learning algorithms. Yet we gave plethora of evidence of the dynamicity of the environment in the context of news recommendation (new items, CTR decay, user taste evolution *etc.*). Because of replay's rejection mechanism, the environment is seen roughly as it is in the real world except that all the changes happen much faster, hence the designation: *time acceleration*. A visual example of the bias introduced by this phenomenon can be found in section 5.2. More intuitively, this bias changes entirely the problem that needs to be solved. Indeed, the essence of an online learning algorithm is the way it tackles the exploration *versus* exploitation trade-off. With replay, exploration is a few tens to a hundred (K) times more expensive!

Consequently, replay which was advertised as unbiased was is fact exhibited in this work to be so biased that it was almost useless in practice. Obviously algorithms that tend to explore as much as each others can be compared rather fairly. This is why we were able to compare non contextual algorithms in chapter 4. Nevertheless this is hardly enough. We want to be able to compare fairly algorithms that use underlying models of various complexities and thus require various levels of exploration. Replay simply can not do that if the environment is not static. From this statement stemmed the main contributions (the third set) of this thesis work and most likely the most significant. To tackle this issue, we used the same resampling tools that are classically used in the statistical bootstrap. We thoroughly analyzed the gain of our new approach in terms of quantity of information gathered from a dataset about the CTR of a policy. We also exhibited excellent empirical results in the practical cases that were of interest to us but that did not meet the theoretical assumptions. In addition to improving evaluation accuracy, we provided another theoretical analysis based on the bootstrap proving that we are able to compute features of the distribution of the CTR of a policy with much higher accuracy than any analytic method. In particular we are able to derive very accurate confidence intervals. This is a highly desirable skill, much more than a simple CTR estimation, considering that we are trying to minimize the risk of putting online a new policy. Furthermore we proved that thanks to the bootstrap, the estimation of these confidence intervals converge much faster than the estimation of the mean of the CTR of a policy. Our approach does yield a better accuracy than replay. Nonetheless because confidence intervals are way more informative and because we estimate them with higher accuracy than the mean, it increases by that much our ability to minimize the risk of putting online a new policy.

All the theory was derived under the assumption of a static policy. We justified that this setting is already highly relevant in practice (see Bottou *et al.* [98], section 5.8.1 or figure 5.10). Indeed the analysis brought us to a very unexpected realization: using replay methodologies no classical recommendation is a viable option. Furthermore our word on the bootstrap significantly improve the state of the art with that regard by allowing to optimize randomization and to estimate confidence intervals over our estimations more accurately. We referred to the method, which is described on figure 5.10 as *iterative improvements*. Nonetheless we were originally interested in the evaluation of learning algorithms. Our method based on bootstrapping - BRED - does not apply directly to that case. As mentioned previously, we obtained very promising empirical results in which time acceleration no longer seemed to be an issue. To achieve that we used a simple trick referred to as *smoothing* or *Jittering* that basically prevents a learning algorithm from overfitting on duplicated records by adding noise. Surprisingly, although we introduced this technique for the sole purpose of evaluating learning algorithms, it also allows to gain accuracy when evaluating static, deterministic policies. See figure 5.11 for a justification and figure 5.12 for examples. We did not study this phenomenon a lot but it is very interesting for it is a very simple way to improve the evaluation of any static policy. It may actually be useful in a different context. In this work we mostly considered the case of a uniform logging policy. In practice however, when evaluating static policy with iterative improvements, this is not the case. One could also have at hand data which was not acquired uniformly. One very well known problem in that case is the high variance introduced by some actions that are not performed a lot [97, 100]. It is usually dealt with by adding thresholds and therefore some bias. Jittering is another way to reduce that variance by using very simply and efficiently the fact that neighboring areas of the context space respond similarly to actions. Obviously if no such regularity can be assumed, smoothing is a terrible idea. Yet in practice, it usually works very well. Finally, since smoothing is part of the bootstrapping procedure (we sample from a KDE), there is no additional work to do in order to derive confidence intervals for estimators computed on smoothed data.

The last contribution of this thesis work is much more practical and is related to smoothing. The major contributions we just summarized laid strong foundations to evaluate policies via extensive resampling and bootstrapping replay estimators. For static policies, which already have their range of applications, there is nothing more to be done. For learning algorithms however one detail was last to be discussed: the selection of the bandwidth of the kernel used to smooth the data, or in other words the amount of Jittering to introduce. We first provided a very simple method based on cross validation. Then we designed a much more interesting approach to actually make the problem of bandwidth selection secondary. Indeed, with this new method, called *entangled validation* we ensure that it is impossible for the evaluation method to be positively biased. In other words, regardless of how bad the bandwidth is tuned, it is impossible to overestimate the CTR of a policy (in expectation). This property allows a very fine control of the risk of putting a new policy online. Another extremely interesting property is that compared to the original method (BRED), this bias control is only performed at a cost of computational resources. We lose nothing in terms of precision (no variance lift) which is quite remarkable considering how successfully we deal with the bias.

To conclude on this review of the various contributions of this work, we have enumerated a certain number of them while trying to precise which ones, according to us where the most significant. The analysis of replay, the identification of time acceleration, the theoretical framework based on bootstrapping that we introduced to deal with this issue and the notion of entangled validation clearly stand out. Yet in our opinion the main contribution of this thesis work is not any particular analysis, idea or empirical result. Rather we do believe that the main, major contribution of this thesis is in fact its totality. Indeed it draws *a much better understanding* of data-driven evaluation of learning algorithms by analyzing it clearly and thoroughly, by identifying its difficulties and providing leads and solutions to overcome them.

2. Horizontal themes

Before moving on to the numerous perspectives that open after this work, let us discuss a few *horizontal* themes that were dealt with in this work. The first theme we mention is theoretical analysis and more specifically the intuition they should carry, the simplifications of a problem and the assumptions that can or must be made, what they should prove to be significant, how they should prove it and a small focus on error bounds. A second theme which is pervasive in this work is research evaluation. Finally we say a word on a very famous trade-off: the one between variance and bias.

Although the motivation of this work is highly practical (risk and thus cost minimization in the context of dynamic recommendation), a significant fraction of it is very theoretical (bounds, variance analysis, bootstrap analysis *etc.*). In a lot of scientific fields, people like using mathematical tools to prove that an approach is working as expected, whatever "working" may mean. This work is no exception to that rule. Among many other things, we proved that replay is asymptotically unbiased, that it converges *etc.* We also studied at length the theoretical properties of our bootstrapping based approach: BRED. Nonetheless, in addition to the inherent technical challenge of the analysis, such a proof of correct behavior is rarely straightforward. Indeed, except maybe in pure mathematics, people are generally not dealing with mathematical problems directly. Therefore, to be able to use math anyway, they model their problem into a math problem that seems analyzable. Moreover, such a problem should be representative enough of the real problem so that its analysis, if it does not tell us exactly what happens in practice, at least give us some useful *intuition*. Indeed, a theoretical result rarely takes into account the whole complexity of a problem so intuition is usually all we get. Nonetheless it is in fact all we need for even with an inaccurate result, as long as we have intuition on the effect of the various important parameters of the problem on the behavior of our approach, we have all we need. However finding such mathematical problem that would carry the right intuition may not be easy and even very challenging. We have had this very problem in this work which led us to completely ignore time acceleration in the theoretical analysis although it turned out to be the most prominent source of bias in practice.

To be more specific, the ultimate goal in our context of news recommendation is to maximize user satisfaction. This is not a math problem but we chose, as it is very common, to model it as a contextual bandit problem. This introduces many implicit assumptions on our problem such as independence between the users, the arms and the times steps. It also implicates a static environment and very importantly, that our goal is equivalent to maximizing a sum of rewards. We did not discuss this assumptions that much except in the first chapter but it is crucial. Yet even when going online is possible, it is extremely hard to measure directly user satisfaction and the CTR is generally used to measure it. With ad placement and the cost per click for instance, this measure is even exactly what we are interested in. Staticity is a rather universal assumption without which nothing can really be done. Indeed, taking into account dynamic aspects would lead to additional assumptions to characterize it, very little generality and we would thus gain nothing except if we are interested in one application in particular for which we have a precise characterization. As a matter of fact, we do believe that in the general case, considering a static environment is not only simpler but also much more interesting. Indeed, given an application, we can generally tell roughly how it evolves. Is it evolving fast, slowly, smoothly, periodically *etc*? Then, although it is not perfect, it is always more or less possible depending on the case to use the results on static environments to acquire intuition on our more complicated case. On the contrary, a more specific result on one particular type of dynamic environment is much harder to compare to an environment with different properties. Unfortunately, the contextual bandit setting was still too general to let us prove anything more than an asymptotic unbiasedness. This is why in the literature as well as in this work, static policies were considered. Similarly to what was said for the environment, considering one particular learning algorithm would be possible but it would make everything much more complicated and it would be hard to get intuition on other kinds of algorithms using this result. The only difference is that the evaluation of static policies is already interesting in itself if we want to do *iterative improvements* as described in section 5.8.4. Nonetheless this really seems like a very strong assumption in general. Also one may wonder if results based on such an assumption would be of any use to understand what is happening with learning algorithms. As a matter of fact possible issues caused by the stationary policy assumption (such as a lack of convergence) were what we were after in the first place when organizing the challenge described in chapter 4. We now all know the story, in the end evaluating learning algorithms ended up being extremely stable. Indeed, one has to remember that a decent learning algorithms is consistently converging toward a static policy. The environment is also assumed to be stationary in the analysis which is not true in practice but the fact that it can be considered as static by parts seemed to shield us from bad surprises. Yet it is this assumption, although partially met, that turned out to be the most problematic: it is the cause of a huge bias and the main contribution of this thesis is to reduce it! This is a perfect example exhibiting that modeling correctly a problem, that is so that it reflects the right intuitions, is as crucial as is it tricky.

Once a satisfactory mathematical model has been defined, one has then to decide what to prove and how to prove it. For various reason, the "what" may not be straightforward. In our situation, we decided to analyze the concentration/variance of the estimator which is something very typical to do. Furthermore this was what was done in the literature so it was somehow a prerequisite in order to compare our results with previous ones. Yet, in our context of risk minimization of putting a new policy online, other possibilities were available. We could for instance have tried to quantify the risk of being fooled by an outlier when evaluating a policy. People in fact have a similar problem when evaluating bandit algorithms (and not evaluation methods for bandit algorithms as here). Instead of minimizing regret (or maximizing the cumulative reward), some argue that it is preferable to minimize risk. It leads to slightly different algorithms [145, 146] that do not necessarily minimize the classical regret. Sometimes the problem is even worse. For instance, with classical recommendation it is not really possible to have access to the quantities we may be interested in like user satisfaction or even the CTR. Therefore we have to evaluate a different quantity which might reflect some aspects of the quality of the system but not the exact quantity we are after. Is this case choosing the quantity a system should optimize (and thus the one we want to analyze) is even more subjective than in our example. Note that we discussed this in section 1.5 as did Herlocker *et al.* in a founding paper of the area [8]. What is to be kept in mind for that work is that we more or less skipped this issue by considering what was done in the literature in order to compare our results. It would however have been perfectly possible to consider different things to analyze such as risk.

Finally, even when the theoretical framework and the objective have been defined, we can still somehow go astray. One example that occurred in this work is that even though a quantity was identified as interesting, its upper bound was just too loose to provide any intuition with regard to what we were interested in. More specifically, we used Chernoff inequality to prove an upper bound on the error made by replay. We then did the same for replay*, discovering with surprise that for any dataset size, replay*'s bound is lower than replay's, going completely against our intuition. Obviously we are only talking about bounds so this does mean much about the real performance of both methods. Also, the bound is in $O\left(\sqrt{K/T}\right)$ as expected which is more than enough to understand roughly how it converges. Yet it is clear that the analysis, although perfectly correct, completely fails its main purpose of providing intuition as to when to use which or which method. Worse than that but very interestingly, we showed that the bound on replay* matched the best that could be hopped to be derived with Chernoff inequalities. Therefore, although we do not know whether or not the bound on replay can be improved or not, it actually does not matter for we simply can not beat replay*. It illustrates our point perfectly, that is even when we know what to analyze, doing it so that it stresses out the intuition we are after may be tricky. More particularly, it also teaches us that bounds, by definition are not always accurate. In the bandit and reinforcement learning (RL) literature, it is quite common to encounter work in which the entire approach is driven by this need to optimize the concentration bound itself, and not the quantity it bounds. These works on bandits by Auer and Ortner [147] and by Audibert and Bubeck [148] are good examples in the field of bandit problems. Without at all questioning the quality of all this work, one may still wonder if this is always the best strategy. Could not we be targeting a sub-optimal solution (whether we manage to reach it or not) of our problem? To anticipate a little bit on the perspectives, it would be very interesting to study this issue and to possibly find examples of it happening. On the contrary, such a study might also lead to the conclusion that it does not generally happen. Yet in our opinion our example with replay and replay* is basic enough to foresee other similar cases. Note that we have had another instance of this problem of somewhat useless or even counter intuitive result while studying the bootstrap. Indeed, our first analysis, embodied by theorems 12 and 13 was based on tools developed by Kleiner *et al.* [113] in order to show that an idea somewhat similar in spirit kept the same order of convergence as the original bootstrap. In our case, applying these tools and showing that our method met the requirements to be bootstrapped was interesting in itself. Yet because the first work was only about orders of magnitude, it was not accurate enough to provide any intuition whatsoever on a possible gain of performance, which is what we were interested in. This is why we studied our method under many other aspects in what followed so as to show that we were indeed achieving what we wanted in terms of precision. This much deeper analysis also allowed to provide a great deal of details on the properties of the bootstrap estimator that we were constructing and turns out to be a significant contribution as well. Anyway, what is interesting to keep in mind here is that although our first analysis was based on classical tools from the literature, fitted our problem perfectly and provided interesting insight on what the bootstrap could bring to this problem, it did not allow to acquire any intuition on a performance lift, which is what we were after. Only a very different analysis, using quite different tools was able to achieve that and in the meantime precise what the first analysis had only let foresee on the interest of the bootstrap in our situation.

So far, we have mostly talked about one main horizontal theme: theoretical analysis. In particular in this thesis we have encountered very interesting examples of how it could be useless or even counter intuitive if carelessly performed or interpreted. One can view a theoretical analysis as a way (among others) to assess the quality of some research (a model, an algorithm, a user interface *etc.*). There are obviously plenty of other ways to do so. Some ways are simply properties that a piece of research should possess while others are methods to compare it with others. In this work we have emphasized that although some experiments already existed about replay, an interesting contribution that we made was to make some new ones that are both repeatable (synthetic model, public data and all the details of the experiments are provided) and as fair as possible (participation of many people whose purpose is to exploit every single flaw of a method and not to prove that it works, automatic algorithm tuning while comparing performance *etc.*). As a matter of fact, it is not a surprise to realize that quality assessment is such a pervasive topic in this work given that the main contribution of this thesis is a method to evaluate one important quality of a particular research, namely the empirical performance of a contextual bandit algorithm on a real application. Nevertheless it is quite interesting to notice that the contributions we made with regard to quality assessment methods can be separated in two groups. Indeed, not only did we designed new methods to overcome issues that were not tackled in the past such as time acceleration, we also significantly enriched the ways by which the quality of this previous research was assessed in the literature. This significant enrichment ranges from deeper theoretical analysis, to better experiments (more fair, repeatable) and from better use of qualifying terminology (unbiasedness) to the detection of a tremendous flaw (time acceleration).

Finally, although quality assessment of research was certainly the main cross domain topic of this work, another detail is worth mentioning. Given the nature of the algorithms we wanted to evaluate and design, the exploration *versus* exploitation dilemma was at the core of this work. It is even the fact that different algorithms deals with it differently that time acceleration is such a big issue because it causes exploration to be much more expensive. It is extremely interesting to notice how another trade-off keeps popping up everywhere: the variance/bias trade-off. In this work we for instance saw it when dealing with replay's bias or when introducing Jittering to evaluate both learning and static algorithms. Nonetheless, we would like to stress a rather amusing detail that occurred in this work. Rather counter intuitively, introducing entangled validation in BRED enabled us to very successfully deal with its unknown and thus uncontrollable bias. Remarkably, forbidding this bias to be positive and thus acquiring control while trying to reduce it was done without any variance lift. Rather the cost is paid in computational resources.

3. Perspectives

To conclude this work, we have to talk about perspectives for they are numerous. Most of them have already been laid out all along this document and in particular at the end of each chapter. This concluding paragraph is meant to organize them in a clearer fashion and to highlight the interest of pursuing them.

Because of the nature of this work, the perspectives can be divided into two categories. We mainly worked on the evaluation of contextual bandit algorithms. Therefore two paths open naturally in front of us:

1. extending this work for we clearly have not addressed all the issues it entails yet,

2. using this work to improve the state of the art in terms of bandit algorithms and in particular design more data-oriented algorithms.

Thinking about extensions is only natural while concluding such a work. Indeed although we clearly achieved something here by - among many other things - understanding better the offline evaluation of online learning systems, there remain many unanswered questions, uncertainties and untackled issues. All of them are exciting perspectives for future work. A great part of the work we accounted for in this thesis is theoretical. Yet all this work was driven by practical issues and plethora of very promising empirical results have been provided. Consequently, this research opens both theoretical and empirical perspectives that are equally exciting and that we will mention in turn.

We have already been through a lot of theoretical perspectives in this document, in particular while concluding chapter 3 and 5. A first perspective would only be technical. The pure variance analysis of replay and RED∞ (not RED$^*\infty$) are not in close form. In may or may not be possible to change that but an interesting lead would be to try deriving very tight upper bounds that would simplify the calculations without losing too much of the intuition contained within the formulas. Such result would probably allow us to compare analytically the biased methods (replay and RED∞) with their unbiased counterparts (replay* and RED$^*\infty$), similarly to what was done visually in section 3.5.6. Remark that this would be very interesting for stationary policies when K is not negligible compared to T. A second perspective is the actual theoretical study of the evaluation of learning algorithms. We established that convergence results were impossible in the general case. Yet we also said at length that such results would be possible for consistent algorithms, which any reasonable learning algorithm is. Such a result would not revolutionize what we did here basing ourselves on intuition acquired from results about static policies. Nevertheless they would allow us to sharpen this intuition and identify with greater precision what are the limiting factors of a convergence result. It is straightforward to conjecture the dependence in an exploration term of some sort. Indeed, the more an algorithms explores, the more it has the possibility to use the full expense of the knowledge contained within the data and converge faster. However it would be very interesting to see this with our own eyes. We will come back on this point in what follows but it is very likely that such an expression of convergence that depends on the time step would be helpful to overcome the lack of knowledge within the data identified at the end on chapter A while experimenting with entangled validation. Furthermore, studying what differ in such convergence expressions for a few distinct but classical learning algorithms, might be source of insight as well. Apart from that, the analysis of replay and replay* do seem pretty thorough. We can make a few additional remarks though. It is obvious that for learning algorithms, replay is going to be biased. Also the quantification of that bias could be done easily. Yet although such an analysis might be interesting in itself, we do believe that contrarily to what we argued for convergence results, it would not bring much to our understanding of replay methodologies in general. Note as well that the convergence analysis in the case of BRED with learning algorithms would surely require assumptions that would be way too specific to be of any practical use. Indeed it would depend on many parameters (Jittering, overfitting, regularization, exploration *etc.*).

Nevertheless, BRED and the bootstrap framework that we developed offers many other theoretical challenges that seem much more promising both in terms of feasibility and interest. As a first example, we exhibited empirically that Jittering helps increase the precision of evaluation of even a static, deterministic policy. We provided intuition justifying this but it would be interesting to design a small but rather realistic theoretical framework to analyze this phenomenon. It may allow us to then measure on real data if introducing Jittering is worth it or not. Indeed let us remind that RED∞, the static policy estimator is an infinite sum that factorizes to T terms, T being the size of the dataset. When Jittering is added, the factorization is not true any longer and we need to compute that infinite sum (or at least until convergence). Besides such an analysis may also help decide which Kernel and which bandwidth to use when evaluating deterministic policies. Indeed, in this context we did not study the problem. Yet it is much more similar to classical bandwidth selection than with learning algorithm for we simply have to deal with a variance *versus* bias dilemma. As such the classical methods from the literature may turn out to be helpful. It would be worth to investigate the matter a bit further. On the contrary, note that entangled validation would be useless here for no overfitting is possible.

Entangled validation is in fact where lies the greatest challenges in both theory and practice for the evaluation of learning algorithms. We saw that there remain a few issues. In particular although overfitting seems to be under control, there is a lack of knowledge in the data to be able to really simulate the real learning process. In the simulation, the algorithm is underestimated simply because the knowledge is somehow less concentrated due to duplicated records. We proved that adding duplicates does allow to drag more knowledge due to Jittering (same record seen as a context and its neighborhood instead of just a point) and exploration (same record played with different actions). Yet as we exhibited it in chapter A, it is not enough in practice. We provided several leads at the end of this chapter. For instance we exhibited a way to use additional expansion to let the algorithm acquire more knowledge, while still keeping the bias under control via entangled validation. This is one way that remains non parametric. Yet, now that we have a way to control any positive bias, nothing prevents us anymore from introducing parametric models. It would in fact be straightforward once a model is built, to evaluate an algorithm against it and entangle real records within the modeled interactions. Doing so would simply be another step following the idea of KDE. Indeed, although it is generally considered as a non parametric technique, it is in fact a way to estimate the distribution of the data with the simplest assumption: normal smoothness. By introducing a model, we are simply performing the next natural step. Yet it is important to note that a model too rich (*i.e.* that contains more knowledge per record about the data than real world interactions) may positively bias an evaluation and that one should be careful about that. This seems unlikely if the model is designed honestly. Yet a very accurate model of the reward expectations that does not contain enough noise may for instance cause that problem. Anyway, this idea opens many theoretical perspectives (model quality, convergence of the evaluation depending on a specific known model, model selection *etc.*) as well as empirical challenges: what kind of model would work for instance? Finally, entangled validation combined with simple non parametric resampling or more complex models could most likely be applied to the evaluation of other online learning systems. One could for instance think about online regression/classification, reinforcement learning or even in multi-agent frameworks. This could also be used under different formalization of *decision making under uncertainty* that are in use in other fields such as management, marketing or even environmental sciences [20, 21].

What we just said is somehow taking one step backward by considering again that models can be of use here as long as they are controlled non parametrically (by entangled validation). This is a very exciting perspective given the tremendous amount of work made on modeling the web and human interactions which are the most well known applications to decision making under uncertainty. We can also come back on another conclusion that may have been made hastily. In chapter 2, we argued that the evaluation method, that we may call *batch replay*, used in the challenge we won was to be completely discarded. Indeed, selecting the user is so much easier than making personalized recommendation that trying to do the latter only deteriorated our performance. *Batch replay* simply changes the nature of the problem too much. What if it were not the case? According to its authors, batch replay's primary purpose was to force the participants of the challenge to consider the users and to avoid non personalized recommendations. In this it clearly failed. Yet this idea of grouping the records by N, N being smaller than K is interesting for it straightforwardly reduces time acceleration. Since the possibility of choosing the user is highly problematic, we could consider only grouping similar users, gathered via a clustering algorithm. We may even not let the evaluated algorithm be aware that it is dealing with several users. Instead we would only provide the "mean user" of the batch. There are obvious issues to deal with. In particular the evaluated algorithm will not be able to choose the actions it desires at each time step (only those within the batch). This will cause bias. Yet several solution tools are at our disposal to deal with that.

- By being aware of the evaluation method while logging the data, we can make more intelligent choices to reduce future bias and/or variance.
- By introducing resampling we can finish reducing time acceleration. The clustering process doing part of the job, this should be much less problematic with regard to overfitting than with the classical BRED.
- By using entangled validation, within or without batches, we can help control the bias introduced by resampling.
- Finally, modeling is still an additional option to help solve any issues due to lack of knowledge.

Anyways, *batch replay 2.0* seems very promising but needs to be experimented on in some future work in order to let us use it correctly.

The last major perspective of future work regarding contextual bandit evaluation is purely experimental. More results on the evaluation of contextual bandit algorithms on real data would have been to be the perfect conclusion to this work. Nevertheless due to time as well as computational resources limitations, we have not been able to produce significant ones. For the obvious reason that to evaluate an evaluation method, the ground truth is needed, most of our experiments were run of synthetic data. These experiments let us identify many practical issues that we then dealt with and in the end, we obtained highly satisfactory results. Nevertheless, we can argue part of what we argued against the state-of-the-art set of experiments on replay. Indeed, we built our synthetic model on which we experimented based on a simplified version of our idea of the world. Thus, although we did it in perfect candor, such a model and all the experiments that were run on it might be biased by preconceived ideas we may have had on the real world. Note however that we believe that the nature of the model had little impact on the success of our evaluation methods but we have not investigated that matter in this work. Doing so might be interesting.

Moreover the main goal of this work was after all to evaluate contextual bandit algorithms on real data. This is why it seems very important to be able to validate this research on real problems. The straightforward way to do so would obviously to do exactly as what Li *et al.* [7,11] have been able to do with LinUCB and replay, that is comparing the evaluations in the real world to the replayed ones. We greatly encourage people who have the possibility to run such experiments to do so and to release the results for they would constitute a real leap forward in the field of dynamic recommendation and any other applications of contextual bandits in the meantime. We obviously can not do this by ourselves. Yet we have already provided a few interesting results using a small trick. In section 5.10.3 we fed a very small fraction of the data to various evaluation methods, and considered replay on the rest of the data to be a realization of the truth. It led to very promising results but it was unfortunately unthinkable to evaluate contextual algorithms on so little data. Thus only the evaluation of non contextual algorithms was exhibited to be improved on real data. Yet we also provided evidence in section 5.7.6 that, as surprising as it may at first sound, non contextual algorithms also suffer from overfitting and underestimation in turn. We also exhibited that their evaluation behaves exactly the same as for contextual algorithms, the only difference being the scale. Consequently, if we managed to show that the evaluation of non contextual algorithms takes advantage of BRED with very little data, contextual bandit algorithms should take advantage of BRED as well with more data. The only issue is that since the necessary amount of data is roughly the amount we have, we have no way to produce some notion of truth to check out that it does work. We are however very confident that is would work given all the positive evidence provided in this document.

We did try a first step in that direction that does not need the access to a real application. With entangled validation, we know that we can proscribe any positive bias (at least in expectation but since the results are pretty stable the distinction is not very important). Consequently, if we were to find that LinUCB outperforms UCB on the data of the challenge, it would confirm the practical significance of BRED. Nevertheless this turned out to require a lot of computational resources due to the cost of entangled validation (more runs to get convergence) and to LinUCB's complexity. As a consequence, we have not yet been able to get significant results. At first sight (not enough runs were performed to be certain) there does not seem to be a great lift in LinUCB's performance but we used a raw version of T-BRED. Therefore the gain due to the removal of time acceleration is probably compensated by overfitting and the lack of knowledge during exploration. It would be very interesting to see what happens when all the aforementioned solutions are implemented. We also did not want to evaluate any other algorithm than the classical ones so as to remain fair and avoid evaluating *our algorithms with our methods.* However it is clear that given the very dynamic nature of the problem and the CTR decay in particular but also the binary nature of the features and their number, LinUCB is far from being the optimal solution. Yet we remain convinced that very simple contextual algorithms could have outperformed UCB and that our methods are the means to highlight that. Indeed numerous pieces of evidence supporting that claim were given all along this document and summarized in this concluding chapter.

In fact apart during the challenge in which people tried many things, we only tried to evaluate LinUCB since it is almost the only contextual bandit algorithm from the literature. This very statement leads us to the second category of perspectives that open after this work. We talked at length about how we could improve BRED and entangled validation. Many sound solutions have already been described and are just waiting to be tested. Here is therefore the main perspective: using all the methods proposed in this thesis work to think, design, evaluate and analyze new bandit algorithms that work well not only in theory or on toy models but on real world applications. In particular it would be very interesting to actually take into consideration the characteristics of the problem such as the CTR decay (or other possible trends noticed in chapter 4), the limited lifetime, user taste evolution *etc.*. Girgin *et al.* [83] published an interesting work that considers limited lifetimes and budgets of clicks. Such a work is clearly correlated with all this. Furthermore, given the complexity and heterogeneity of the data, many challenges are ahead of us. It would be very interesting, instead of simply building a reward model online, to try to improve online as well the input provided to this model. In other words we could set our interest to new very exciting tasks such as the online selection of features, online clustering of items (according to CTR trends, users tastes *etc.*), online clustering of users or even computing online a PCA or another compression of the context space. Performing all this seemed impossible because of time acceleration for things had to be learned and exploited fast. Yet with BRED, many things can be envisioned. All these ideas to design practical bandit algorithms also open theoretical perspectives. Indeed although we want to get nice performance in real applications, we always are after intuition on what is happening and how we could do better. To acquire such intuition we would need to model relevant characteristics of a real application (CTR decay, user clusters *etc.*) in various richer contextual bandit problems and to perform a regret analysis.

Speaking of theoretical analysis, it is not entirely true that we have not been able to beat UCB. We did but not with a contextual algorithm. We beat it with optimistic greedy, the simplest optimistic bandit algorithm one could find. It would be very interesting to get a regret bound for that algorithm or see if it is even consistent. We believe it is possible but looking into it, even in the most basic and simple setting (Bernoulli rewards), would definitely be interesting to understand further bandit problems. Another nice algorithm that was designed here is LAP. We argued that it was an efficient way to approximate LinUCB and make it much faster. This was proved empirically but it would be very interesting to quantify how much we lose by making this approximation. Given the nature of the algorithm, we suspect that an analysis very similar to what was done for Thompson sampling would be possible [55].

Finally, let us mention one last and very exciting perspective that was not really planned when we started working on all this. So far we have mostly talked about contextual bandit algorithms whose design and evaluation were our primary purpose. Yet in chapter 5, we also argued that our work also applied in a slightly different context. Indeed by improving the precision of the evaluation of a *static* policy and in particular reducing the dependence in K the number of items, we broaden the range of usability of replay methodologies. Recommender system with thousands of items are now likely to be evaluated with a direct metric of performance: the CTR or any other online reward function that may seem appropriate. If we consider iterative improvements as detailed in section 5.8.1, one can even hope to perform evaluations with millions of items. Although this accuracy gain is significant, this is not the main contribution of our work. We argued that confidence interval are much more useful than raw predictions and our method offers an even more significant improvement with that regard (convergence rate improved from $1/\sqrt{T}$ to $1/T$). Therefore this range extension we are talking about for replay methodologies on recommendation is in fact much more important than what the raw precision gain lets us expect. By considering confidence interval directly, we in fact go much further. One very exciting perspective is thus to use such methods in practice and to observe the extent of what they bring to classical recommendation. Nevertheless if using our methods in conjunction with what is already done (existing systems, existing logging processes, existing evaluation methods...) is interesting, this is not much compared to the perspective of using this on its own. This method really meets its full potential when it is used as a complete method to design a recommender system (design, evaluation, improvement *etc.*). Indeed we gave tools to tackle the issues that appear at every step of the process. The diagram on figure 5.10 is much more accurate but in a few words, from an existing system or from scratch we provide ways to:

- define the best way to explore (which is proved in chapter 5 to be nothing more than convex optimization),
- improve the system accordingly,
- put it online while managing the risk of doing so,

- reduce that risk (Jittering and expansion for precision, bootstrapping for confidence intervals...).

Obviously putting this in practice would require a full system. It might be interesting to try that on a synthetic model, just to identify general issues but to produce any meaningful result, this has to be done on a real application. With this work, people who have the opportunity to enter such an endeavor now have the sufficient knowledge to do so. It is up to them but the perspective is very interesting. Of course a lot of work is left to be done. All this can be applied directly but does require human intervention in the process to decide of the amount of randomization, of when to change the policy, how to do it *etc.*. For now all we have are tools that are to be used as guidance at every iterative improvement of the system. Yet this whole idea of trying to find the best recommender system, change it according to the environment *etc.* is in fact one big learning problem in which one simply has to balance optimally between exploitation (current policy) and exploration (targeted randomization). A very nice line of future work would be the design of an all in one algorithm that would deal with all the aforementioned steps on its own (see diagram 5.10).

We can finally make two remarks about this iterative improvement strategy. The first one is that if we have an algorithm that does everything automatically, why not consider that micro improvements should be performed at each time step. In fact this would reduce the problem to the one we studied and everything would be dealt with under the formalism of contextual bandits. There are obviously various problems with that. First we would have to come up with a way to update the system very efficiently. It is generally very challenging especially with large item and user sets. At best we would have to make approximations and even then, we would certainly reduce the quality of the learning process. Another issue is risk control. This would require a leap of faith in our algorithm for we will allow blind updates on a tremendous system that potentially generates millions of dollars. It is not the same as with dynamic recommendation (news), in which we know nothing about the incoming content so we can only win by trying to learn as fast as possible. Nevertheless, a tool that would take the logged data of the day or the week, and generate the next policy based on the observations, with the included exploration (exploration may however be allowed to be updated online) in prediction of future updates and a risk quantification would be highly desirable. We could even imagine to be able to modulate the magnitude of the update in order to control the risk better. The second remark is very simple. One may argue that all we are doing is local optimization. If this iterative improvements are done carelessly, it is entirely true for we only allow to update the current policy to a neighboring policy. This is necessary to have a decent precision on the evaluation and thus a controlled risk. Yet nothing prevents us from looking much further, in very risky areas of the policy space. Then if an interesting point is found, instead of directly updating the policy which would be too risky, we can design the next policy so that it explores this area specifically (this may take several iterations). Then we can make the current policy shift safely toward that area while keeping the risk under control if it is confirmed to be promising. As a consequence, "iterative improvements" are not necessarily greedy and may be done in thought of future, bigger improvements. This is just another reason why working on such a procedure seems so interesting.

To conclude, in case it was not emphasized enough, the most exciting perspective that opens after this work is not a possible extension, improvement or variant. Its most exciting perspective both in the academic world and the industry is simply to use that work to evaluate bandit algorithms accurately on problems that matter. Moreover, thanks to the release of at least on public dataset by Yahoo! [12], this is already feasible. Hopefully, this work has highlighted the need for such datasets and others will be released in a near future by the major companies of the web. What is to be kept in mind is that the cost is not that high for the evaluation of learning algorithms. Indeed, it does require data logged uniformly whose cost may seem prohibitive to many. Yet even in a highly dynamic environment considering that what matters is not what is learned but how it is learned, the evaluation of such algorithms only requires one *dense*[3] sequence of interactions per application. Compared to the continuous cost in terms of exploration entailed by the use of iterative improvements for instance simply to be aware of what changes in the environment, this is nothing.

Anyway our last word is that what is the most exciting about this work is simply all the future work on bandit that it will make easier but more importantly, more significant and relevant with regard to real life applications. All that remains to be done now is to evaluate recommender systems or any other system that one can model as a bandit algorithm with a much greater accuracy than what could be envisioned a few years ago.

[3]Because of *time acceleration*, a *dense* sequence acquired on a short period of time and containing uniform interactions with all the traffic is far more preferable to sparse interactions acquired on a longer period of time on a fraction of the traffic.

Appendix A

Bias control for online learning evaluation on expanded data using entangled validation

Abstract of the appendix *This first appendix provides details, experiments and future research leads on a very promising technique: entangled validation. As such, it is the main appendix of this document for it contains contributions that have the potential to be of high practical use. Here we only deal with the evaluation of learning algorithms. In particular we tackle the problem of bandwidth selection of the Kernel Density Estimation (KDE), used to reduce time acceleration and without which BRED can not work correctly with contextual algorithms such as LinUCB. We propose two means to do so. The first one is mostly based on what is done in the literature for standard KDE but is not entirely satisfying. The second one is entirely novel and seems really promising for it completely prevents any overestimation of the per trial payoff due to overfitting. Such a method could also be generalized to evaluate many other online learning problems different from bandits. One has to keep in mind however that the work presented in this appendix, although quite promising in our opinion, is still rather immature and should be pursued further. Consequently we conclude this appendix with a section that highlights the leads we have for future research.*

Keywords Cross validation - Entangled validation - Contextual bandits - Offline evaluation - Replaying past data - Bias control.

Contents

A.1 Introduction

This appendix presents the last but less mature contributions of this thesis work. With the work about BRED in chapter 5, we opened a few perspectives of future work. The main one is obviously looking further into how to design a recommender system by replay evaluations, while performing iterative improvements. We provided a lot of material to get on track but perfecting the approach would definitely require some empirical work. Nonetheless these ideas are slightly outside of what was defined at first for this thesis work, that is the study of online learning algorithms in the context of dynamic recommendation. Indeed, the first purpose of this work was to design and to be able to evaluate learning algorithms. BRED does just that but, as it was showed in chapter 5, it also achieves much more. In this appendix, we set our focus back on our original problem and complete the solution this thesis work provides to it.

For the evaluation contextual algorithms using BRED, we highlighted a key parameter: the bandwidth of the kernel density estimation (KDE). Nevertheless we provided no means to choose that parameter although a bad choice would make BRED useless. Indeed, a too small bandwidth would allow the algorithm to overfit. Its CTR would thus be overestimated. On the contrary, a too large bandwidth would smooth the data too much and prevent the algorithm from learning a correct representation of the underlying structure. In that case the CTR would obviously be underestimated.

In the literature, the problem of picking the bandwidth to obtain the best Kernel Density Estimation of a process given some data was extensively studied [136, 137, 140]. Nonetheless our needs are different from classical variance minimization objectives. Here we want to alter the data enough to prevent a learning algorithm from recognizing the same record presented several times, but not too much in order to preserve the information contained within the data. Furthermore the literature usually refers to optimal values of the bandwidth in $O\left(\frac{C}{T^{1/5}}\right)$, where T is the size of the sample and C a constant whose value is around 1. Our experiments clearly exhibited that interesting values are more in $O\left(\frac{50}{\sqrt{T}}\right)$, which quite different. It is simply explained by the fact that the need for Jittering is greater when T is small and we even have to lose information to make BRED work. On the contrary, when T is large enough we want to keep every ounce of information contained in the data to let the algorithm learn as efficiently as in the real world. Moreover in that case the regularization performed by the algorithm acts exactly the same way as noise added on the records by the evaluation procedure. Indeed, let us recall that the classical regression performed by most regressors (adding the square of all the components of the parameter vector, multiplied by λ), is equivalent to training the same regressor on the same data to which normally distributed noise of variance λ is added. This is by the way one of the uses of Jittering in the neural network field [123].

In this work we first try to choose the bandwidth by adapting a widely used technique in bandwidth selection in the KDE field: cross validation. Then, after arguing that this technique is not completely satisfactory, we introduce a completely novel technique that could in addition be used for other online learning problems. We refer to this new technique of evaluation as *entangled validation*. The procedure is inspired by cross validation techniques but it has major differences. We exhibit in this work that when used conjointly with BRED to evaluate contextual bandit algorithms, entangled validation allows to bypass the bandwidth selection issue, and so without introducing variance. Furthermore we show that entangled validation has a major strength: it prevents any positive bias. Such a feature is extremely desirable in the context of risk minimization. Indeed, if an estimation is positively biased, there is a very high risk to be fooled by an overly optimistic evaluation. It is even more dangerous with BRED as it is now because this bias is not even quantifiable. Note that a negative bias is not desirable either but much less dangerous because if despite this bias, we output a estimation that beats the current algorithm, we can provide an upper bound on the risk. This is true even if the bias can not be quantified. Yet another nice feature of entangled validation is that it allows to quantify the bias. We justify this research with several experiments on synthetic data. Indeed similarly to what we experienced in chapter 5, the evaluation of an evaluation method requires to know for sure the value it estimates. Tricks can be envisioned but we saw in section 5.10.3 that they reduce too much the quantity of usable data within a dataset. Finally, as mentioned in the abstract, many interesting leads to improve the techniques presented here are left unexplored and would be very interesting perspectives of future work. We mention them along the way.

A.2 Bandwidth selection via cross validation does not work

The idea is so simple that it hardly requires explanations. The idea is to consider that the correct amount of Jittering is equal to $\frac{C}{\sqrt{T}}$ (or any other function of T depending on the problem), where T is the size of the dataset and C a constant to determine. To do so we can simply split the dataset randomly into two parts of respective sizes: $\frac{K.T}{K+1}$ for the test set and $\frac{T}{K+1}$ for the training set. This way the test set is K times bigger than the training set. We use BRED on the training set and change the value of the bandwidth until the algorithm we consider reaches the same CTR as what replay outputs on the test set. Replay acts as the ground truth since a sequence it produces can not be distinguished from a real world sequence. This replayed sequence that acts as the ground truth has a size of $\frac{T}{K+1}$ in expectation, which is the size of the training set. We use the result to infer a value for C and extrapolate to get the bandwidth for a dataset of size T. The results of this approach on the synthetic model are given by figure A.1.

Having a look at the figure, one may think that we get decent results. Indeed, the value we find is not far from the best one. There is however a major flaw in what we did here. The choice of a function in $T^{-1/2}$ was based on experiments using our model extensively to generate many datasets of any size of interest to us. Obviously this is impossible with the real world. We only have the insufficient amount of data contained in S. We suspect that the bandwidth function is highly dependent on both the bandit problem *and* the algorithm. As a consequence, we can not even use results from other datasets, and worse from other algorithms on the same data. This make this method completely impracticable.

Nevertheless, we did not presented it for nothing. This method has an advantage. Let us consider that we do not try to extrapolate the optimal parameter for T and keep just the one we find for $T/(K+1)$, the longer sequence we can study without bias. We know that the optimal Jittering parameter for a bigger dataset of size T has to be smaller. Therefore using this overestimation of the parameter prevents any positive bias (overfitting) which is what we want to avoid the most when minimizing risk. Nevertheless, this bandwidth parameter would certainly be way too big and the resulting negative bias would make the method way too pessimistic to uncover any interesting improvements.

As such, this method does not work. Yet we may be able to improve it significantly. Indeed, we do not have access to unbiased estimations for longer sequences than $\frac{T}{K+1}$. Yet we do have access to such estimations for smaller datasets. We could use that knowledge to produce a much more accurate estimation of the bandwidth function by finding the best Jittering parameter all along $\left[1, \frac{T}{K+1}\right]$ and extrapolate. The method would then rely on the assumption that the bandwidth function is regular over T instead of the unrealistic assumption that it is a function of the form $C.T^{1/2}$. Yet again although this assumption is far more reasonable, we have no guarantee that it is true. The method may have potential but studying it is left for future work. Indeed, in what follows we propose and start studying a much more promising and subtle technique to prevent positive bias. However, instead of trying to completely prevent an algorithm from overfitting which turned out to be difficult in general, we will simply prevent the estimator to be fooled by such an algorithm

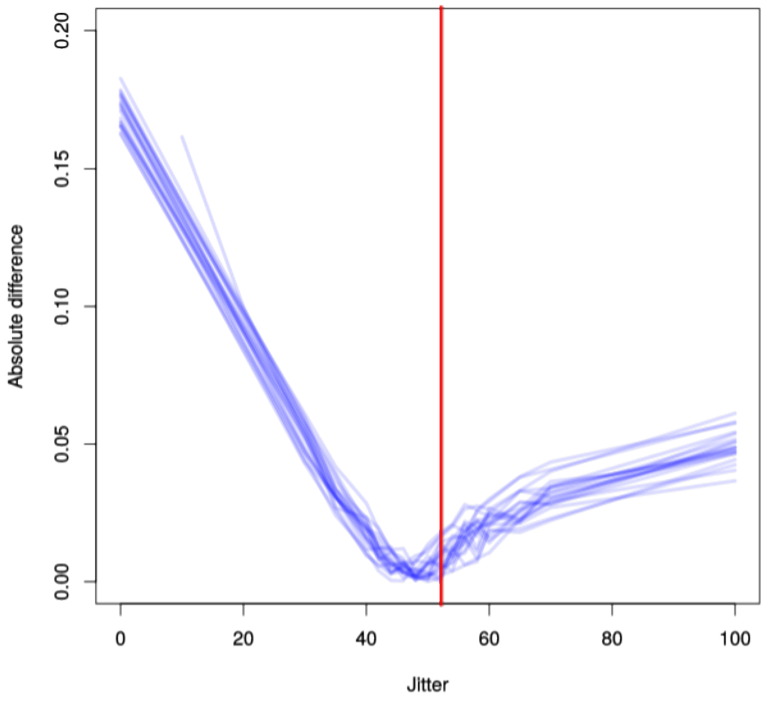


Figure A.1: Absolute difference between the CTR of LinUCB evaluated by replay on the test set of size $\frac{K.T}{K+1}$ and its CTR evaluated by S-BRED on the training set of size $\frac{T}{K+1}$. The results were computed on 20 datasets of size $T = 10,000$. We recall that $K = 10$. The red line accounts for the optimal amount of Jittering for T. For each dataset, the training and test sets were generated 1,000 times at random and the results averaged. Each blue curve corresponds to one dataset. The size of the training set is 909 which is much lower than the original size but necessary to have access to a ground truth via replay. As a consequence, although the bias is very small, the method clearly has one. On average our cross validation method finds $48/\sqrt{T}$ to be the best parameter whereas we know that it is $52/\sqrt{T}$. Notice also that from one dataset to another the results are very stable. This is all the more surprising considering that on such a small dataset, the CTR of LinUCB was found to be very variate.

A.3 Entangled validation: the approach

The previously mentioned approach simply tries to find the best Jittering parameter for BRED in order to use the method as such. The problem we have is that we can only find that parameter for $T' \approx T/K$. Consequently, the quality of the evaluations based on that strongly relies on the assumption that the optimal Jittering parameter is a function of $\frac{C}{\sqrt{T}}$, C being a parameter to determine. Yet we exhibited that this is not entirely accurate. In our experiment, the result were however pretty close. Yet in practice we have no way to know how far we are from the true value. We could significantly improve the approach but we chose another, more innovative path.

We propose to bypass the problem. We saw that it was tricky to try to prevent the algorithm from overfitting. Instead of trying to control the algorithm so that it is close to its real world behavior (which we obviously ignore), we can design an evaluation method that would not be tricked by an overfitting algorithm and would even penalize it.

The idea is the following. Similarly to what is done in cross validation, we divide the dataset into a test set and a training set. We then associate uniformly at random a distinct index between 1 and $K.T$ to each record of the test set. Notice that most indices are consequently not attributed. From there the new method goes as follows. We enumerate each index i from 1 to $K.T$. If that index i was attributed to a test record, we consider that record. Otherwise we draw a record uniformly at random from the training set (which we may choose to Jitter). Then a replay iteration takes place, that is we ask the algorithm for an action to perform given the context, if that action is the logged one then the algorithm gets to learn from the event, otherwise nothing happens. The major difference is that the estimator only takes into account the reward if the record was taken from the test set. Therefore the records that matter are only seen once and overestimating an algorithm that overfits can not happen. Note that this does not prevent overfitting. It simply prevents the estimator from being fooled by overfitting and severely penalizes an algorithm that does so. Indeed by nature an overfitting algorithm will not generalize well. Note that similarly to what differentiates BRED and S-BRED, it is possible to sample from the training set in a different fashion to limit the number of duplicates. The only difference here is that we do not necessarily use a number of records which is a multiple of the size of the training set. Nevertheless this can be dealt with easily. Let us denote by τ the ratio of test data. Then the size of the training set is $(1-\tau)T$ and it needs to be expended to $T - \tau.T$. To do so we simply:

1. create a list of record L made of the smallest number of copies of the training set such that the size of L is greater or equal to $(1-\tau)T$,

2. take a random permutation L^{rand} of L,

3. consider the $(1-\tau)T$ first elements of L^{rand}.

This approach, regardless of how the training examples are sampled, is very similar in spirit to cross validation. The major difference is that instead of leaving apart some of the data for future validation, the test data is entangled with the training data so as to be able to measure how well the algorithm learns. This is why we call this kind of approach **entangled validation** (EV).

Obviously since one iteration only uses part of the data, a nice evaluation method should perform the random separation and the validated evaluation a lot more than once. Also in supervised learning, one sometimes consider 3 distinct subsets. A training set used to train the predictor, a validation set used to tune the hyperparameters and a test set to check for generalization. Obviously to get a true sense of the generalization properties of the trained predictor, one can only use the test set once. Our approach can very well work this way. We may tune a learning algorithm using a training set entangled with a validation set. Then we can verify if it generalizes by entangling the data used in the training phase (training+validation) with the test set.

We named this new evaluation method Tested-BRED or T-BRED for short. Its pseudo code is given in algorithm 17.

Remark Let us consider the simplest version of T-BRED, that is with a training set and a test set. From one bootstrap iteration to another, the training set changes and so does the test set similarly to what is done in cross validation from which the idea of entangled validation stems from. Therefore all the data is used for evaluation in the end. Therefore introducing entangled validation within BRED should not hurt its concentration properties. This is verified empirically in what follows.

A.4 Experiments

A.4.1 Experiments with non contextual algorithms

In this section we are interested in how Tested-BRED works compared to BRED. Because it is so common in Machine Learning and because Tested-BRED has a feel of it, one may

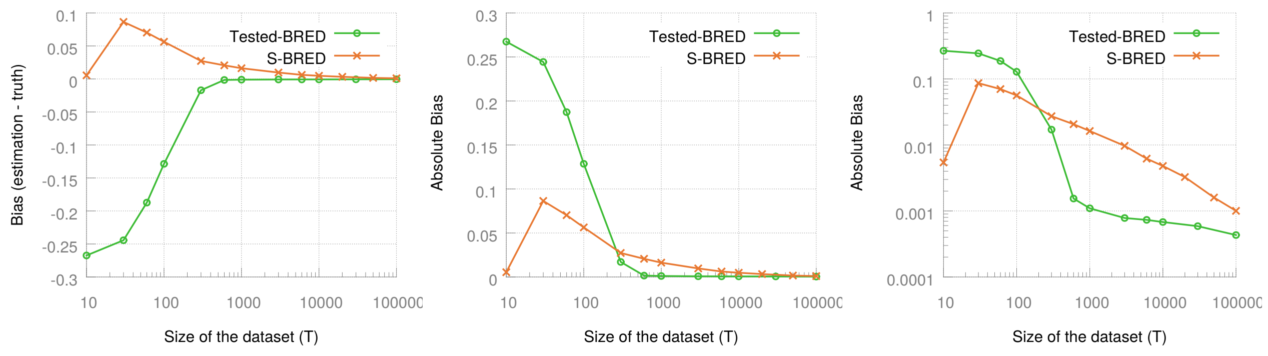
Algorithm 17 Tested-BRED (T-BRED): BRED provided with the entangled validation mechanism so as to prevent any positive bias without introducing additional variance.

```
Input:
  • A contextual bandit algorithm $A$ with a choose and an update function.
  • A set $S$ of $T$ triplets $(x, a, r)$.
  • A number of "bootstrap" iterations $B$.
  • A ratio of test data $\tau$.
Output: An estimate of $g_A(T)$
  $N \leftarrow 0$ ; $G_A \leftarrow 0$
  for each $b \in 1..B$ do
      Divide $S$ at random between $S_{test}$ of size $\tau.T$ and $S_{train}$ of size $(1-\tau)T$.
      $I_{test} \leftarrow \{\tau.T$ indices sampled uniformly at random from $1 .. K.T$ without replacement $\}$
      Create at random a bijective mapping $m : I_{test} \mapsto S_{test}$.
      for each $i \in 1 .. K.T$ do
          if $i \in I_{test}$ then
              $(x_i, a_i, r_i) \leftarrow m(i)$
          else
              $(x_i, a_i, r_i) \sim S_{train}$
          end if
          $a \leftarrow choose(x_i)$
          if $a = a_i$ then
              $update(x_i, a_i, r_i)$
              if $i \in I_{test}$ then
                  $N \leftarrow N + 1$ ; $G_A \leftarrow G_A + r_i$
              end if
          end if
      end for
  end for
  return $G_A/N$
```

assume at first that by introducing entangled validation, we somehow trade off variance so as to remove bias. Indeed we actually reduce the number of records considered for evaluation which, intuitively, may cause higher variance whereas validation may cause lower bias.

We will see in this section that it is true that T-BRED help reducing the bias much faster than BRED or S-BRED. Nonetheless it does not come with additional variance as long as the number of bootstrap resamples is sufficiently large. To study T-BRED we first consider a non-contextual algorithm: UCB. Indeed we saw in chapter 5 that with regard to bias and variance, it behaves exactly as a contextual algorithm. Indeed we recall that figure 5.8 showed that for small datasets, UCB overfits in the very same fashion as a contextual algorithm. The main difference is that the problem is solved with much less data. Consequently considering UCB will allow us to get a sense of how things go at a much lower cost than with LinUCB that, in addition of needing more data to be studied properly, is quite computationally intensive for it inverses a $f \times f$ matrix at each time step (f is the number of features of the context space). The reader is referred to section 2.6 for more precision and possible alternatives.

Let us first leave variance aside and set our interest to the bias reduction yielded by T-BRED. As it can be seen on figure A.2, the improvement is clear. It is also interesting to note that for very small datasets, S-BRED is positively biased (overfitting), whereas T-BRED is negatively biased. It may seem like nothing but it is a crucial feature. In a context of minimizing the risk of putting online a new algorithm, it is very important not to overestimate that algorithm for that would come with a big risk. On the contrary T-BRED is not capable of doing so (at least in expectation), which makes it a much more suitable evaluation method to control risk. The other very nice feature of T-BRED is obviously that its bias converge toward zero much faster than the one of S-BRED. For datasets of size between 1,000 and 10,000, which are the order of magnitude we considered all along this work, the bias is even ten times lower. For T=100,000 the bias of S-BRED is still more than twice greater than the bias of T-BRED. Note that all this indicates what would happen for contextual algorithms but on a larger scale (for T much larger than what is studied here). Therefore the interest of T-BRED in that context is obviously much more important.



(a) Simple bias, regular scale. (b) Absolute bias, regular scale. (c) Absolute bias, log scale.

Figure A.2: Various views of the bias of Tested-BRED compared to S-BRED when evaluating UCB($\alpha = 1$), a non-contextual algorithm over the amount of data. Notice on (a) the negative bias of T-BRED as opposed to the positive one of S-BRED. T-BRED prevents all kind of overfitting (that causes the bias of S-BRED) but lets the evaluated algorithm underfit the data when there is too little of it. Nevertheless notice on (b) that the absolute bias of T-BRED converges much faster to zero that the one of S-BRED. Also (c) shows that when T is between 1,000 and 10,000, the bias of S-BRED is ten times greater than the bias of T-BRED. Obviously this ratio goes to one as T increases but it is still greater than 2 for T=100,000.

Naturally such a bias reduction, as interesting as is it, is irrelevant in practice if it makes the variance take off. Let us remember what happens with replay and its unbiased but more variate counterpart replay* (see chapter 3). The bias reduction is small and so is the additional variance. Yet in general, in statistic, a bias reduction is very often synonym of a variance increase. Figure A.3 proves that this is not the case at all here. The expected squared error (ESE) of T-BRED is even lower than the ESE of S-BRED provided that there are enough computational resources to perform more than 100 resamples. Note that although T-BRED requires more resources, all the methods we proposed so far scale very well horizontally since all the bootstrap resamples can be computed in parallel and averaged at the end. It even fits perfectly the *map reduce* paradigm. Note however that the computation of one bootstrap resample cannot be compressed in the case of learning algorithms since they learn sequentially (this is not the case for stationary policies). From there, a multiplication of the amount of processing units by N yields a reduction of the time complexity by the same factor N.

The conclusion is easy to draw. As long as enough resources (*i.e.* processors) are available, there are only advantages to use T-BRED instead of S-BRED for non-contextual algorithms. Indeed it reduces bias without hurting variance properties and prevents being fooled by overfitting. Moreover its usefulness is not bound to a good choice of the Jittering bandwidth as it is the case for S-BRED.

One important feature is left unstudied although it may seem important: the ratio of test data to be used. In the experiments discussed in this section, as well as is the next section, we used $\tau = 10\%$ as a test ratio only because it felt right. In the field of classical supervised

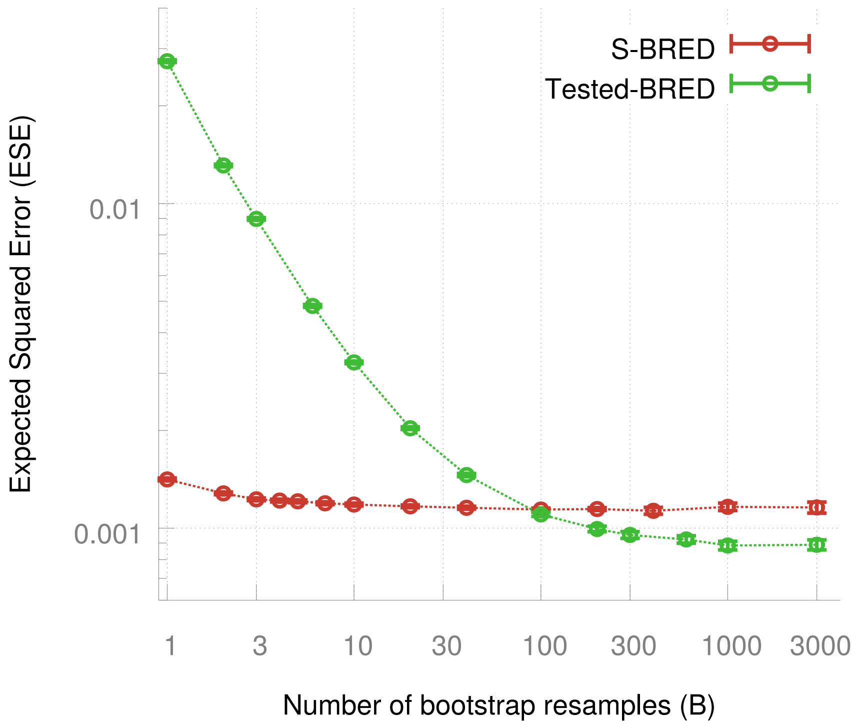


Figure A.3: 95% confidence intervals on the ESE of Tested-BRED and S-BRED over the number of bootstrap resamples when evaluating UCB ($\alpha = 1$) for T=1,000. Notice a key feature of Tested-BRED. Although it clearly requires more computational resources to be accurate, it allows to reduce bias (see fig. A.2) without increasing the variance of the evaluation. Note that contrarily to what seems to appear on the figure, T-BRED is not less variate than S-BRED. Their respective ESE do not converge to the same value because of the relatively big bias of S-BRED compared to T-BRED.

learning, people generally encourage using around 30% of test data, generally less for cross validation meant to select hyperparameters. Here we are doing something that looks like cross validation. Moreover it is rather clear that using too much data would hurt the performance, probably much more than with supervised learning. Indeed, we need to perform a determined amount of interactions and the more we take data aside to validate, the more we have duplicates within the training data. Thus taking too much test data would probably help getting results using less resources but would definitely increase the tendency of an algorithm to overfit and therefore of being penalized (negative bias). Note anyway that all the data ends up being used as test data at some point since many bootstrap resamples are computed. It is possible also that using too little test data, in addition to increasing the amount of necessary resources, would hurt the performance as well. Indeed let us consider the extreme case of only one test record. In expectation it is located in the middle of the evaluation. Thus the bootstrap resample is equal to $g_A(T/2)$ in expectation. Therefore the estimator is biased and it certainly takes a certain number of test records to correct that bias. As a consequence the selection of the parameter τ might be a trade off between a pure bias of evaluation (and a huge number of resamples to converge) and bias due to overfitting (but a more reasonable time complexity). Studying all this is left for future work.

A.4.2 Experiments with contextual algorithms

Entangled validation seems to improve everything when evaluating UCB. Yet it was mainly designed to deal with contextual algorithms and in particular to prevent ourselves from being fooled by an overfitting algorithm. Unfortunately, the experiments of the previous section

Table A.1: Average results of the evaluation of LinUCB using various evaluation methods, including T-BRED that can not be fooled by an overfitting algorithm. The results are averaged over 10,000 datasets of size T=10,000 and we provide 95% confidence intervals. The TL exponent stands for Tweaked learning. It is a variant of T-BRED in which we only let an algorithm learn from records it has never learned from before. Note that although Jittering does not seem to reduce the bias of T-BRED, the TL trick does so significantly.

Method \ Jitter	0	10	52	100
Ground truth	0.507 ± 0.0001	-	-	-
S-BRED	0.564 ± 0.0003	0.558 ± 0.0003	0.508 ± 0.0002	0.476 ± 0.001
T-BRED	0.474 ± 0.001	0.476 ± 0.001	0.473 ± 0.001	0.467 ± 0.001
T-BRED^{TL}	0.487 ± 0.001	0.487 ± 0.001	0.480 ± 0.001	0.468 ± 0.001
Replay	0.418 ± 0.0004	-	-	-
Replay*	0.418 ± 0.0005	-	-	-

already took several days to run on four 2.5 Ghz cores. Contrarily to UCB for which $T = 1,000$ was enough, for LinUCB we need to consider $T = 10,000$ to let it reach its stationary regime. Consequently it would have been quite complicated to run the same set of experiments entirely for LinUCB, especially considering its cubic time complexity compared the UCB's linear complexity.

Nevertheless we managed to pull something off. We do not have error/variance results for they are the greediest. Yet we can assert without too much doubt that similarly to what happens with UCB, Tested-BRED, provided a sufficient amount of cores to run parallel bootstrap resamples, does not come with more variance than S-BRED. Given that they both use all the data for evaluation, this is only natural. Here we only study the bias of the evaluation of LinUCB. This allows us to converge much faster by only running T-BRED (and the other BRED like algorithms) with B=1 on each dataset of size T=10,000 we generate. We were able to generate 10,000 datasets for each method we experiment on here.

It is really intuitive to realize that because T-BRED penalizes overfitting, there is going to be a negative bias. Consequently we can try tackling the issue by introducing Jittering in the same way as what was done with BRED. The results of the experiment are in table A.1. Surprisingly, the improvement entailed by the introduction of Jittering, although statistically significant (*c.f.* the confidence intervals), is very small. A different trick allowed to achieve a much more significant improvement. While replaying the data, we only call the update function of the algorithm if that was never done before for that particular record. The idea behind that is simply to limit the risks of overfitting. We call this trick "tweaked learning". In that case Jittering becomes completely useless. Finally as it can be seen in table A.1, a significant bias remains. Yet this bias is much easier to deal with when controlling risk than the one S-BRED potentially has in real life because we know for sure it is there and we know it is negative.

The very nice point that this experiment shows is how well we do compared to replay. With our best approach (T-BRED with the small tweak), we have a bias of 0.02 when replay has a bias of 0.09. In other words, the bias is divided by 4 which is a tremendous improvement. Let us recall that our goal was not to find a perfect estimator but to reduce the effect of time acceleration without introducing uncontrollable bias and possibly reduce the variance. We showed in chapter 5 that the latter was achieved with a great margin by BRED and S-BRED compared to replay. Yet we had introduced an uncontrollable bias to do so. Here we demonstrated that T-BRED does even better with regard to variance. More importantly, with T-BRED and entangled validation, we removed the uncontrollable bias that was entailed by BRED and S-BRED while maintaining a tremendous improvement with respect to replay.

This is basically as far as our empirical work goes on the evaluation of contextual bandits.

Before mentioning a few perspectives of future work that would aim at reducing that negative bias further, we can somehow put the amount of bias entailed by T-BRED into perspective. First of all we were evaluating LinUCB with a standard choice for regularization ($\lambda = 1$), similarly to what we had done all along chapter 5. We had exhibited in sections 2.6 and 4.4.1 that greater values of λ could work better. In fact because we are dealing with duplicates, an evaluated algorithm is much more prone to overfitting the data. Yet it is very likely that an algorithm regularized with more care would be less affected and its evaluation less biased. Note that in practice an evaluation method that pulls strongly toward more regularized algorithms is not necessarily a bad thing. Indeed, such algorithms typically have a much more stable behavior which enables to have a better control over the risk of putting them online for they have less variance. All this remains however purely theoretical and should be verified empirically in some future work. The second argument that puts the relatively important bias of T-BRED into perspective is that $T = 10,000$ (or 1000 records per possible action) is relatively small, in particular considering that we learn from scratch. In the Yahoo! Today dataset [12], there are roughly 40 to 50 news article a day, which represents two millions records. This gives 40,000 records per news articles which is much more. The Yahoo! Today dataset comes with more features (135) so it may take more records to learn than with the 15 features of our model but not 40 times more, especially considering that the Yahoo! features are binary. Furthermore, in Yahoo! Today we do not learn from scratch. Not all the news are replaced at the same time. Finally figure 5.8 from chapter 5 exhibited that for $T = 100,000$, the bias of S-BRED *without Jitter* was starting to be reasonable. We also exhibited in the previous section that the bias of T-BRED converges much faster toward zero than the bias of S-BRED. Consequently, if we assume the very likely event that the bias of the evaluation of LinUCB as a function of T is similar to the bias of UCB but on a larger range (we recall that this is the case for S-BRED: see figure 5.8), T-BRED would certainly have a bias close to zero for $T = 100,000$, and probably even before that. In any case, let us recall that the improvement in terms of bias and variance compared to replay, the only alternative before our work is already spectacular.

We do believe that it is already possible to work with T-BRED as is with real data and real applications. It has a bias but which will turn out to be quite small if enough data is available. Furthermore that bias is negative, which prevents overestimation which is what we want to avoid at all cost. That is not all. Even for small values such as $T = 10,000$, it is probably possible to reduce that bias further and make T-BRED more efficient by using additional tricks of the flavor of the tweaked learning we already introduced. In the following section, we mention a few interesting leads.

A.4.3 Discussion: explaining the negative bias

The main idea we have is based on the results about information/knowledge that we acquired in chapter 5. With Jittering and tweaked learning, we can be confident enough on the fact that overfitting is limited to a minimum when evaluating LinUCB with T-BRED. Therefore how come the evaluated algorithms are underestimated ? The answer is very intuitive. We saw in section 5.10.2 that interactions simulated by replay on expanded data were less informative with regard to the CTR of the algorithm than the same number of real world interactions. This was also true when evaluating static policies. Basically this also means that N simulated interactions are less informative than N real world interactions for the algorithm. As a consequence, its learning process is bound to be less efficient. In other words, LinUCB learns slower when evaluated by T-BRED (or any other BRED) than in the real world. With a static policy this only yields more variance. Here it causes a negative bias as well. Note therefore that overfitting is merely a side effect of that phenomenon. The main reason for the negative bias is not a penalization of overfitting by entangled validation. Rather the negative bias comes from simulated interactions that are slightly less efficient, which causes an underestimation of the learning capacity of the evaluated algorithm. This is why Jittering, which is only one way to deal with overfitting failed to reduce the bias significantly in this context. The slight improvement that we noticed is very likely due to the fact that as we studied it in section 5.10.1, Jittering does also allow to uncover a little bit of information from the data combined with a slight reduction of overfitting.

A.5 Additional perspectives

The raw idea of entangled validation and the little work we presented here is already quite mature. The goal was to select the Jittering parameter without information on the underlying model of the data. Indeed, a bad choice of parameter leads to under-fitting (if the parameter is too big, we destroy information) or worse to overfitting (if the parameter is too small) which is extremely dangerous when trying to limit the risk of putting a new algorithm online. The main issue was obviously that in practice, we had no way to say which of the two was happening. We first identified that cross validation was not an option. We then proposed a method - entangled validation - that possesses very interesting properties: (i) it does not introduce extra variance, (ii) it prevents any positive bias which was our primary goal and (iii) as a side effect, selecting the bandwidth of the kernel simply becomes a stochastic optimization problem with one clear peak. Finally we identified the main issue of this technique: under-fitting (and thus negative bias) due to the fact that interactions with our estimation (by bootstrap) of the underlying bandit distribution are less informative that interaction with the true distribution.

In this last section we propose perspectives to improve and enrich this work that were much mess studied that the raw concept of entangled validation. We firstly mention how we could use expansion and secondly models to improve entangled validation and so without introducing positive bias. Finally T-BRED is a mature and safe (it forbids positive bias) enough to be used on real data. We discuss this perspective in a third part of this section.

A.5.1 Reducing bias via additional expansion

A naive way to tackle this new issue would be to simulate more than T interactions. Nonetheless this would obviously create more problems that it actually solves. Indeed, there exists a number of interactions greater than T for which the expected CTR evaluated by BRED is equal the CTR of the algorithm when played for T time steps against the real world. The problem is that we have no way to know that number of interactions. Also it is very likely that this number tends to T as T increases but we do not know how fast. Therefore knowing that number of interactions for $T' = \frac{T}{K+1}$ using the same cross validation approach we proposed for Jittering (see section A.2) would be of no help. The worst part to keep in mind is that as soon as we expand the data to simulate more than T interactions, T-BRED loses all its guarantees on the bias. It can be negative or positive (which is very risky) and we have no way of knowing which one it is.

Fortunately we can work out a few solutions anyway. We justified in chapter 5 that no matter how many times we replayed N records, it was not possible to drag more knowledge out of them than N real world interactions (or N fully labeled records). The idea is quite simple. With T-BRED, we start by setting validated milestones that are used for the evaluation. These milestones are positioned uniformly between 1 and $K.T$ exactly as introduced for T-BRED and entangled validation. As far as the simulated interactions are concerned, we can consider that they are positioned between $i = 1/K$ and $i = T$. Using the latter notation, between a milestone at position i_{k-1} and a milestone at position at position i_k, we can only allow to sample from a restricted subset of the training set of size at most i_k. Note that we say at most because the training set is necessarily smaller than T. From there, we can keep this way of fractional indexing between $1/K$ and T and expand the data as much as we want. More formally, this means that the $\tau.T$ milestones (the test set) are positioned uniformly over $[1..K.T.E]$ where E is an arbitrary additional expansion factor. Then we divide all the index by $K.E$ to go back to the indexing we talked about earlier. Since when evaluating a milestone at position i, it is impossible for the algorithm to have acquired more information than i fully labeled records, we can not get the positive bias we may be afraid of. Indeed, even with a big value of E, with the fractional indexing, i can not be bigger than T. The intuition justifying this idea is the following. An algorithm is underestimated with T-BRED because information is not concentrated enough within the simulated interactions. Yet there is more information contained in the data than what one bootstrap resample reveals (see how S-BRED's accuracy increases with B). Thus our additional expansion allows to use that additional information to make the learning of the evaluated algorithm more efficient and reduce the bias of its evaluation. This is done without forfeiting the impossibility of a positive bias by controlling the sampling set at each fractional index.

There are a few issues that we would have to deal with in practice. First the evaluated algorithm would be more prone to overfitting because of the greater number of duplicates, in particular at the beginning. Moreover to avoid funny behaviors at the beginning when the sampling subset is very small, we may want to introduce a very small bias and set up a lower threshold on the size of the subset we sample from. We do not provide the implementation of this approach but one could easily derive it from T-BRED's implementation (algorithm 17). The main modifications would be to change the indexing and to only allow to sample from the corresponding subset of the training set at each step. An empirical study of this method that we could call T-BRED∞, although we would not necessarily need to expand infinitely, would be very interesting and is left as future work.

A.5.2 Entangled validation for models

In most recommendation textbooks [3, 24] models are not trusted when it comes to evaluating the precision of a recommender system. See chapter 1 for more details on the subject. They are only used for very specific purposes such as profiling or even debugging. Avoid their use by developing purely data driven methods is actually one of the main motivations of this thesis work. But why are they not trusted? Because they are biased by nature! Here we proposed a method that does not control the bias of the data used to play an algorithm. Indeed, when using T-BRED, an algorithm can very well overfit. Yet thanks to entangled validation, the evaluation method is not fooled by that and can not be positively biased. In other words, although the interactions are biased, the bias of estimation is kept under control.

The problem we have with T-BRED is that the records that are duplicated do not contain enough information. T-BRED∞ would probably do better about that and solve part of the problem. This is clearly a purely data-driven or non parametric way of dealing with things. Nonetheless we could think of another way to tackle the lack of information. We could build a model and use it to generate the training set. Then we could control the bias of the evaluation via entangled validation, that is by introducing test records that come from the data. Obviously note that if the model is build from the data, part of it still has to be left aside for testing. As a consequence, all the training interactions, between the validation milestones would be with fully labeled records. This would clearly introduce a bias on the data the algorithm uses to learn. Yet this would allow the algorithm to learn much more efficiently and we would have the validation set to prevent the bias of the data from positively biasing the overall evaluation of the CTR. Note that this approach is in fact very similar to what we are doing with Jittering. Indeed, a Kernel Density Estimation is in fact based on the assumption that the context space is smooth. It might consequently be considered as the coarsest model.

Such an approach could also be used to evaluate the quality of a model by comparing the validation CTR with the CTR on the synthetic records.

A.5.3 Validating T-BRED on real data

Before jumping to the conclusion, let us talk a little bit about real data. Our work is mostly justified by an empirical study on a model of our design and by theoretical guarantees. Indeed note that if we did not provide any mathematical proof to describe T-BRED's properties, we provided intuitive arguments that prove its non positive bias as well as its variance of the same order as all the other BRED procedure provided that B is large enough. Nonetheless it would have been quite a nice conclusion to this thesis work to quantify the bias due to time acceleration when evaluating LinUCB and to correct it with T-BRED on the data of the challenge we organized (see chapter 4). Indeed, this challenge was the starting point of this work on bootstrapping and entangled validation. Unfortunately this has not been possible in the limited time we had to carry on this work. Indeed the reader should keep in mind that manipulating LinUCB is extremely computationally demanding. It takes approximately 5 hours on a 2.5 Ghz and with a standard hard drive to evaluate it on the data of the challenge with replay. With BRED it would take K=40 times that time. Adding additional expansion as suggested in the description of T-BRED∞ would worsen the situation by that much. Moreover since this is a sequential problem, it can not be parallelized. Note however that the fact that T-BRED requires at least 100 resamples to be accurate is less problematic as they can be parallelized. Yet one needs to have that many cores available which we did not have at the time this work was conducted. We tried to compute a Principal Component Analysis (PCA) on the feature set to accelerate LinUCB but we did not get really good results. Indeed the numerical results we not very impressive and still quite difficult to be made statistically significant for even with ten features, LinUCB is still greedy in resources. PCA was certainly not the way to go here anyway. Indeed with binary features, and in the Yahoo! today dataset [12] in particular, the distribution of the variance over the different features has a very heavy tail. Consequently a PCA loses a lot of information, which is the key to make T-BRED work.

All this justifies even more the relative irrelevance of LinUCB in a real time system for if it can not be evaluated offline, it is not very likely to be put online. Moreover it may not even be able to stand the responsiveness that a web application requires. Therefore let us highlight once more the significance of our work on an algorithm which is able to perform roughly as well as LinUCB while being faster by several orders of magnitude (see section 2.6 for our work on LAP, whose time complexity is linear *versus* the cubic complexity of LinUCB entailed by the systematic matrix inversion).

To conclude, an empirical study of T-BRED based on real data would be of utmost interest with regard to this work. Unfortunately it is obvious that this is what this thesis misses the most. It is far from impossible but would require a few cores for a few days to be able to begin to get significant results. Nonetheless given the theoretical guarantees we have on our methods, especially on T-BRED∞, and the empirical evidence we have that the Yahoo! today dataset [12] contains enough contextual information to outperform a most popular item policy (see section 4.4.3) we do believe that things should work out in the end. It only needs a little bit more of empirical work to demonstrate it properly. Furthermore, plethora of models of user behavior exist in the literature [77, 78, 114] and could be used in conjunction with entangled validation to obtain very interesting and non positively biased evaluations.

A.6 Conclusion

Chapter 5 was a rather comprehensive introduction of a new methodology based on resampling that we refer to as data expansion. We justified theoretically as well as empirically all the tools that were used conjointly such as our information measure, bootstrapping, Jittering *etc* and exhibited how they solved the problems raised by the simple use of replay. In that chapter we also showed how to apply our resampling methodology in other situations such as a global and iterative approach to the design of a regular recommender system. Here we reset the focus on contextual bandit algorithms and their applications to dynamic recommendation. The main issue we had at the end of the last chapter was that although we knew that there existed a method that evaluates fairly such algorithms, we had no way of designing it in practice. More specifically, we had no way of knowing what bandwidth to use when Jittering.

In this work we first provided a very naive method based on cross validation, a widely used tool when it comes to selecting the bandwidth in Kernel Density estimation in particular or to select any hyperparameter in general. Nevertheless in our situation the optimal bandwidth depends on the size of the dataset and we are only able to validate a bandwidth on a much smaller portion given the amount of data needed for validation. We do have an idea of the shape of that dependence but this idea is only based on intuition and is far from being accurate enough. As a consequence we proposed a second approach called entangled validation which, to our knowledge, is completely novel. The major strength of the evaluation method based on entangled validation, Tested-BRED, is that it can not be fooled by an overfitting algorithm. Consequently in expectation it cannot overestimate the CTR which is very comfortable when trying to minimize the risk of putting a new algorithm online. Moreover this new approach has no additional cost in terms of variance. This is rare enough to deal successfully with a bias situation without hurting the variance properties to be highlighted. Unfortunately this work is still rather immature and the negative bias of Tested-BRED might still seem important. Yet we provided arguments to put that bias in perspective that seem. They indicate that this bias may be less important in practical applications than with the empirical example we took. In addition we proposed an extension of Tested-BRED, Tested-BRED∞ that would most likely help us reduce the bias further. We finally proposed a model based approach that would help to achieve the same purpose. We did not study these last extensions but it is a very interesting perspective of future work.

Finally we said a few words about the lack of results based on real data in our work on evaluation methods for contextual bandits. Note however that evaluating our methods on real data is obviously very complicated given our need for the ground truth. Yet it may have been interesting to exhibited a lift of performance when evaluating LinUCB with T-BRED compared to a replay evaluation that we argued to suffer from time acceleration in chapter 4. The computational greed of LinUCB as well as the requirements in terms of resources of the rejection process on which replay is based and of entangled validation made that quite complicated. Unfortunately there is not much we can do about the evaluation method besides considering a validated model based approach which would almost completely rid us of rejections (they would only be necessary for the validation records). Yet we could consider different algorithms, less computationally expensive as LAP (chapter 2). Our will to keep considering LinUCB was mostly driven by the fact that this is basically the only generic algorithm available in the literature. It would have been confusing at best and very controversial at worse to do research in which we design both the algorithm under evaluation and the evaluation method. As we already mentioned it, finding evidence that our methods work with real data would be very interesting. Yet we do believe that the most exciting perspective is to just use T-BRED and its variants to design new contextual algorithms oriented toward real applications. In the past, researcher did not have the possibility to validate their new ideas on real data. There existed two non satisfactory alternatives: synthetic models and live evaluations. This is most likely the reason of the lack of diversity in the contextual bandit algorithms available in the literature. This thesis work is a great chance to change that. Standard benchmarks could be derived from T-BRED and standard datasets in order to encourage the development of a wide set of algorithms. In particular in view of the study of the Yahoo! Today dataset [12], some future algorithms could among other things:

- perform dynamic dimension reduction (via clustering, feature selection or other means),

- pay attention to the dynamic aspect of the problem (CTR decay, taste evolution, environmental factors such as politics, whether forecast...),
- consider shifting to a most popular news approach in specific cases.

Obviously one should not forget about the main aspect of contextual bandits that is learning online an accurate model of the environment. The linear and additive model based on independent features advocated by the authors of LinUCB might be too simplistic to capture the complexity of the tastes of a community of very diverse users.

To sum up, although the very end of this work about BRED is not yet completely mature, together with the appearance of publicly available datasets such as the R6B from Yahoo! [12], it opens gates to many researchers toward very exciting perspective of research about real world online learning systems.

Appendix B

An equivalence between classification and recommendation and the derived evaluation method

Abstract of the appendix *This appendix presents a minor - yet interesting - contribution of this thesis. This contribution is a method - classify to recommend - that allows to use classification on CTR data in order to evaluate or build a recommendation policy. This link with classification opens a gate toward putting to use of all the existing classification methods that are all over the literature in the context of recommendation. In particular in this appendix we:*

1. *present the "classify to recommend" method,*
2. *prove a few theoretical guarantees,*
3. *discuss its strengths and shortcomings.*

We also show a link between this method and replay which allows to speed up a replay* evaluation by a factor* g_π^{-1}.

Keywords Classification - Recommendation - Offline evaluation - Contextual bandits

Contents

B.1 Introduction

This appendix deals with a procedure, different from the *replay method* that we just dissected and that at some point we considered using to evaluate contextual bandit algorithms on CTR data. We came up with this idea while preparing the presentation of the algorithm who won the Exploration *vs.* Exploitation challenge 2 given at the eponymous workshop held at ICML-2011.

At the time the *replay method* was not very well-known as the paper which mainly contributed to its popularization had just been published [11]. We were wondering what kind of conclusions we could draw from the relative failure of the evaluation method of the challenge. Moreover we were after alternatives to perform evaluations, and perhaps a new challenge. Consequently we concluded the paper we wrote on the challenge [5] by a presentation of a this new method of ours along with a theoretical guarantee.

This method is a simple way of transforming the recommendation problem into a multi-class classification problem and to evaluate a recommendation *policy* without bias. Yet as we will show it in this work, this new method as far as only the offline evaluation of *algorithms* is concerned has less interesting properties than the *replay method*. This is basically what will be dealt with in the first part of this appendix. In the second part however we will argue that this method remains quite interesting. Indeed among other things, the bridge with classification - a relatively old and mature field of research - allows us to use the plethora and diversity of classification algorithms that can be found in the literature in order to perform recommendations.

B.2 The *classify-to-recommend* method

First let us consider as in chapter 3 the simplified problem of evaluating a recommendation *policy*, that is to say a non-learning or static algorithm. To do so we assume that we have a dataset S acquired using a random uniform policy on a contextual bandit distribution $\mathcal{D}$ with Bernoulli rewards exactly as in that chapter. Also, before going any further let us describe quickly the *classify to recommend* method. In a nutshell it consists in:

1. building a dataset S_1 by taking only the records associated with a reward of 1 in S,
2. considering the classification problem that maps a context (input variables or features) to an action (class),
3. computing the hit rate $q_\pi(S_1)$ for that problem of the evaluated policy π - Note obviously that if S_1 is used to train the classifier/policy, which is very often the case, q should be computed on a subsample of S_1 left aside during training (test set) in order to avoid bias,
4. multiplying the result by $\frac{|S_1|.K}{T}$ with K and T being the sizes of respectively the action pool and S.

The detailed procedure is given by algorithm 18. Note that in the conclusion of Nicol *et al.* [5] we did not talk about the fourth step since the result we had at the time was not as mature as the one we propose here.

In what follows we basically show that it is possible to use the resulting classifier to make recommendations on the real problem. Such a recommendation will have an expected CTR of:

$$q_\pi(S_1)\frac{|S_1|.K}{T} ,$$

which is consequently proportional to the classification hit rate.

B.2.1 Early result

The point of our early and immature work presented in Nicol *et al.* [5] was to propose a problem that could be evaluated using the dataset we had at hand and whose best policies/algorithms would at least give us information as to how to deal with the actual recommendation problem (that we denote with P_R in this section).

Algorithm 18 The classify to recommend method.

Input

- A dataset S of triplets (x, a, r).
- A classifier trainer *train* that given an array of examples X and an array of their respective classes Y, outputs a classifier π that maps examples to classes.

Output

- A recommendation policy π
- An estimation of its CTR on $\mathcal{D}$.

```
split $S$ into $S_{test}$ and $S_{train}$
$S_{1,train} \leftarrow \{(x, a, r) \in S_{train} | r = 1\}$
$S_{1,test} \leftarrow \{(x, a, r) \in S_{test} | r = 1\}$
$X \leftarrow \{x \mid (x, a, r) \in S_{1,train}\}$
$Y \leftarrow \{a \mid (x, a, r) \in S_{1,train}\}$
$\pi \leftarrow train(X, Y)$
return $\pi$
return $\hat{g}_\pi(\mathcal{D}) = q_\pi\left(S_{1,test}\right) \frac{|S_{1,test}|.K}{S_{test}}$
```

In fact we came up with two similar problems. The first one is the classification problem on S_1 that we just described. We note it P_1. The second one is the very same classification problem but on S and considering that the records with a zero reward have no class. That is to say that they can never lead to a hit. Note that the multiplying factor from the fourth step becomes $\frac{KL}{L} = K$. This problem is useless as far as evaluation is concerned but we had two ideas when we thought about it. First to let the *one versus all* classification algorithms use more data. Second to introduce more noise within an online version of the classification process we were interested in. In this online task, a label is revealed only if it is correctly classified by the algorithm so as to make it more difficult and measure the learning speed. We denote this problem with P_{1+0}.

We then were able to prove the following theorem:

Theorem 22. *Let us consider the following notations.*

- *Let π^*_R be the optimal policy that maps a context to an action for P_R.*
- *Let π^*_1 be the optimal policy that maps a context to an action for P_1.*
- *Let π^*_{1+0} be the optimal policy that maps a context to an action for P_{1+0}.*

For any the contextual bandit problem $\mathcal{D}$, any dataset S acquired by T interactions of a random uniform logging policy with $\mathcal{D}$, we have that:

$$\pi^*_R = \pi^*_1 = \pi^*_{1+0} \ .$$

Basically this proves that the best policy for any of the two problems we mentioned is also the best policy for the actual recommendation problem. This and the fact that the evaluation method does not allow an algorithm to make a choice between several users made us quite confident that we would obtain much more interesting results than with the one defined for the challenge. We do not prove this result as it is merely a corollary of the new result we have that fully justifies the statement *equivalence between classification and recommendation.*

B.2.2 New result

The new result is actually much simpler and much more interesting. It says that using the classify-to-recommend (C2R) technique, we can build an unbiased estimator of the CTR of a policy. Note that when evaluating a static policy the zeros are useless so we will not talk about P_{1+0} in this section. We will also see in the experimental section that this problem has in the end little interest, at least if we omit the online problem it allow to formulate.

Theorem 23. *For any contextual bandit problem $\mathcal{D}$, any policy π, any dataset S acquired via T interactions of a uniform logging policy on $\mathcal{D}$ and with $S_1 = \{(x,a)|(x,a,1) \in S\}$, the hit rate of π on S_1 (if we consider x as the input and a as the output) that we denote by $q_\pi(S_1)$, is unbiassedly proportional to the true CTR of π on $\mathcal{D}$ (g_π). The coefficient of proportionality depends on the problem but is easily computable which makes the following quantity an unbiased estimate of the CTR of π.*

$$\hat{g}_\pi^{(C2R)}(S) = \frac{|S_1|.K}{T}.q_\pi(S_1)$$

Theorem 24. *With the same assumptions, the estimator of $g_\pi(\mathcal{D})$ that results from theorem 23 has the following variance:*

$$\frac{K}{T} Var(\vec{r}[\pi(x)]) + \frac{K-1}{T}\mathbb{E}\left[\vec{r}[\pi(x)]\right]^2 \ .$$

Note that the variance is actually the same as the variance of the *replay method** and is thus comparable with the variance of the *replay method* for reasonable values of K and T which is quite surprising considering the fact that we only have to perform the evaluation on a small part of the dataset. In particular, in the context of news recommendation the average CTR is typically between 1% and 10% which results in a speed up of a factor 10 to 100. The explanation of this similarity comes with the proof of those two results.

Proof. The proof is very simple:

$$q_\pi(S_1) = \frac{|\{(x,a) \in S_1|\pi(x) = a\}|}{|S_1|}$$

Thus:

$$\begin{aligned}
\hat{g}_\pi^{(C2R)}(S) &= \frac{|S_1|.K}{T} \cdot \frac{|\{(x,a) \in S_1|\pi(x) = a\}|}{|S_1|} \\
&= \frac{K}{T} \cdot |\,\{(x,a) \in S_1|\pi(x) = a\}\,| \\
&= \frac{K}{T} \cdot |\,\{(x,a,r) \in S|\pi(x) = a, r = 1\}\,| \\
&= \frac{K}{T}\sum_{t=1}^{L} V_t.r_t \ .
\end{aligned}$$

The last line is just a translation using the notation used in the chapter 3 and to more clearly exhibit that the estimator resulting from the evaluation using the *classify to recommend method* is strictly the same as the one resulting from the *replay method**. Thus this new estimator has the same bias (none) and variance. □

Basically as far as the evaluation of static policies is concerned, *classify to recommend* has similar guarantees than the *replay method*. Its major advantage is that it only needs to go through the clicked events. One may argue that it can only handle CTR data and not real valued rewards. This is not true. Classify to recommend as we presented it is just a special case of a procedure who would give a weight equal to its reward to each record. In the real valued case however, there would not be any gain of time (except if there are still a lot of zeros).

Note that it is also possible to use the procedure with fluctuating pools of items by weighting each item i at time t with the following factor:

$$\frac{\sum_{t=1}^{T} I(i \in \{1..K_t\}).K_t}{sum_{t=1}^{T} K_t}$$

Nevertheless we gave up on trying to use this method in practice to evaluate online learning algorithms because it suffers from more severe flaws in this setting as we will see it in the next section.

B.3 Using the method

First classify to recommend allows to generate a recommender system given any classification technique. The classification technique optimizes the hit rate which is proportional to the CTR. Therefore by simply taking a classification method out of the box, we obtain a technique that tries to build the best recommender system it can. Considering the tremendous amount of work that was done over the years in the classification field, this is a very interesting result for recommendation.

This procedure can not however be used to evaluate online learning algorithms. Here is why. The greatest strength of the replay method to evaluate learning algorithms in that all the histories of events have the same probabilities in the real world and under an evaluation (Theorem 2). This means that if the dataset is big enough, an algorithm will behave exactly the same way when replayed on a dataset than in reality.

With *classify to recommend*, this is not the case. Basically there are two ways to turn the classification problem we defined into an online problem.

1. The most straightforward way is simply to consider each record one at a time, to ask the algorithm we evaluate to classify that record and then let it learn from the outcome. This is exactly what is defined in the literature as *online classification*. The obvious flaw here is that since the outcome is revealed no matter what, there is no need for exploration. Thus by evaluating an algorithm here, we measure its ability to learn but not its ability to handle the exploration *vs.* exploitation dilemma.

2. The second way is the one we were actually considering when we designed this method. The idea is simply to only reveal the label (the action), if the recommendation was successful. Therefore to learn something about an action, one has to "explore" it. The is how the aforementioned dilemma is introduced. This method has two very distinct flaws. A successful algorithm for that task would have to balance efficiently between exploration and exploitation. Nevertheless the first flaw we mentioned is that the online learning problem that we defined is different from the original problem. Therefore there is no saying that a successful algorithm on this new task would perform well in dynamic recommendation. Yet we may reasonably assume that given how close the tasks are, there would be correlations and we would learn something. The main issue is the second one. A replayed algorithm gets to learn from one Kth of the data. Here, given that we would prevent learning if the algorithm does not hit the correct class, the algorithm would only get to see a fraction CTR/K of the data with CTR the click through rate of the logging policy. On the Adobe dataset of the challenge described in chapter 2, which was very big and that only contained 6 items, it was a viable option. It might not be the case with more items and less data, although it would be interesting to study this point further. We will not do it in this thesis work for we chose to focus on replay.

Yet even though we chose to discard this approach as an evaluator for online learning algorithms, we do think that it is not completely useless with that regard and could be used for other purposes that we will detail in section B.4.

B.4 Further discussion

In the end we realized that we could not really use this method for online algorithms which is our purpose. Yet this bridge with classification really seems worth talking about and this is what we will do here.

The first good point about this method is something we already discussed: it can speed up the evaluation process if someone wants to evaluate a static policy. Yet one may be skeptical about the interest of evaluating such a policy. First in classical recommendation this is all we do. Nonetheless the K-dependent convergence rate may prevent us from using this method or replay in that context. We may however think of very interesting ways to dodge this issue. For example one may wonder which way to perform recommendations is the best depending on the context. Indeed there are many ways to design a recommender system that are quite appealing: frequently bought together, bought by similar users, most popular, users who visited that page bought this, social networks *etc.* We could very well design a set of recommendation policy and allocate them randomly between users for a time. Then with replay or this method (that provide a significant speed-up) one may evaluate without bias a policy balancing those policies depending on the context.

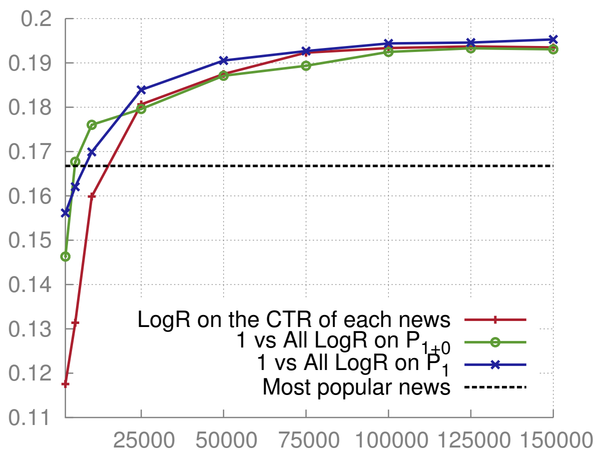
Another interesting feature of this method is the bridge that it builds between classification and recommendation. Indeed let us consider the following idea. Let us assume that we have a dataset and want to design a learning algorithm for a dynamic recommendation application. Where can we start knowing that there exist only a few contextual bandit algorithms out there that we can try? An idea could be to keep only the clicks and use the plethora of classification methods that are available online and that require no engineering effort (since most of them are already implemented) in order to learn things from the data. In comparison, a similar and more basic idea would be to train various regressors that, for each action, map the context to the reward. We could then consider and evaluate the policy that consists in picking the item whose regressor outputs the highest click probability. There are a few differences between these two approaches:

- First the second way is much more greedy in terms of resources: K regressors to train, T records to use whereas *classify to recommend* only requires one classifier to train on a few percents of T records (only the clicks).

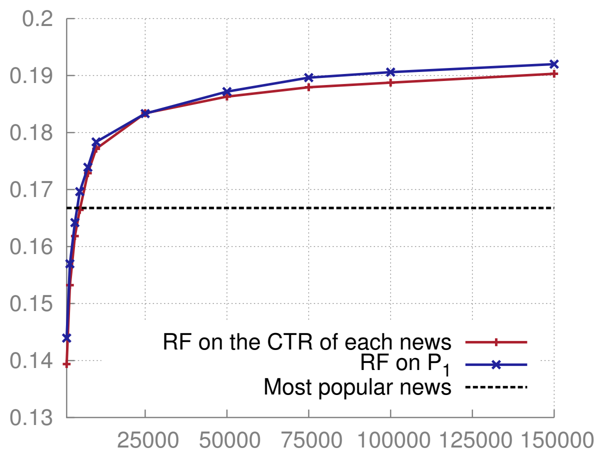
- The multiple-regressor method also requires much more engineering effort. Indeed one has to write code to build the policy interrogating the regressors, gathering the results and evaluating the policy on the dataset. On the contrary most classification libraries available online provide scripts to train on some data while keeping a fraction of it to output a hit rate computed on a test set.

- Finally and this is the main advantage in our opinion, the training procedure of the classifier will aim at maximizing a quantity (the hit rate) which is proportional to what we want: the CTR (see theorem 23). On the contrary the multiple regressor approach optimizes the performance of each regressor which does not ensure the performance of the gathering of their results.

The respective performance of the various classification approaches would definitely give us hints as to how the online problem should be tackled. For instance we exhibit in the chapter 4 that random forests [106] perform very well on a dataset provided by Yahoo! from a news



(a) Logistic regression.



(b) Random forest [106].

Figure B.1: Supervised learning on CTR data. The curves are the hit rate obtained using the classify to recommend method. Higher is better. Three approaches are compared: (i) in red: K separate regressions to predict the CTR of each news and then select the one with the highest likelihood. (ii) in green: Mapping contexts to actions with *one versus all* classification on P_{0+1}, that is the non-click record are always considered in the "all" category. (iii) in blue: Classification on P_1. The second one is not possible with random forests but note that even though the latter uses less data, it seems more accurate. The second one seems better for very small datasets.

recommendation application [12]. As a consequence, an online learning approach based on an ensemble of trees may seem like a good start. We only tried a few libraries but the key argument here is that we did it almost for free given that trusted implementations of many classification techniques are available online and can be tested. It would have taken much more time to get this information by trying to design from scratch various new contextual bandit algorithms, which would have had to be tested, tuned *etc.* In conclusion, using this method as a cheap yet highly informative diagnostic tool based on the tremendous amount of mature work on classification seems like a very promising idea. This is the main use of this method, although we are rather certain that other uses could be thought of based on this interesting bridge between classification and recommendation.

One may argue that despite all these advantages, as *classify to recommend* only learns on the clicks whereas the multiple regressor technique gets to learn on all the data, the latter should obviously perform better. Note by the way that LinUCB is an online multiple-regressor so this alternative method is far from being a lunacy we made up.

To check on that, we designed a little experiment. This experiment is based on the R6B dataset from Yahoo! [12] which is described at length in chapter 4. This dataset fits the requirements of this method and was acquired on a news recommendation application. In this dataset the pool of news is dynamic so to make things simpler, the dataset we use here is the longest part of this dataset in which the same ten news remain alive together. We only kept the records concerning these news and obtained a dataset of 254,000 recommendations (13,000 clicks). Note that we also removed the records for which no features were available (see chapter 4 for more details).

The following experiment shows that surprisingly the trained classifier on the clicks is not outperformed by the multiple regressor trained on the entire dataset (almost 20 times more data). It is even slightly better. We also used this experiment to train a *1 versus all* classifier on P_{1+0} as proposed in section B.2.1 that is by always considering the non-click in the *All* category. It seems to help when the dataset is small but presents little interest afterward.

B.5 Conclusion

In this appendix we presented an equivalence between classification and recommendation. Even though the derived offline evaluation method of contextual bandit algorithms does not offer the same guarantees as the *replay method*, this equivalence is at least interesting intellectually. We also exhibited several potential practical use of the equivalence such as the possibility to take advantage of the decades of work in the mature field that classification is in order to learn things from the data. Moreover as far as replay evaluations are concerned, this new method offers a significant speed up (which is justified theoretically) when we are interested in stationary policy and CTR data.

Appendix C

Historical and current approaches to Collaborative filtering

Abstract of the appendix *This section completes section 1.3.1. It provides a brief history of Collaborative filtering (CF) and describes its early approaches: first user-based CF and then item-based CF. We then provide the basis of the current and main approach to CF: matrix factorization. We finally debate on the preponderance of this new technique in the CF literature and even the recommendation literature.*

Contents

Collaborative filtering (CF) is a method based on a very seductive yet quite simple idea: using the past opinion/behavior of an entire community can help predicting the current tastes of one particular user. The basic principle is that *similar* people in the past will in general remain so in the future. The idea and terminology were introduced by Goldberg *et al.* [25] in a famous article entitled *Using collaborative filtering to weave an information Tapestry* and published in 1992. At the time, CF was already much more than a theory since the authors of the aforementioned article had implemented their ideas in the Xerox PARC Tapestry system, a system able to recommend documents to its users thanks to the community's *annotations* in addition to the content of the documents. The paper considers the general case of annotations that could be complex objects (made of text, numerical evaluations of various criteria) and not only one single rating or vote. This system can be seen as the ancestor of popular recommendation websites such as *del.icio.us* and *Digg.* It is also considered by many to be the first work on recommendation in general. Their approach is fairly simple and is very similar to the one we describe in the upcoming subsection. Besides the obvious scientific interest of this founding paper with regard to recommendation, it is really amazing to see how clearly web 2.0 had already been pictured more than 20 years ago. At the time (in 1992), even the web 1.0 barely existed as the Internet only consisted in less than 1,000,000 interconnected computers.

The WWW had been opened to the public only a year ago and the Internet society was just under creation [26,27]. Furthermore the historical perspective with respect to recommendation can also be quite amusing when reading sentences such as the following.

> *"Filtering on incoming documents is a very computationally intensive task. Imagine a Tapestry system with hundreds of users, each with dozens of filter queries, running on a document stream of tens of documents per minute."* (Goldberg *et al.* [25], 1992)

Before diving into the details of CF, it is important to mention that one of its greatest strengths is the fact that it only requires a *matrix of ratings* to make it work! But what is a *matrix of ratings*? And what are *ratings*? A rating is a special case of what Goldberg *et al.* [25] called an *annotation* of a user on a document, or any type of item in general. It is actually a bounded real value accounting for the opinion of a user on an item. The aforementioned matrix gathers the ratings of all the users on all the items. An example of such a matrix is given in table C.1. Note that this matrix is in general very *sparse* as one given user generally rates a small subset of items.

The most accurate way of collecting the users' opinion is probably by asking for *explicit* ratings on the items. In other words, when a user is confronted with an item, through a recommendation or not, he or she is asked to *rate* this item. This is generally done with an N-star rating system derived from what is called in psychology a Likert scale [149] and that ranges from *Strongly dislike* to *Strongly like*. Five, seven or ten are typical values for N. The number of points necessary in a scale to achieve the most accurate recommendations is discussed by Cosley *et al.* [150] and seems to be domain dependent. The effect of a pseudo continuous rating scale based on a graphical slider was even studied for joke recommendation [30] (Jester, based on the Eigentaste algorithm that we describe in section C.3.2). The main issue with explicit ratings is that they require an additional effort from the users but one may argue that with the emergence of Web 2.0, people are more and more inclined to contribute to the common knowledge for the good of the community.

Nevertheless, for various reasons in many applications explicit ratings may not be available so *implicit* ratings need to be inferred. The most widespread example is to consider the purchase of an item in a web shop as a positive rating but a lot of other things can be taken into account. The system can monitor user behavior: Did he/she click on the recommended items? How long did he/she spend on the page of an item? Did he/she click on "Detailed description" or "See more"? *etc.* Did he or she scroll to the bottom of the page or in other words, did he or she read the entire content provided? It is even possible to go as far as to monitor mouse movements. Implicit ratings have the advantage of being collectible constantly without any effort from the user. However one cannot be sure that the behavior is correctly interpreted by the system. A user who purchased a book might not have liked it. He might even not have bought it for himself. Yet in general this additional noise does not prevent CF patterns to appear. In particular there are some reports of recommender systems (such as for music) being more accurate when using implicit feedback rather than explicit ratings [151].

C.1 User-based CF

C.1.1 The approach

In this section we describe one of the earliest approaches to the recommendation problem: User-based nearest neighbors for Collaborative Filtering. Note that it is obviously not the only way to design a user-based CF algorithm but it is the main idea of two of the founding applications of the field: the Xerox PARC Tapestry system [25] that we already mentioned and the GroupLens system [28] (1994), a Usenet news message recommender. The paper featuring

Table C.1: Matrix of ratings corresponding to a 5-star scale.

	Item 1	Item 2	Item 3	Item 4	Item 5
Alice	5	3	4	4	?
Bob	3	1	2	3	3
Charlie	4	3	4	3	5
Dave	3	3	1	5	4
Eve	1	5	5	2	1

the latter also shows how it is possible to distribute and automatize the whole Collaborative Filtering process. In the 1990s this kind of approach turned out to be so effective that it became the main comparison standard for any new algorithm.

The main idea of user-based nearest neighbors CF is as follows: in order to make a recommendation to a user, we identify its *K nearest neighbors* - that is to say the K users with the most similar preferences - and use their ratings to predict what the current user might like. The implicit assumptions of such a procedure is that similar users will remain similar and that their tastes are stable over time, even if the latter could be easily addressed by weighting the ratings depending on their age.

To be more specific, let us denote by U the set of users, I the set of items and $r_{u,i}$ the rating of user u on item i. To predict the rating of a user $u \in U$ on an item $i \in I$, we select the K users maximizing a similarity metric such as Pearson's correlation coefficient:

$$sim(u,u') = \frac{\sum_{i \in I_{u,u'}} (r_{u,i} - \overline{r_u})(r_{u',i} - \overline{r_{u'}})}{\sqrt{\sum_{i \in I_{u,u'}} (r_{u,i} - \overline{r_u})^2}\sqrt{\sum_{i \in I_{u,u'}} (r_{u',i} - \overline{r_{u'}})^2}},$$

where $\overline{r_u}$ is the average rating of user u and $I_{u,u'}$ the set of items that both u and u' rated. This metric ranges from -1 to 1.

Note that the subtraction of the average rating of the user is meant to ignore the different interpretations of the rating scale made by each user. Indeed some user are typically harsh in their notation and never use the highest level of the scale whereas some others are more generous and give the best star rating to all the movies they like. As an example, consider the data in table C.1. Even though Alice's and Bob's ratings are significantly far apart, they will be considered very similar since there is a clear linear correlation. Indeed, when computing the similarity of Alice with the other users, we find:

$$\begin{aligned} sim(Alice, Bob) &= +0.85 \ , & sim(Alice, Charlie) &= +0.70 \ , \\ sim(Alice, Dave) &= +0.00 \ , & sim(Alice, Eve) &= -0.79 \ . \end{aligned}$$

To predict the unknown rating of a user on an item, we simply take the weighted sum of the ratings of this item made by its nearest neighbors. We then apply the same correction using the average rating of each user in the computation. Thus if we call $N_K(u,i)$ the set of u's K nearest neighbors who rated item i, the computation of the prediction goes as follows:

$$\widehat{r_{u,i}} = \overline{r_u} + \frac{\sum_{u' \ inN_K(u,i)} (r_{u',i} - \overline{r_{u'}})\, sim(u,u')}{\sum_{u' \in N_K(u,i)} sim(u,u')}.$$

So now we can easily compute a prediction of how Alice would rate Item 5 were she asked to. If we take $K = 2$ for the number of considered neighbors, we get:

$$r_{\widehat{Alice,Item5}} = 4 + 0.85 * (3 - 2.4) + 0.70 * (5 - 3.8) = 4.87 \ .$$

C.1.2 Discussion

This kind of approach proved to be very effective when people started using it in the 1990s. In addition to the raw approach that we just described, a lot of details can be added in order to improve prediction accuracy. They are discussed at length in the literature (see Herlocker *et al.* [152] for example). First it is important to note that a lot of similarity/distance metrics where proposed over the years and in many different fields (mean squared difference, cosine similarity...) and can be used to determine the nearest neighbors. Pearson's correlation coefficient, the measure we presented here was shown to outperform all the others for the best-known recommendation domains [152]. Yet some other metrics could turn out to perform better in some special cases and should not be discarded right away when building a recommender system.

Nevertheless this is not the main point one can make about similarity metrics. Even though Pearson's coefficient proved to outperform all the other measures, using its raw version may not be the best choice. For instance, a given item may be universally liked so knowing that two users both like it is much less informative in terms of similarity than two users both disliking this item. Classical metrics such as Pearson's do not take into account that kind of contextual importance of the ratings. At least two distinct techniques were proposed in order to address this problem. The first one is based on the application of a factor that their authors call *inverse user frequency* on the ratings which reduces the importance of universal agreements [153]. The second one gives more weight to items with a high variance of ratings [152]. An other issue can be raised as far as the metrics are concerned: sparsity. The example we used to illustrate the method (table C.1) was idealistic as only the rating we were trying to predict was missing. In practice users only rate a very small subset of the items. This can be problematic when computing similarity since it is natural to wonder if two users who rated similarly very few common items are as similar as the metric would suggest. In the literature this problem is successfully tackled using simple linear weights and/or thresholds [152, 154]. These weights are usually proportional to the number of items rated in common so as to favor neighbors we are sure of. The idea of thresholding consists in completely discarding similar users with too few common ratings when looking for neighbors.

Herlocker *et al.* [152, 154] also discuss the problem of neighborhood selection. There are basically two strategies to select the number of neighbors to consider: (i) taking the K nearest or (ii) taking all the users with at least a fixed similarity value. Both approaches have their strengths and weaknesses that are quite intuitive. For instance with a threshold on the similarity value, we may include too few or too many neighbors for some users to compute good predictions. On the other hand, by thresholding the number of neighbors, we may include "distant" neighbors or discard neighbors with valuable information. The interested reader is referred to Herlocker *et al.* [152] for more details. Anyhow, what comes out of the experiments on the subject is that the optimal number of neighbors is problem-dependent. Yet, independently from the selection method, an average number of neighbors between 20 and 50 generally does a really good job.

C.2 Item-based CF

Item-based collaborative filtering is actually very similar to user-based CF but instead of considering similarities between users, we consider similarities between items to make recommendations. In fact, any approach used to predict ratings with user-based CF (*e.g.* User-based nearest neighbors) could be reversed into an item-based CF approach since the only input is a matrix that can be transposed. Why even bother then? There are actually many reasons and some of them are really intuitive. For example in some domains of application, it might just work better, the same way as some metrics are more or less effective depending on the domain. Another intuitive reason is that it is much easier to justify the recommendation to the user, giving it more credit and thus more chances to be followed. Indeed it is easy to justify to someone that he or she is recommended something because it is similar in a given way to one he or she bought or likes. Amazon, one of the firsts to report of Item-based CF in the literature [29], typically does so by putting the recommendations below phrases such as "users who bought this also bought...". The idea is to consider that two items that were bought at the same time or simply by the same person are somehow similar.

Before talking about the main reason why Item-based CF was introduced, let us mention that in the literature, the standard measure for similarity evaluation (to select neighbors for instance) is not Pearson's correlation coefficient. This new standard measure, the cosine similarity (see *Recommendation: an introduction* [24], section 2.2.1), simply measures the cosine of the angle θ between two items (which are in fact rating vectors). Therefore the similarity between two items can be computed as follows:

$$sim(i, i') = cos(\theta) = \frac{i.i'}{||i||\ ||i'||},$$

where the dot is the inner product between two vectors and $||v||$ the Euclidean norm of a vector v.

Notice that this measure does not take into account the difference of interpretation of the rating scale by different users. This is the reason why the *adjusted cosine similarity* was introduced. This new measure is the same as the aforementioned one but subtracts to each rating the average rating of the user who issued it. One may then notice that surprisingly the adjusted cosine similarity becomes very similar to Pearson's correlation coefficient and even wonder if they are not two different names for the same thing. Indeed, the adjusted cosine similarity between two items i and i' is given by:

$$sim(i, i') = \frac{\sum_{u \in U_{i,i'}} (r_{u,i} - \overline{r_u})(r_{u,i'} - \overline{r_u})}{\sqrt{\sum_{u \in U_{i,i'}} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{u \in U_{i,i'}} (r_{u,i'} - \overline{r_u})^2}},$$

where $U_{i,i'}$ is the set of users who rated both item i and i'.

The only difference with using Pearson's coefficient is that to measure the similarity between two items, we would subtract the item's average rating and not the user's which would make much less sense. Apart from that, the two measures are exactly the same which reinforces the idea of usefulness or superiority of that kind of measure compared to others (squared difference, Spearman's rank *etc.*) in collaborative filtering.

Finally let us discuss the main advantage of item-based CF compared user-based CF which is not as intuitive as the others. In the early 1990s, user-based CF had given very interesting results and was very popular. Nevertheless, people were struggling to scale it to bigger applications such as online commerce. Indeed, at each recommendation, the system would have to scan through tens of millions of users to check for similarity which is unfeasible in real time. Yet one may argue that the same goes for item-based recommendation. Indeed if the system has to scan through a typically smaller number of items (still probably hundreds of thousands), similarities are longer to compute than between users since the vectors are larger making both approaches equally greedy. Nevertheless researchers at Amazon came up with the idea of pre-computing an item-to-item similarity matrix so as to avoid computing the costly similarities at run time [29]. Using the same idea should enable to scale up user-based CF as well. Yet it is much more effective with item similarities and so for two main reasons that are somewhat related:

- Pre-computing a user-to-user similarity matrix it is much more space-consuming and brings less performance improvement. Indeed the expensive part with user-based CF is to scan through tens of millions of users to find neighbors. Similarities are fast to compute anyway given the small average number of common rated items. On the contrary item-to-item similarities are heavy to compute given the tremendous amount of users in the database, hence a huge gain when they are pre-computed.

- In addition, people noticed that item similarities are much less sensitive to the arrival of new ratings and thus much more interesting to pre-compute. On the contrary, user similarities tend to vary a lot when new ratings are entered into the system. This is very likely due to that fact that most users typically have very few ratings. Consequently any rating addition is significant with respect to their similarity with others.

Moreover, further "pre-processing savings" are possible. Without going into the details, a possible option is to discard the item-to-item relationships with too few common ratings. However this can lead to more cases in which no CF recommendation is available [155]. Another interesting option would be to only search for similarities in a subset sampled at random from the original matrix of similarities (or matrix of ratings depending on the approach used).

C.3 From memory-based to model-based recommender systems

C.3.1 Model-based VS Memory-based

A memory-based algorithm is an algorithm which only relies on the raw data stored inside the memory to make its predictions. The first approach we presented (user-based nearest neighbors) is such an algorithm. On the contrary the item-based approach we presented computes offline an item-to-item similarity matrix. This pre-computed matrix is much smaller than the whole dataset and can be considered to be a model of the data, hence its categorization as a model-based recommender system. *In general*, if time is not a constraint, memory-based recommender systems produce more accurate predictions than model-based systems for the obvious reason that they use more information (this is in fact true for data-driven systems in general). However with the growing amount of available data, such coarse approaches are no longer tractable in real time. It would however be possible to keep using them with sub-sampled versions of the datasets but this proves much less efficient than reducing the complexity by smarter means such as pre-processing the data into a model.

The first step toward this introduction of models into recommender systems was the item-based collaborative filtering that is in use for example on *Amazon.com* and that we just described. Note that model-based does not necessarily means preprocessing. A model can be at the same an efficient way to represent the data and something that can updated when new events occur such as a new rating issued by a user. However this needs an additional effort of system design. Indeed if for memory-based approaches, appending a new record to the existing dataset is enough to update the system, a model-based recommender system requires a more sophisticated update method. An algorithm using a model that has to be recomputed from scratch to take into account new events is called an *offline algorithm.* On the contrary an *online algorithm* uses a model that can be updated without a complete recomputation. The *LAP* algorithm that we present in the next chapter (see section 2.6) is a good example of a model-based algorithm which is updated online.

C.3.2 Matrix factorization

The item-to-item similarity matrix was one of the first step toward model-based recommendation. However the most famous and revolutionary one is most certainly the application of matrix factorization to pure collaborative filtering. Before going into the details, it is interesting to note that this approach was brought to light by a worldwide competition: the *Netflix Prize* that we already mentioned without further explaining. The Netflix Prize was launched in 2006 by the eponymous US movie rental company. They offered a one million dollar prize to anyone who would improve the rating prediction accuracy of their own recommendation algorithm by at least 10%. The challenge was completed in 2009 and the main component of the winner team as well as many of the other top teams was matrix factorization and, in particular, *Singular Value Decomposition* (SVD).

Even though it was made popular by the Netflix Prize, matrix factorization-based collaborative filtering was introduced a few years earlier (in 2000 to the best of our knowledge) by Sarwar *et al.* [32]. In 2001, matrix factorization was also reported to be used in Eigentaste, a generic recommender system developed at the University of Berkeley by Goldberg *et al.* [30]. Eigentaste is in use in Jester, the famous joke recommendation application. Before that, matrix factorization was used successfully in the late 1980s in information retrieval to query text documents for instance. See Deerwester *et al.* [156] for more details. Remark that the system described by Deerwester *et al.* is one of the precursor of what we now refer to as *latent semantic indexing*.

The whole idea of this method is to infer latent features or factors from the rating matrix, characterizing both users and items. These factors can then be used to fill in the blanks in the matrix by allowing to reconstruct it entirely and thus make predictions. Accordingly, for each user and each item we want to infer a feature vector of size f and use the inner product of these vectors as an approximation for the ratings. More formally a rating of user u on item i will be predicted as follows:

$$\widehat{r_{u,p}} = q_u.q'_i,$$

where q_u and q'_i are the feature vectors of u and i. Therefore note that if it is possible to reconstruct the entire matrix of ratings, it is not necessary when making a particular prediction. A simple inner product is required.

Let us consider the well-known movie example for a minute. These latent factors can be understood as characterizing a movie (level of action, romance, terror or even the quality of the soundtrack *etc.*) and the appeal of a user to movies having these characteristics. Figure C.1 is an illustration of the principle in two dimensions. Nevertheless some factors learned by performing SVD can sometimes be much trickier to interpret. Note that this process drastically reduces the dimensionality. Indeed if we denote by M the number of users and N the number of items, the $M \times N$ matrix of ratings is mapped to $(M + N) \times f$ factors with f being typically between 20 and 100 [31]. It also reduces the time complexity at run time since a prediction only requires an inner product of two vectors of size f.

The Singular Value Decomposition theorem [157] states that any given $M \times N$ matrix R can be decomposed into three matrices such that:

$$R = U\Sigma V^T,$$

where U is a $M \times M$ matrix, V a $N \times N$ matrix and Σ a $M \times N$ positive and diagonal matrix whose coefficients are the singular values of R.

According to the Eckart-Young theorem [158], what is fantastic about this decomposition is that for any given value of f it provides us with the optimal solution to:

$$\min_{q_1,q_2...,q_M,q'_1,q'_2...,q'_N \in \mathbb{R}^f} \sum_{u \in U} \sum_{i \in I} \left(r_{u,i} - q_u.q'_i\right)^2,$$

where for a given user u (resp. item i), q_u (resp. q'_i) is the corresponding feature vector. In other words, the SVD of the matrix R provides the feature vectors $q_1, ..., q_M, q'_1, ..., q'_N$ (for any length f) such that the sum of the squared error on the rating predictions (made via inner product of the corresponding vectors) is minimized. More formally these feature vectors are

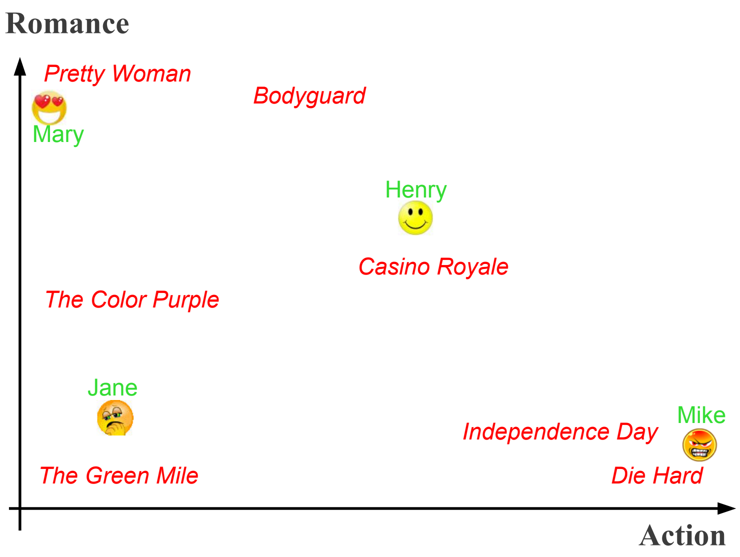


Figure C.1: A simplified illustration of the latent factor approach, which characterizes both users and items (here movies) using two axes: the level of romance and the level of action.

the one minimizing the Frobenius norm[1] of the difference between the original matrix and the prediction matrix.

To get the solution we first reorder the singular values in decreasing order (which is a common convention). Then we just have to take the first f singular values given in Σ (we get Σ_f a $f \times f$ diagonal matrix) and the first f column vectors of U and V^T. This is what the Eckart-Young theorem states. An intuitive explanation (which will make more sense when talking about stochastic gradient descent in what follows) is that the greater a singular value is, the better its corresponding latent factors describe the matrix R. Finally in order to get a solution of the exact form we expected it is necessary to include the singular values into q or q'. If we choose the former, for a given user u and a given item i, q_u is equal to Σ_f multiplied by the u^{th} line of U and q'_p is equal to the p^{th} line of V^T.

In theory and for small matrices, the decomposition can be computed with a linear algebra library. Nevertheless in practice it is a bad idea and so for two crucial reasons:

- The first reason is only pragmatic: the best way to compute the exact SVD has a time complexity of $O\left(M^2N + N^3\right)$ which is intractable for datasets encountered in real applications with millions of users and thousands of items.

- The second reason is more interesting. The linear algebra approach will consider R as a complete matrix with zeros (or anything we want) instead of the gaps. These zeros will be included in the minimization process and thus the SVD-based predictor will tend to predict zeros for all the unknown ratings which is hardly what we expect. Indeed the actual minimization problem we are interested in solving is the following:

$$\min_{q_1,q_2\ldots,q_M,q'_1,q'_2\ldots,q'_N \in \mathbb{R}^f} \sum_{\{u\in U, i\in I | r_{u,i}\in R\}} \left(r_{u,i} - q_u.q'_i\right)^2,$$

  where R is actually an incomplete matrix of ratings. This is not what the SVD solves exactly.

Luckily there exists a very nice alternative to the exact method: the *Stochastic Gradient Descent* (SGD) which was popularized by Simon Funk in a blog post [159] and can be proved

[1]The *Forbenius norm* of a matrix is the square root of the sum of is squared coefficient.

to converge almost surely toward the optimal solution [160, 161]. The method is quite simple and consists in trying to find each singular value (along with its corresponding left-singular and right-singular vectors), one at a time, from the greatest to the f^{th} greatest. More precisely, since we only care about a decomposition and not about the singular values, we simply look for a decomposition as follows:

$$R = U.V^T.$$

To find the j^{th} singular value and the two corresponding vectors, we consider the following sub-problem of minimization:

$$\min_{q_1[j],...q_M[j],q'_1[j],...q'_N[j]\in\mathbb{R}} \sum_{\{u\in U, i\in I | r_{u,i}\in R\}} \left(r_{u,i} - q_u[1..(j-1)].q'_i[1..(j-1)] - q_u[j]q'_i[j]\right)^2,$$

where $v[k]$ denotes the k^{th} element of vector v and $v[a..b]$ the vector composed of all the elements of the vector v between index a and index b included. Notice that here for $j \in \{1..f\}$, $(q_1[j], ...q_M[j])$ (resp. $(q'_1[j], ...q'_N[j])$) is the j^{th} left-singular vector (resp. right-singular vector) of R up to a constant. These f vectors are the column vectors of U (resp. V). The singular values (Σ) that we have no interest in finding are embedded within U and V. Note that the latent feature vectors (or factors) q_i and q'_j are not singular vectors of R for they are the row vectors of U and V.

In the equation of the minimization sub-problem, the factors corresponding to the $j-1$ first singular values ($q_u[1..(j-1)]$ and $q'_i[1..(j-1)]$) are considered to be known since they were computed in previous iterations. Therefore this minimization problem is only about finding the j^{th} element of all the feature vectors ($q_u[j]$ and $q'_i[j]$). To solve it, we perform a gradient descent, that is we:

- compute the gradients of the objective function with respect to the latent factors of the items on one hand and with respect to the latent factors of the users on the other hand,
- follow the gradient to update all the factors at the same time,
- iterate until convergence.

An iteration could be done in one big calculation but to avoid having to put all the data in the memory at once, we fake an online process.

To be more specific, for the j^{th} singular value we scan through the dataset rating by rating. For each rating $r_{u,i}$ we first compute the current error of prediction (gradient):

$$\varepsilon \leftarrow r_{u,i} - q_u[1..j].q'_i[1..j] \ .$$

At this point we are looking for $q_u[j]$ and $q'_i[j]$ so for the first iteration, they need to be initialized. Nothing fancy is required. They can simply be set to zero. Then we perform simultaneously the following update equations (we follow the gradient):

$$\begin{aligned} q_u[j] &\leftarrow q_u[j] + \gamma\varepsilon q'_i[j] \ , \\ q'_i[j] &\leftarrow q'_i[j] + \gamma\varepsilon q_u[j] \ . \end{aligned}$$

The process is repeated until convergence of the latent factors which typically requires the entire dataset to be scanned several times. Note that γ is the learning rate, a tunable parameter that determines the amplitude of the updates. A value too small for γ significantly slows down the process whereas a value too big may cause it to simply diverge.

This method has the advantage of being relatively fast compared to the classical linear algebra techniques. Moreover it only takes into account the ratings we actually have which, besides allowing the computation to be tractable, turns out to be a critical detail. Indeed, the exact algebraic solutions needs a full matrix and filling the blanks so that it works as expected is far from being trivial. Note that for the sake of simplicity we purposefully omitted numerous details. For the algorithm to achieve decent performance, one may want to think about regularization, normalization, priors and much more details [31]. See also Simon Funk's blog post [159] for technical details such as implementation guidelines.

This section was just a shallow overview of the basics. It is also interesting to note that *Stochastic Gradient Descent*, although it is the most famous, is not the only way to learn the decomposition. Indeed another method called *Alternating Least Squares* (ALS) exists. The idea is rather simple. The optimization problem that we deal with for each factor j is not convex. Nevertheless, if we fix either $q[j]$ or $q'[j]$, then minimizing the squared error becomes a quadratic problem. The idea behind ALS is simply to fix $q'[j]$ and solve the least-squares optimization problem for $q[j]$ only. When this is done, we fix $q[j]$ and solve for $q'[j]$. The first feature vector is fixed randomly and the next ones are fixed to their current value. We iterate the process until convergence which can be ensured analytically [162]. According to Koren *et al.* [31], this technique of alternating least squares requires slightly more computational resources. It makes sense since by performing full optimization steps on only a part of the parameters, we might not follow the gradient as strictly as a stochastic gradient descent. Anyway its popularity comes from a major practical advantage. With stochastic gradient descent, an online learning process is simulated. Thus every single step depends on the previous one making parallelization a tricky task. This is not the case with ALS. For a given factor j and for a given least squares iteration, all the $q_u[j]$ (or $q'_i[j]$) are computed independently from one another. This gives rise to massive possibilities of parallelization and thus of speed up [163]. ALS possesses another strength compared to SGD. We mentioned that SGD was meant to be used on sparse matrices of ratings. Yet in some cases the matrix is dense. Because it iterates over all ratings many times, SGD is very inefficient in that case whereas ALS handles the situation perfectly well [164].

C.3.3 Remarks on the winning approaches to the Netflix Prize

The top methods of the Netflix Prize all used SVD but not only. They were actually all some kind of ensemble predictors gathering the results of tens or even hundreds of predicting algorithms. It is funny to note that if matrix factorization-based recommender systems were made highly popular by this contest, making it the top research topic in the recommendation field, another of the predictors achieved similar results to SVD's. This method called the Restricted Boltzmann machines (RBM) is based on approximate graphical models that are also looking for latent factors [165]. If SVD alone improved the prediction accuracy of Netflix's recommender system by 6%, RBM were also highly competitive and accounted on their own for a 5.5% lift. The most interesting is that combined, these two approaches were responsible for a 7% amelioration [87]. One may try to guess why Matrix factorization turned out to be a much more popular research topic than RBM. A reason may be that it bridges directly pure linear algebra with an existing application and that the possible approaches to matrix factorization are numerous. Consequently many theoretician have been able ever since to work on a popular practical application by only focusing on a problem of pure linear algebra. Another reason may be that the resulting model for the prediction of ratings (the latent factors) is quite interpretable. It is also *a part* of recommendation in general that is very easy to evaluate without much bias and for which many benchmarks are available.

There is another funny fact about the Netflix Prize that is worth mentioning. The company actually never implemented the winning method according to what they report on their blog [87]. In fact they did put some effort to use SVD and RBM models in practice but they considered that the rest of the package "*did not justify the engineering effort needed to bring them into a production environment*". Furthermore it is important to note that improving the prediction quality does not necessarily ensures better recommendations. Netflix also gave up on the full implementation for another reason that was much more unexpected: their business had shifted in the meantime. Indeed, in 2006, Netflix was a movie rental company. In 2010 it had become more of an online streaming company and as it turned out recommending online streaming content and DVDs are two different things.

The last remark on the Netflix prize is something that is not commented very often in machine learning. Most of the time people talk about relative improvements and forget about absolute results. For instance in the Netflix prize the winners achieved a 10% lift on the prediction accuracy which is quite significant. Nevertheless it is interesting to have in mind that it made the root mean square error (RMSE) shrink from 0.9525 to 0.8525. This RMSE is of the order of 1 on a 1 to 5 Likert scale of range 4. This means that the average error on a prediction is between a quarter and a fifth of the total scale which is huge. Yet one may be quite confident in the fact that considering the tremendous endeavor that led to these results, it would be hard to go much further. This exhibits the limits of pure collaborative filtering, that is a method based on only user and item IDs and the corresponding matrix of ratings. It demonstrates the need to use additional sources of information such as item descriptions, user profiles, social networks *etc* in order to make better predictions.

C.3.4 Putting the importance of matrix factorization into perspective

We mentioned in the previous remark that matrix factorization has taken a tremendous importance in the recommendation field. We also tried to give a few explanations. We really want to stress the importance it has taken in the last years in research areas such as Machine Learning or Data Mining. We have no actual figures to support that claim but a quick glance at the proceedings of *RecSys 2013*, the ACM Conference on Recommender Systems [166] or at the recommendation track of *the International Conference in Machine Learning (ICML) 2012* [167] is already quite convincing. Out of 8 papers in the latter, one is about active learning, one is a Bayesian approach to neighborhood methods in CF and 6 are about matrix factorization although one of those six is more original as it considers user×item×query tensors of ratings. Yet the basic idea to compute a decomposition remains unchanged. The 5 remaining articles are about pure matrix factorization techniques. Nonetheless it is very important to have in mind that matrix factorization is just a small step along the recommendation process. If we only consider the Machine Learning part and skip things such as the user interface, one can think of:

- The data we need to acquire in order to improve the system (active learning).
- The cold start problem (new system, new users, new items).
- The offline evaluation process.
- The prediction of the ratings given the reconstructed matrix. Should we simply use the ratings predicted by the SVD? It is not clear. Indeed some research exhibited that additional processing such as nearest neighbors in the latent space can be performed to do better than just considering the ratings given by the reconstructed matrix [31].
- Last but not least, once we have rating predictions, what shall we recommend? Do we always recommend the item corresponding to the best predicted rating? Should not we try to take into account the current session of the user as he/she may be looking for something specific? What if several items have to be recommended? Do we recommend the K best items even though some of them may be very similar? Many more questions are very interesting and not answered by matrix factorization or any other rating prediction

technique. Indeed predicting ratings does not tell us how to recommend items so as to maximize relevant metrics such as the click through rate, the sale volume, user satisfaction or anything else.

Note that this list of steps within the recommendation process is far from being exhaustive. Anyway it is enough to emphasize that matrix factorization is only one step among many others to design a good recommender system. Moreover, matrix factorization is a part of a pure collaborative filtering approach. Therefore considering our previous remark on the limits of pure CF reached in the Netflix Prize, it may not be the most promising line of research as far as improving recommendation is concerned. Nowadays it is unfortunately clearly the most explored area in recommendation.

C.4 Discussion

Before discussing the strengths and weaknesses of collaborative filtering, let us first say that this section was just an overview of the first techniques (nearest neighbors) and the most popular ones (matrix factorization) as well as the context in which they were introduced and popularized. Note that some other CF techniques exist such as probabilistic approaches. The basic idea is to infer distributions conditioned by the ratings of items and to use them to make predictions (for example if an user rated this item, he or she likes that other item with some probability). RBM that we already mentioned could be put in this category as graphical models are an advanced and systematic way of computing conditional probabilities.

Now let us discuss the numerous strengths of collaborative filtering. First it is really interesting to note that the only input necessary for CF is a matrix of ratings which can be explicit or implicit (CTR, product purchase, browsing behavior...) which is quite universal. It means that even though all the recommendation domains are different, all the research advances in the field of collaborative filtering can be applied almost in every system. It also means that CF requires very little effort from the user (no need to fill in a profile) and from the owner of the website (no need to manually label the items). Second CF is very intuitive which leads to interpretable results. For item-based nearest neighbors CF, we have seen that it is even possible to explain very intuitively to the users how their recommendations are generated. It was not tackled in this review but a lot of research has been done on explaining CF-based recommendations in general to users [33, 34]. Such explanations were exhibited to increase the trust of the users in the system and thus its overall performance [33]. Another strength of only using a matrix as input is that it is possible to take advantage of all the previous work in linear algebra/information retrieval to improve the quality of the predictions. Last and not least, it was proved empirically to work very well in various domains in spite of the extreme simplicity and generality of the input.

Nevertheless, the simplicity of collaborative filtering can also be considered as a weakness. Indeed, in its "pure" version, CF is unable to use the side information that may be available (product characteristics provided by the manufacturer, geographic and other demographic information, time of the day/month/year...) which could prove useful. This is not really a problem if the system exhibits satisfactory results. Yet we saw that there seems to be an invisible wall of performance that cannot be crossed with such a generic approach. Nonetheless the main issue is elsewhere. Collaborative filtering needs a substantial amount of data in order to work properly. This raises a few issues:

- For instance it is very difficult to use collaborative filtering in small applications as the main strength of CF is its ability to mine and "summarize" a lot of data in order to make personalized recommendations. Furthermore, CF would be completely useless to recommend cars, houses or other products that are bought once in a decade or more.

- This first issue was predictable and as it is not what CF was introduced for, it is not a big deal. Yet another issue is much more concerning: what kinds of recommendations can we do when a new web application with new items and unknown users is put online? Because of its need of existing ratings, raw collaborative filtering is completely useless with that regard.

- One may argue that an application and its recommender system are normally only launched once in a lifetime and thus that this is not a big issue. Nevertheless many researchers disagree and the amount of work on this problem is proof enough. See for instance a founding work by Schein *et al.* (2002) [168] or a more recent one by Lam *et al.* [169]. Both approaches use side information on the items and are the subject of the upcoming section. Furthermore even when the system has been online for quite some time and is successfully working, one particular issue never completely leaves. Indeed, collaborative filtering is a completely useless tool to predict the tastes of a new user since no similarity can be computed. Similarly a new item would also be a mystery to a pure CF recommender system as users and items have a fairly symmetric role. In some applications, things get even worse. In news recommendation application for instance, one typically has to deal with a pool of items that come and go. Therefore almost all the items are always new hence the need for a totally different approach in this case.

These two last points, and especially the former are known as the *cold start problem.* In the problem that is referred to as dynamic recommendation in this thesis (*e.g.* news recommendation), we can view the system as constantly cold or vaguely lukewarm at best. Indeed the recommender system constantly faces a pool of very recent items. In section 1.3.2 we briefly talk about the other principal approach to recommendation (which is not the only one though) and which is the main answer in the literature to some of the issues that we raised in this discussion. This thesis discusses another approach based on online learning and formalized as a contextual bandit problem.

Appendix D

Metrics for the offline evaluation of recommender systems

Abstract of the appendix *This appendix is the complement to the content provided in section 1.5.4. It deals with the strength and weaknesses of rating prediction metrics and mainly RMSE in the context of recommendation evaluation. It then proposes alternative metrics to use in complement or in replacement. This review is mostly based on a work by Herlocker et al. [8] published in 2004 and which is the authority in the area.*

Contents

D.1 Prediction accuracy metrics

In the main body of this thesis document one offline evaluation metric for recommender systems: the root mean square error (RMSE) on predicted ratings. This historical metric is still the most commonly used along with mean absolute error (MAE) (see table 7.3 in *Recommendation: an introduction* [24]) also applied to prediction accuracy. Although it is accepted by the community to be meaningful when evaluating a recommender system, it has weaknesses. A paper by Herlocker *et al.*, published in 2004 [8], is the authority in the field of the evaluation of CF-based recommender system. This article points out the strengths and weaknesses of recommendation evaluation through rating prediction accuracy metrics. It also provides many reasons why other metrics should be used in complement or sometimes even in replacement. Finally other metrics are proposed to measure different qualities of a recommender system. Another very interesting survey about recommender system evaluation was published more recently (2009) by Gunawardana and Shani [79].

The strengths of MAE like metrics are fairly obvious. The most obvious one is that all the ratings are taken into account, whatever their value, so this metric would probably be a descent one for any kind of recommender system. Then MAE and RMSE are simple which makes them easy to compute, easy to understand and easy to interpret which are non negligible considerations. Finally a second consequence of MAE's simplicity is that it has well studied statistical properties which, for instance, make the tests of significance straightforward. However the simplicity of these metrics brings along a few weaknesses. If they may just by what is needed for the MovieLens recommender system that displays the raw predictions of the ratings to its users, it is generally not the best choice. Indeed most of the recommender systems' main task is to just find *good items*. For such systems, we only want good items to be ranked high. The accuracy of prediction for bad items is of little importance as long as they are not recommended. Yet with MAE or RMSE, all the ratings have the same weight which is problematic. The fact that it is not a good indicator can even be proved theoretically [80]. Furthermore, a recommender system tends to recommend several items to a given user. It can be done sequentially during a session or at the same time in a recommendation list. In this context even the prediction accuracy of a good item does not matter very much as long as it is classified as good. *e.g.* On a 5 stars-Likert scale [149] if we consider the separation between good and bad to be 3, predicting 4 stars instead of 5 or 1 star instead of 2 is not a big deal. However predicting 3.5 instead of 2.5 or the opposite is an error that should weigh much more in an evaluation metric. The interested reader is also referred to McNee *et al.* [81] for a severe criticism of the systematic use of accuracy metrics.

A much more thorough discussion of the weaknesses in different contexts of prediction accuracy metrics is also made by Herlocker *et al.* [8]. In addition to these weaknesses, another point is raised by the authors and presses toward the use of other metrics at least as a complement. Back in 2004 when the article was published, they had noticed a trend in the newest algorithms. Most of them were yielding a MAE of approximately 0.73 on popular movie rating datasets. Although their authors empirically demonstrated the superiority of their new approaches compared to the older ones, Herlocker *et al.* [8] found that they all reached roughly the same MAE when all carefully tuned. They formulate the hypothesis that this means the community is facing an invisible barrier that cannot be crossed using pure CF, probably because of a lack of information or natural variability. This hypothesis is supported by the fact that it was shown that users are inconsistent when asked to rate a movie at different times [170]. Consequently it cannot be hoped to design an algorithm with a better precision than the variance of a single user's ratings on the very same item. Note that we did not say that all the algorithms are strictly equivalent. Sometimes statistically significant differences can be measured. Yet as statistically significant as they may be, they usually are very small quantitatively [8]. Would a user really be sensitive to a 0.01 change in prediction accuracy of the ratings on a scale of range 4 or 5? If the hypothesis is true and all the recent algorithms are just equally good and almost optimal at what they do that is predicting ratings, the idea to compare them with different criteria becomes natural.

D.2 Other metrics

Let us briefly mention a few metrics and their properties that are listed and commented by Herlocker *et al.* [8]. Note that those metrics require the same methodologies as the evaluation of rating predictions with the RMSE or the MAE, that is holding back some part of the data for testing. As such they present the overfitting issues that are inherent to this type of method.

- **Tweaked prediction accuracy:** The first basic idea is simply to use the intuition that users want good items and are really annoyed by really bad items. The resulting metric is the MAE on only the extreme ratings and exhibits nice results in complement with other metrics [82]. Nevertheless this metric should not be used alone as anything can be predicted without penalization for intermediate ratings. In particular the corresponding intermediate items can be ranked very high and recommended instead of really good items.

- **Classification accuracy:** This kind of metric targets the classical recommender systems that aim at finding the best few items for a given user. A binary classification task which consists in separating good items from bad items is thus defined. The separator is generally a simple threshold as the aforementioned example with the 5 stars Likert scale. With such a metric, the recommender system is actually rewarded for finding good items and penalized if it recommends bad ones. Consequently this reflects the actual user experience better than MAE. Several classical classification metrics can be used such as precision, recall, F1 score which is a combination or the first two ones *etc.* They each have various purposes, strengths and shortcomings that are detailed by Herlocker *et al.* [8].

- **Ranking accuracy:** Rank scores are basically finer classification metrics. They differentiate items successfully classified as good items by taking their position into account. This type of metric is particularly well suited for recommender systems that offer an ordered list of recommendations. Again, several metrics are available in the literature and most of them are discussed by Herlocker *et al.* [8].

- **More metrics:** Other metrics have been derived for various profiling purposes. For example to study the behavior of the system with new users, one can compute user coverage which measures the number of users to whom a non-empty recommendation list can be provided. We can also mention list diversity metrics that reflect the fact that the recommended items are not too similar (see Ziegler *et al.* [171] for more details). Indeed it may be pointless to recommend 5 very similar items. If the user is not interested by the first one, he or she will very likely not be interested by the other ones as well (although this statement might be domain dependent). One can also be interested in data efficiency, that is the number of ratings (or more generally past interaction within a dataset) needed to achieve decent predictions. This consideration actually will turn out to be a key point of this thesis work.

D.3 Other criteria of evaluation

There are additional criteria to evaluate a recommender system that cannot be measured and compared with a metric. The most popular one is probably its ability to explain its recommendations to the users. Different kinds of explanations can be given depending on the algorithm and no offline metric can attest of their efficiency. However it was exhibited that they drastically improve the confidence of the users in the recommendations as well as the tendency of these recommendations to be followed [33]. It was also found that the best-performing explanations are based on similarities between users. Content similarities and simple indications on past performance of the recommender system are other well performing explanations. On the contrary, indications about ratings from experts such as movie critics does not seem to increase acceptance. Finally some people argue that the only criterion that matters is user satisfaction. Note however that even if it is true, the metrics that we described here were shown to be significant in that regard. In general user satisfaction can be measured using the number of non returned purchases. In non commercial applications, users can be asked about their satisfaction via questionnaires.

Appendix E

Details on our approach to win the *Exp. vs. Exp.* challenge 2 and a generalization to a highly flexible and computationally-efficient contextual bandit algorithm

Abstract of the appendix *Here we first propose a contextual bandit algorithm: GAP which is much faster than LinUCB with linear features (linear instead of cubic complexity) and which - additionally - is capable of handling discrete and hybrid features. This algorithm is a generalization of AdPredictor [6], that we adapted and tweaked to win the Exploration vs. Exploitation challenge 2 (ICML 2011). In this appendix we also provide more details than in chapter 2 on how this algorithm can be used to handle dynamic environments and on its empirical properties.*

Contents

E.1 Generalized Ad Predictor (GAP): a highly flexible and computationally-efficient contextual bandit algorithm

This section provides more details on GAP, an algorithm briefly presented in section 2.6.2.

Before describing this algorithm, let us recall that to win the *E vs. E* challenge 2, we adapted Ad Predictor [6] (alg. 7 page 54), an algorithm that takes as input discrete features. We then proposed Linear Ad Predictor (LAP: alg. 8 page 58), a variant that uses linear, continuous features the same way as LinUCB does in order to compare both approaches. As a result, they exhibit similar performance although LAP has the advantage of being incredibly faster (by a factor 10 with 15 features and a factor 100 with 50 features). More specifically, LAP has a linear time complexity in the number of features F whereas LinUCB's is cubic.

E.1.1 Heterogeneous features

Linear Ad Predictor has the advantage of being able to use continuous features without preprocessing and thus to be compared with state-of-the-art methods. However it suffers from several shortcomings which paradoxically could be considered as *Ad predictor*'s strengths:

- It cannot deal smartly with missing values. *Ad predictor* does it by considering a missing value as a distinct discrete value of the feature.
- A lot of real life features are categorical (country, gender, socio-professional group etc.) and *LAP* cannot deal with them. Yet in practice people usually compute numerical features using these categories (binary features, distance to someplace etc.) but the use of *Ad predictor* allows us to skip this annoying step.
- *LAP* considers that all features bring the same amount of uncertainty on the prediction. On the contrary, *Ad predictor* was able to determine for which feature it was confident on its contribution to the click probability and for which it was not in order to improve both its exploration strategy and its updates.

Yet linear features could be useful in some cases. For instance retirement plans could be positively correlated with age whereas the interest in an ad for a supermarket could be negatively correlated with the distance to its location. They are also simpler to use as they do not need to be discretized. This is the reason why at some point during the challenge we tried to use the continuous features as linear features with one unique weight as in *LAP* while keeping the categorical features intact. Indeed nothing prevents us from having both features with several weights and features with one linear weight as all the features are updated separately and also contributes separately to the prediction score. To make things clear, here is how the score is now computed:

$$Score(d) = \mathcal{N}\left(\theta_c(d)^T w \ + \sum_{a \in active\&discrete(d)} m_a \ , \ \sum_{a \in active(d)} \sigma_a^2\right)$$

with $\theta_c(d)$ the vector of continuous features of d that we consider to be linear, w the linear weights and $active\&discrete(d)$ the list of the discrete active contributors.

All the attempts we made trying to use that kind of features in the challenge consistently led to performance drops. As we mentioned it before, we had identified groups of similar continuous features so we tried to only make some of the groups linear, we tried to consider some features as both linear and discrete (resulting in two features as far as the algorithm is concerned) but everything failed. We also try to pre-transform the features with various classical functions such as $log(x)$, $exp(x)$, x^2 without any more success. This can be understood by having an other look at the density plots of the features (see fig 2.3). The values can be grouped into clusters that are mostly discernible to the naked eye.

E.1.2 Hybrid features

We have seen that it was possible to use two different kinds of features: discrete ones and linear ones. It turns out that it is possible to use features that are a little bit of both. Of course it is then possible to use a mosaic of the three kinds of features. We call this more general algorithm *Generalized Ad Predictor* (*GAP*). The idea of hybrid features is quite simple.

Sometimes, continuous features are not linear. The classical way to handle a non-linear function of a feature f is to compute additional features such as f^2, f^3, $\frac{1}{f}$, e^f *etc.*and to treat them as linear features. Obviously such features can be added to GAP. The downside of this approach is that it requires to know in advance the kind of function we want to learn, or to generate a broad set of potentially useless features that will slow down or even impede the learning process. Yet there exists another method - easy to plug into GAP - that does not require any assumption : instead of considering that a feature is either discrete or linear, we consider that some features can be linear "by part", that is independent linear features on discrete parts of the original one (hence the term hybrid). a feature is *discretized* (say uniformly) and each part is considered as a *linear* feature independent from the others. This way any function can be approximated. Choosing the granularity of the discretization is a trade-off between accuracy and learning speed. Indeed, only one part is active and thus updated per display. Technically, handling this with *Ad predictor* is quite simple: for a given feature, we consider that it has several linear weights that can only be active one at a time. The three kinds of features can be distinguished visually on figure E.1.

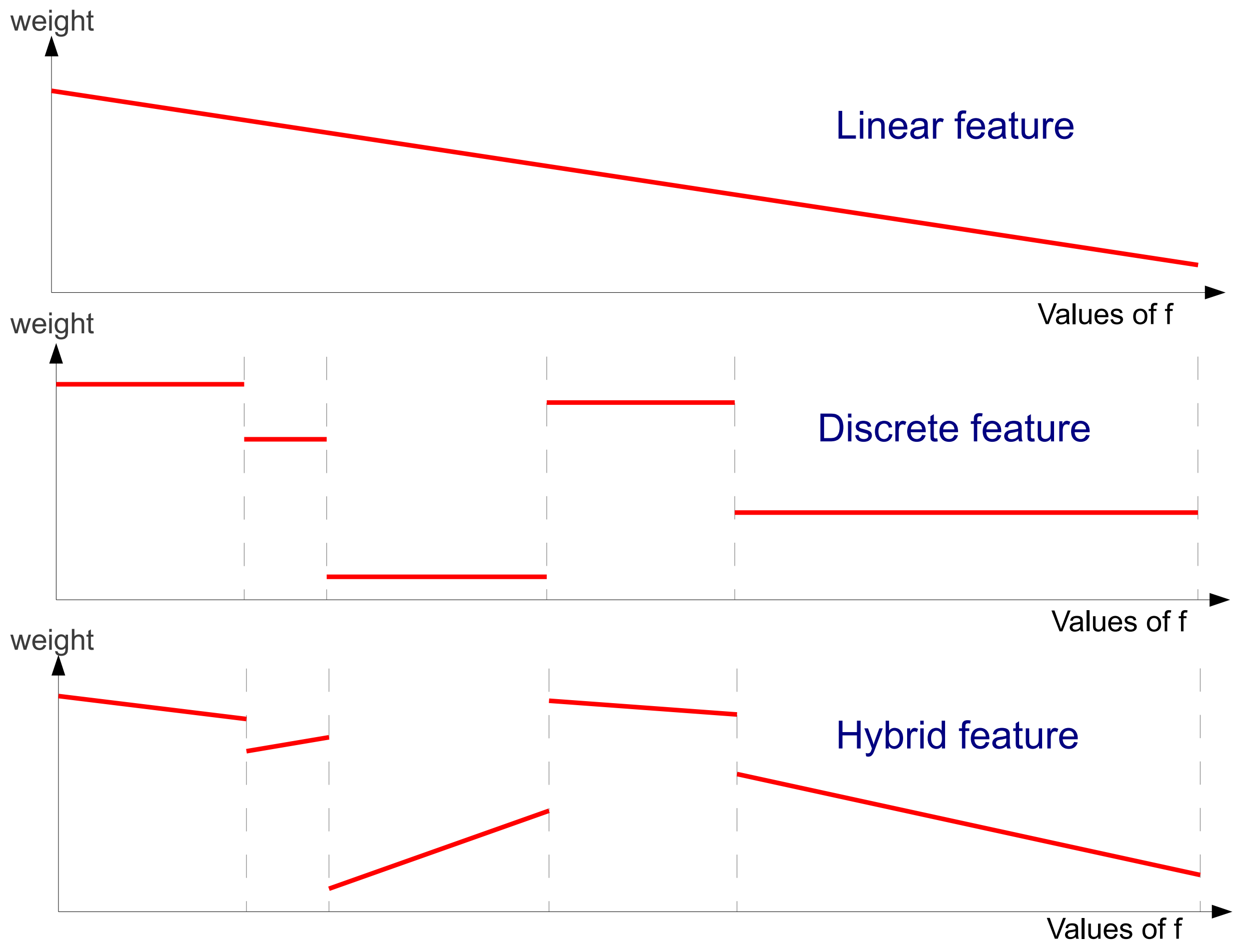


Figure E.1: Representation of the weights that can be learned by *GAP* depending on the type of the feature *f*.

This hybrid approach cancels the shortcomings of *LAP* that we mentioned above.

- Missing values within a linear feature can be considered as a value corresponding to a non-linear weight (indeed, it is not compulsory to consider all the possible interval/values of a feature as linear weights. Some can remain discrete. The interesting idea about this is not to learn things about the contribution of a missing value even though if there is something to learn, *GAP* will eventually learn it. What is interesting is that if there are a lot of missing values for a given feature, its uncertainty will be higher than the one of the others, leading to a smarter exploration behavior. Moreover the challenge pointed out the performance lift of having contributors corresponding to missing values as opposed to simply ignoring them.

- It is possible to consider continuous features on which we have little knowledge as hybrid features with a uniform discretization instead of considering them blindly as simply linear. This has two major advantages: (i) even if the feature turns out to be more or less linear, the uncertainty is not uniform anymore which can be useful if a feature has an unstable behavior at his bounds for instance (ii) if the feature is not linear, more complex functions can be coarsely approximated by linear by part functions which partly lifts the need to add non-linear functions of the continuous features as additional features. The same method can be applied to any function of the original feature and so without knowing anything about that function in advance. However one may have to keep in mind that the finer the uniform discretization, the longer it will take for *GAP* to capture the feature's actual contribution. We then face some kind of accuracy *versus* learning speed trade-off.
- This algorithm is very flexible and allows an engineering team who tries to build a recommender system to input a lot of marketing insights in various forms to the learning algorithm such as different levels of uncertainty depending on the feature or even depending on the values of the feature (*e.g.* old people may have a less uncertain behavior than young people). Flexibility could however be considered as a shortcoming as it means many parameters but as we will see in the experimental section, this number can be significantly reduce without hurting to much the results. This can also be considered as a way to communicate between the different teams working on recommendation (mainly engineering and marketing) as the algorithm is highly tunable and transparent.

We do not provide the pseudo-code for this hybrid approach as it can be straightforwardly derived from *LAP* (algorithm 8) and *Ad Predictor* (algorithm 7). The update rules are the following:

- For the linear features, we update the weights as in *LAP*.
- For the discrete features, we update the weights as in *Ad predictor*.
- For the hybrid features, we update the linear weights which are active (as we have several linear weights per feature) for the considered display as in *LAP*. Note once more than within an hybrid feature, some weights can be chosen to be discrete. This is what one may want to do with the missing values for instance or to values that are out of range.

Remark on the necessary weights When using *LAP*, one weight per feature is enough. In general we also add an intercept term (a feature always equal to one) to account for a possible constant contribution to the click probability. This is equivalent to adding an always active non-linear weight. Similarly for an hybrid feature which is linear by part, using only one linear weight per part is not what we want to do as it assumes that if the feature is equal to zero, its contribution is also zero. What we want to do most of the time is inferring a function which is affine ($y = ax + b$) by part (see on fig. E.1 that not all the lines necessarily cross the origin). A user of *GAP* may want to set for each part of his discretization a non-linear Gaussian weight to learn b and a linear weight for a.

To conclude this section, we came up with those hybrid features after the end of the second phase of the challenge. This is the reason why we did not talk about them in section 2.4 when talking about the successive improvement of our approach. We did not run a lot of experiments as the number of possibilities is exponential and it would have required something quite intensive. Nevertheless only transforming all the discretized features in hybrid linear features did not hurt our performance. On the contrary, we seemed to get a few additional points but nothing significant enough to be quantified here.

E.2 Trying to handle the dynamics in the challenge

Algorithm 19 Shifting models : Using more than one model to handle the dynamics. *chooseDisplay*$_M$ (resp. *update*$_M$) is the same as *chooseDisplay* (resp. *update*) but the model on which it is applied is specified in subscript since there are several of them. Note also that the *chooseDisplay*$_M$ and *update*$_M$ procedures to which we refer to in the algorithm are not defined here. Indeed the shifting models approach can be used with any kind of model. For an example of these two procedures, the reader can refer to the winning algorithm (alg. 7).

```
parameter $M_{max}$: the maximum number of distinct models.
parameter $\tau$: the number of steps between two changes of models.
parameter $(m_{c(p)}, \sigma_{c(p)})$: a prior for each contributor $c$ of the model.

procedure INIT
    $modelList \leftarrow emptyList$
    $i \leftarrow 0$
    initialize a model $M$ with the priors in parameter
    add $M$ to $modelList$
end procedure
function CHOOSEDISPLAY($batch$)
    $M \leftarrow lastElement(modelList)$
    return $chooseDisplay_M(batch)$
end function
procedure UPDATE(display, click)
    for each $Model\ M \in modelList$ do
        $update_M(display, click)$
    end for
    $i \leftarrow i + 1$
    if $i = \tau$ then
        if $size(modelList) = M_{max}$ then
            remove the last element of $modelList$
        end if
        $M_{new} \leftarrow copyoffirstElement(modelList)$
        for each contributor $c$ in $M_{new}$ do
            $\sigma_{M_{new},i} \leftarrow \sigma_{c(p)}$
        end for
        add $M_{new}$ at the beginning of $modelList$
        $i = 0$
    end if
end procedure
```

This section - which completes section 2.4.3 is about modifications that can be brought to Ad predictor or GAP to make them capable of dealing with a changing environment and in particular with shifting reward distributions as a function of the features. Let us recall that nothing worked in the context of the challenge but for specific reasons. The methods we expose here could be useful in other contexts.

E.2.1 Microsoft's idea

Herbrich *et al.* [6] propose the following update rule to be used at each time step and for each weight of each feature in replacement of the one we previously introduced:

$$\begin{aligned} \sigma_i^2 &\leftarrow \frac{\sigma_{i(p)}^2 \cdot \sigma_i^2}{(1-\varepsilon)\cdot\sigma_{i(p)}^2 + \varepsilon\cdot\sigma_i^2} \\ m_i &\leftarrow \sigma_i^2 \cdot \left((1-\varepsilon)\frac{m_i}{\sigma_i^2} + \varepsilon\cdot\frac{m_{i(p)}}{\sigma_{i(p)}^2}\right), \end{aligned}$$

where $\mathcal{N}(m_i, \sigma_i)$ indicates the current value of the contributor and $\mathcal{N}(m_{i(p)}, \sigma_{i(p)})$ its prior value. The idea behind these update equations is to let the weights evolve back to their prior in order to slowly forget the influence of past data. This allows the model to adapt to the dynamics since the uncertainty is prevented from going to zero. In the equations, the bigger ε is the faster the algorithm forgets. Unfortunately the smaller ε was the better the algorithm performed in the challenge. The better performance was even achieved with $\varepsilon = 0$, that is when nothing at all was being forgotten and nothing could change in the model once updated a sufficient number of times.

E.2.2 Two or more models

The previous approach is not fully satisfying as it cannot take into account the fact that some weights may evolve at different speeds. Some others may also not evolve at all. Moreover, the convergence of the weights towards their prior makes the model learn slower as we are always forgetting a little.

To deal with these issues, we propose to use several separate models. At each time step only the oldest model is used to make a prediction but all these distinct models are updated by the observation. Every τ time steps, we destroy the oldest model and replace it by a new one and the second oldest becomes the oldest, the one that we use for predictions. This new model can be initialized in several ways. The simplest one would be to always give to each weight the same prior. Then we would handle the dynamics by making predictions with a model which has been learning for $\tau \cdot (M_{max} - 1)$ to $\tau \cdot M_{max}$ time steps, with M_{max} the number of models being concurrently updated. A good idea if we do think that some parameters of the environment are not evolving or evolving very slowly is to give to all the weights the mean of their values in the previous model but to reset the variances to predefined values. This is the version for which we provide the pseudo code (algorithm 19). We tried a bunch of multi-model approaches but to our surprise, nothing seemed to work. Contrarily to what was advertised by the organizers, the environment just seemed plain static. However we will see in section 2.5 that we have good reasons to think that this failure was due to the challenge itself and not to the application the data was from. This problematic of dealing with different speeds of evolution for the various parameters of a model would be a very interesting topic of further study.

E.3 More experiments with Ad Predictor

This section studies two properties of Ad Predictor. The first one is its ability to predict click rates of displays (what it was designed for) compared to its ability to rank displays (what we used it for). We then study how this algorithm responds to parameter tuning.

E.3.1 Click rate prediction

Graepel *et al.* [6] proceed the following way to infer click probabilities from the model:

$$p(d) = \Phi\left(\frac{s(d)}{\sqrt{\beta^2 + \sigma^2}}\right)$$

with the same notations as in section 2.3.3.

We experiment this equation on a toy example. In this example, we assume that reality is modeled as follows: each click probability is the result of a sum of contributors. In the following experiment we use 10 features which can each take 5 possible values. For each feature f_i, the hidden values of the 5 contributors are as follows:

$$\{10^{-4}.p_i^0,\ 10^{-4}.p_i^1,\ 10^{-4}.p_i^2,\ 10^{-4}.p_i^3,\ 10^{-4}.p_i^4\}.$$

The real parameters p_i are real numbers uniformly drawn in $[1, 5]$ at the beginning of the experiment.

At each time step a random display is presented with its reward to update the model (the values of the features are drawn uniformly). The result are quite interesting. The error on the predictions naturally converges as the number of displays increases but its value remains very high when it stabilizes (after $20,000$ displays). Indeed the squared error stabilizes around 2.10^{-4}. The mean of the click probabilities is around 0.04 and the RMSE (to be on the same scale) is $\sqrt{2.10^{-4}} \approx 0.014$ which represents 35% of the average click probability. This is huge.

However when simply considering the order of the click probabilities (which is what we asked Ad Predictor to do in the challenge), we notice something different. The ranking is learned very fast. The model reaches a rate of 80% of well ranked probabilities after less than $1,000$ displays, reaches 90% after $40,000$ displays and then goes up to 95%. This explains the success of the algorithm when it comes to choosing between displays during the challenge.

We can then remark that ranking displays, which was the task of the challenge is very likely to be a much easier task than click rate prediction, for which Ad predictor only exhibits modest performance.

E.3.2 Influence of the parameters

The model has a few parameters and we propose here to study their influence on the performance of our algorithm. Note that we will not talk about α since the best option in the challenge was to do no exploration at all ($\alpha = 0$).

The parameter β

β impacts the learning speed. Fig. E.2 shows that the algorithm is not very sensitive to its value. Any value between 300 and 600 achieves almost the best performance. Then when β grows we do not learn fast enough and the performance decreases until stabilizing around 1630. If β gets closer to 0 we try to learn too fast and the performance dramatically decreases until 1200 (random strategy).

The priors on the Gaussian weights

Another important parameter is the prior we take for the Gaussian weights. As we had no prior knowledge about the data, we had to give default values. As far as the mean is concerned, if it is between -1 and 1 it does not change anything. For larger values, we eventually learn correctly but it takes more time, hence a decrease in the performance.

Optimizing the standard deviation (σ) was more interesting. As we can see on Fig. E.3 we can find an optimal value to give to each weight which is around 20. However Fig. E.4a and E.4b (the latter being a zoom on the best scores of the former) show that we can do better. If we initialize with different values the standard deviation of the discrete features (σ_d) and the standard deviation of the continuous features (σ_c), we can win something like 30 points. It seems to mean that learning from $\mathcal{C}_D$ is easier than learning from $\mathcal{C}_c^*$. We tried to split $\mathcal{C}_c^*$ and give different values of σ to each resulting group but nothing significant came out. Note that on Fig. E.4a even though we can find optimal values for σ_d and σ_c the algorithm remains very stable. Only very low values for σ (less than 2) or very high values (more than 80) lead to scores inferior to 2000.

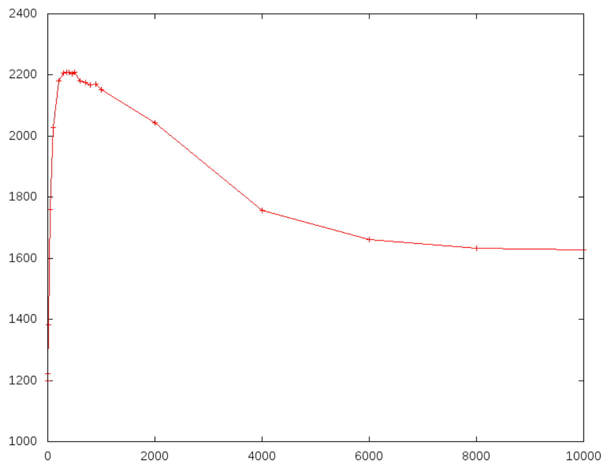


Figure E.2: Score in the challenge against β.

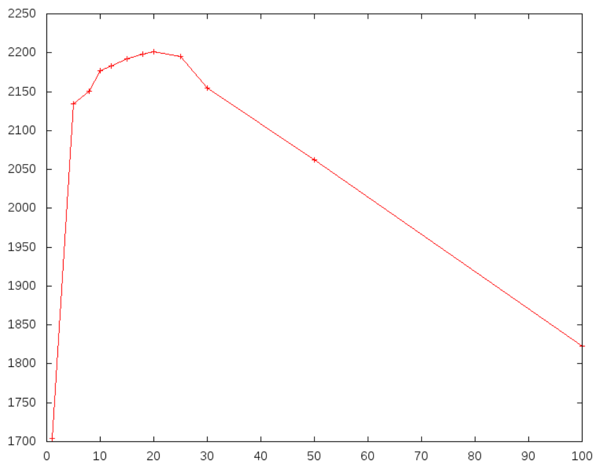
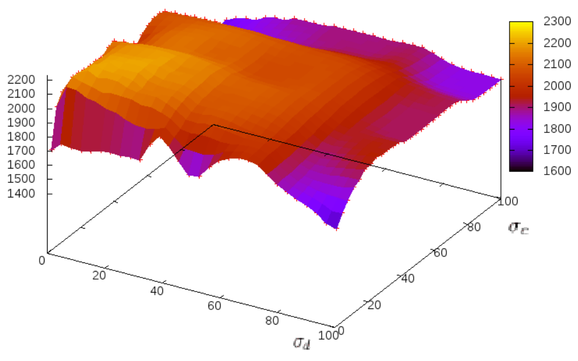
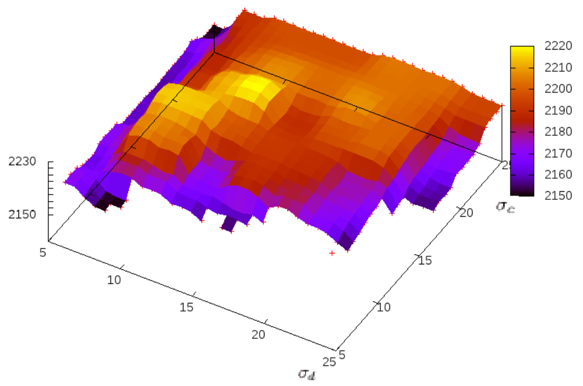


Figure E.3: Score in the challenge against σ.



(a) Large scale (all the values considered in the experiment).



(b) Zoom on the best mean scores achieved in the challenge.

Figure E.4: Score in the challenge against σ_d and σ_c, the priors on the uncertainty (standard deviation) of the discrete and continuous features respectively. The larger they are, the more time it takes for the algorithm to converge (and the less risk we take to converge to a suboptimal solution). We clearly have to balance risk and learning speed.

Appendix F

Proofs of the theorems of chapter 3

Contents

F.1 Bias of the replay method

This is the proof of theorem 3 from section 3.5.1.

Proof. The proof simply comes from the law of total expectation. $\sum_{t=1}^{T} V_t$ can take values from 0 to T and is a variable following a binomial distribution $B(T, \frac{1}{K})$ with K the size of the pool of actions.

Note also that:

$$\frac{\sum_{t=1}^{T} V_t.r_t}{\sum_{t=1}^{T} V_t} = \frac{\sum_{t=1}^{T} V_t.\vec{r_t}[a_{\pi,x_t}]}{\sum_{t=1}^{T} V_t} ,$$

with a_{π,x_t} the choice made by π at iteration t given x_t.

These two expressions are equal because $\vec{r_t}[\pi(x_t)]$ is equal to r_t when $V_t = 1$ and different only when $V_t = 0$. Note that we do not use $\pi(x_t)$ here because as π can be stochastic, this can be a discrete distribution of actions.

Thus first using the law of total expectation and then the linearity of expectation we have:

$$\begin{aligned}
\mathbb{E}\,\hat{g}_\pi^{(replay)}(S) &= \mathbb{P}\left(\sum_{t=1}^{T} V_t = 0\right).0 + \sum_{i=1}^{T} \mathbb{P}\left(\sum_{t=1}^{T} V_t = i\right).\mathbb{E}\left[\frac{\sum_{t=1}^{T} V_t.\vec{r_t}[a_{\pi,x_t}]}{\sum_{t=1}^{T} V_t} \,\middle|\, \sum_{t=1}^{T} V_t = i\right] \\
&= \sum_{i=1}^{T} \mathbb{P}\left(\sum_{t=1}^{T} V_t = i\right).\mathbb{E}\left[\frac{\sum_{t=1}^{T} V_t.\vec{r_t}[a_{\pi,x_t}]}{i} \,\middle|\, \sum_{t=1}^{T} V_t = i\right] \\
&= \sum_{i=1}^{T} P\left(\sum_{t=1}^{T} V_t = i\right).\frac{\sum_{t=1}^{T} \mathbb{E}\left[V_t.\vec{r_t}[a_{\pi,x_t}] \,\middle|\, \sum_{t=1}^{T} V_t = i\right]}{i} .
\end{aligned}$$

Then, all the $V_t.\vec{r_t}[a_{\pi,x_t}]$ are independent since all the $(x_t, \vec{r_t})$ were drawn from $\mathcal{D}$, a_t was chosen by a random uniform policy and π is static. So we have that:

$$\sum_{t=1}^{T} \mathbb{E}\left[V_t.\vec{r_t}[a_{\pi,x_t}] \,\middle|\, \sum_{t=1}^{T} V_t = i \right] = T. \mathop{\mathbb{E}}_{(x,\vec{r})\sim\mathcal{D}, a\sim\mathcal{U}(A)} \left[V.\vec{r}[\pi(x)] \,\middle|\, \sum_{t=1}^{T} V_t = i \right] .$$

Furthermore, the reward that π would obtain after a recommendation in context x ($\vec{r}[\pi(x)]$) is independent from whether or not π would be evaluated for that same recommendation by the replay method (V) because of that same random uniform logging policy. Thus:

$$\begin{aligned}
\mathop{\mathbb{E}}_{(x,\vec{r})\sim\mathcal{D}, a_t\sim\mathcal{U}(A)} \left[V.\vec{r}[\pi(x)] \,\middle|\, \sum_{t=1}^{T} V_t = i \right] &= \mathop{\mathbb{E}}_{(x,\vec{r})\sim\mathcal{D}, a\sim\mathcal{U}(A)} \left[V \,\middle|\, \sum_{t=1}^{T} V_t = i \right] . \mathop{\mathbb{E}}_{(x,\vec{r})\sim\mathcal{D}} \Big[\vec{r}\,[\pi(x)] \Big] \\
&= \mathop{\mathbb{E}}_{(x,\vec{r})\sim\mathcal{D}, a\sim\mathcal{U}(A)} \left[V \,\middle|\, \sum_{t=1}^{T} V_t = i \right] .g_\pi \; .
\end{aligned}$$

Finally the calculation ends like this:

$$\begin{aligned}
\mathbb{E}\, \hat{g}_\pi^{(replay)}(S) &= \sum_{i=1}^{T} \mathbb{P}\left(\sum_{t=1}^{T} V_t = i \right) \frac{T. \mathbb{E}_{(x,\vec{r})\sim\mathcal{D}, a\sim\mathcal{U}(A)} \left[V \,\middle|\, \sum_{t=1}^{T} V_t = i \right] .g_\pi}{i} \\
&= \sum_{i=1}^{T} \mathbb{P}\left(\sum_{t=1}^{T} V_t = i \right) \frac{T}{i} \frac{i}{T} g_\pi \\
&= g_\pi \sum_{i=1}^{T} \mathbb{P}\left(\sum_{t=1}^{T} V_t = i \right) \\
&= g_\pi \left(1 - \mathbb{P}\left(\sum_{t=1}^{T} V_t = 0 \right) \right) \\
&= g_\pi - \left(\frac{K-1}{K} \right)^T g_\pi \; .
\end{aligned}$$

$g_\pi \left(\frac{K-1}{K}\right)^T$ obviously decreases exponentially as T increases since $\frac{K-1}{K} < 1$. This finishes the proof. □

F.2 Unbiasedness of replay*

This is the proof of theorem 4 from section 3.5.2.

Proof. The proof is pretty straightforward and uses the same independence property as the proof of replay's bias (section F.1).

First we have:

$$\begin{aligned}
\mathbb{E}\, \hat{g}_\pi^{(replay*)}(S) &= \mathbb{E}\left[\frac{K}{T} \sum_{t=1}^{T} V_t.\vec{r_t}[a_{\pi,x_t}] \right] \\
&= \frac{K}{T} \sum_{t=1}^{T} \mathbb{E}\left[V_t.\vec{r_t}[a_{\pi,x_t}] \right] \; .
\end{aligned}$$

For the same independence reasons as in the previous proof (see F.1) we can then write:

$$\begin{aligned}
\mathbb{E}\,\hat{g}_{\pi}^{(replay*)}(S) &= \frac{K}{T}T \underset{(x,\vec{r})\sim\mathcal{D},a\sim\mathcal{U}(A)}{\mathbb{E}}[V.\vec{r}[\pi(x)]] \\
&= K \underset{(x,\vec{r})\sim\mathcal{D},a\sim\mathcal{U}(A)}{\mathbb{E}}[V.\vec{r}[\pi(x)]] \\
&= K \underset{(x,\vec{r})\sim\mathcal{D},a\sim\mathcal{U}(A)}{\mathbb{E}}[V] \underset{(x,\vec{r})\sim\mathcal{D}}{\mathbb{E}}[\vec{r}[\pi(x)]] \\
&= K\frac{1}{K}g_{\pi} \\
&= g_{\pi} \ .
\end{aligned}$$

which finishes the proof. □

F.3 A tighter bound for the replay method

This is the proof of theorem 5 from section 3.5.3.

Proof. The sketch of the proof is quite simple and follows exactly what we said previously. We lower bound in probability $\sum_{t=1}^{T} V_t$, that we denote N, using a result derived from the Chernoff bound. We then use the actual multiplicative Chernoff bound and the previous lower bound to bound the error of the estimator.

The key of this proof is that we can rewrite the estimator as follows:

$$\hat{g}_{\pi}^{(replay)} = \frac{\sum_{t=1}^{T} V_t.r_t}{\sum_{t=1}^{T} V_t} = \frac{1}{N}\sum_{i=1}^{N}\vec{r_i}[a_{\pi,x_i}] \ ,$$

with N following a binomial distribution ($N \sim B(\frac{1}{K}, T)$) and a_{π,x_i} the choice made by π at the i^{th} iteration for which $V_t = 1$. This is true because r_t, the reward present in the dataset and the choice made by π are different only when $V_t = 0$ so this equation simply comes from a re-indexing and that all the V_t are independent.

Bounding N As possible small values of N are what limit the estimator's concentration, let us bound the probability of encountering such small values. To do so we use the following result that can be proved using the general Chernoff bound [172]:

Lemma 2. *For all X which follows a Binomial distribution $B(n,p)$ of mean $\mu = np$ and for all $\gamma \in]0,1[$ we have that:*

$$P\left(X < (1-\gamma)\,\mu\right) \leq exp\left(-\frac{\gamma^2\mu}{2}\right) \ .$$

As N follows a Binomial distribution $B(T, \frac{1}{K})$ we have that for any $0 < \gamma < 1$:

$$P\left(N \leq (1-\gamma)\frac{T}{K}\right) \leq exp\left(-\frac{\gamma^2 T}{2K}\right) \ .$$

Let the right hand side of this inequality be $\frac{\delta}{2}$ and solve for γ:

$$\gamma = \sqrt{\frac{2Kln(2/\delta)}{T}} \ .$$

Bounding the estimator Now, if we fix N so that it is non-negative and only index with i the records of the dataset for which π chose the same action as the one in S, we have as we wrote it at the beginning of the proof that:

$$\hat{g}_\pi^{(replay)} = \frac{1}{N}\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] \ .$$

The sum in this equation is a sum of independent random variables because of all the independence reasons that we already mentioned when proving the unbiasedness results: the reward obtained by π - $\vec{r_i}[a_{\pi,x_i}]$ - is independent from whether or not this reward is taken into account in the evaluation because the actions were chosen uniformly and all the successive rewards are i.i.d. because π and $\mathcal{D}$ are stationary.

Thus we can use the multiplicative Chernoff bound to get the following result:

$$\begin{aligned} P\left(\left|\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] - Ng_\pi\right| \geq \lambda N g_\pi\right) &\leq 2exp\left(-\frac{\lambda^2 N g_\pi}{3}\right) \\ P\left(\left|\frac{1}{N}\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] - g_\pi\right| \geq \lambda g_\pi\right) &\leq 2exp\left(-\frac{\lambda^2 N g_\pi}{3}\right) \ . \end{aligned}$$

If we let the right hand side of the last inequality be $\frac{\delta}{2}$ and solve for λ then we get:

$$\lambda = \sqrt{\frac{3ln(4/\delta)}{Ng_\pi}} \ .$$

Gathering the two results with a union bound Using a simple union bound on the two aforementioned probabilities, both bounded by $\frac{\delta}{2}$ we have that:

$$P\left(\left|\frac{1}{N}\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] - g_\pi\right| \geq \lambda g_\pi \quad \underline{OR} \quad N \leq (1-\gamma)\frac{T}{K}\right) \leq \delta \ .$$

So with probability at least $1-\delta$ the following two inequalities are both true at the same time:

$$\begin{cases} \left|\frac{1}{N}\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] - g_\pi\right| < \lambda g_\pi \ , \\ N > (1-\gamma)\frac{T}{K} \ . \end{cases}$$

The first inequality is the one that is really of interest to us for it bounds the error of the estimator. We finish first part of the proof (proving the raw bound) with the following calculation:

$$\begin{aligned} \left|\frac{1}{N}\sum_{i=1}^{N} \vec{r_i}[a_{\pi,x_i}] - g_\pi\right| &< \sqrt{\frac{3ln(4/\delta)g_\pi}{N}} \\ &< \sqrt{\frac{3ln(4/\delta)g_\pi}{\left(1-\sqrt{\frac{2Kln(2/\delta)}{T}}\right)\frac{T}{K}}} \\ &< \sqrt{\frac{3Kln(4/\delta)g_\pi}{T-\sqrt{2TKln(2/\delta)}}} \\ &= O\left(\sqrt{\frac{Kg_\pi ln(1/\delta)}{T}}\right) \ . \end{aligned}$$

The first line comes when replacing λ by its value. The second one is the most important one. As N is on the denominator, replacing it by its lower bound $(1-\gamma)\frac{T}{K}$ preserves the inequality and its direction. The third line is trivial and the fourth comes from the fact that if T is large enough, T is much larger than $\sqrt{2Tln(2/\delta)}$.

Decomposition To be able to decompose the bound "outside" of the square root, we need to slightly loosen it. We use the fact that $\sqrt{x}-\sqrt{y}<\sqrt{x-y}$ when $x>y>0$ to lower bound the denominator and thus to upper bound the bound we proved (1st line). It yields the following calculation.

$$\begin{aligned}
\sqrt{\frac{3Kln(4/\delta)g_\pi}{T-\sqrt{2TKln(2/\delta)}}} &< \frac{\sqrt{3Kln(4/\delta)g_\pi}}{\sqrt{T}-\sqrt{\sqrt{2TKln(2/\delta)}}} \\
&= \sqrt{\frac{3Kg_\pi ln\left(\frac{4}{\delta}\right)}{T}}+\frac{\sqrt{\sqrt{2KTln\,(2/\delta)}}x\sqrt{3Kg_\pi ln\,(4/\delta)}}{T-\sqrt{T\sqrt{2KTln\,(2/\delta)}}} \\
&= \sqrt{\frac{3Kg_\pi ln\,(4/\delta)}{T}}+O\left(\sqrt{g_\pi}\left(\frac{Kln\,(1/\delta)}{T}\right)^{3/4}\right) .
\end{aligned}$$

The second line was obtained by subtracting the highest order term from the bound. It can be verified easily by summing the decomposition. The third line is rather straightforward. It mainly comes from noticing that the denominator is of the form $T-O(T^{3/4})$ and is thus in $O(T)$. This concludes the second part of the proof.

□

F.4 An exact bound for replay*

This is the proof of theorem 6 from section 3.5.4.

Proof. We want to bound the estimator:

$$\hat{g}_\pi^{(replay*)}=\frac{K}{T}\sum_{t=1}^{T}V_t.r_t=\frac{K}{T}\sum_{t=1}^{T}V_t.\vec{r_t}[a_{\pi,x_t}] ,$$

with a_{π,x_t} the choice made by π when seeing x_t at iteration t and $\vec{r_t}$ the vector of reward drawn from $\mathcal{D}$ when acquiring S. As we mentioned it before the equality is true because $r_t\neq\vec{r_t}[a_{\pi,x_t}]$ only when $V_t=0$. All the $V_t.\vec{r_t}[a_{\pi,x_t}]$ are i.i.d. for reasons we gave previously and for each t, V_t and r_t are independent.

For these reasons, $\sum_{t=1}^{T}V_t.r_t$ is a sum of i.i.d. Bernoulli random variables of parameter $p=\frac{g_\pi}{K}$. We can bound using the multiplicative Chernoff bound as follows:

$$\begin{aligned}
\mathbb{P}\left(\left|\sum_{t=1}^{T}V_t.r_t-T\frac{g_\pi}{K}\right|\geq\frac{Tg_\pi\gamma}{K}\right) &\leq 2exp\left(-\frac{T\gamma^2g_\pi}{3K}\right) \\
P\left(\left|\frac{K}{T}\sum_{t=1}^{T}V_t.r_t-g_\pi\right|\geq g_\pi\gamma\right) &\leq 2exp\left(-\frac{T\gamma^2g_\pi}{3K}\right) .
\end{aligned}$$

If we take the right hand side of the equation to be δ and solve we have:

$$\gamma=\sqrt{\frac{3Kln(2/\delta)}{Tg_\pi}} .$$

Finally with probability $1-\delta$ we have that:

$$\begin{aligned}\left|\frac{K}{T}\sum_{t=1}^{T}V_t.r_t - g_\pi\right| &< g_\pi\gamma\\ &< \sqrt{\frac{3Kg_\pi ln(2/\delta)}{T}}\ ,\end{aligned}$$

which concludes the proof.

□

F.5 Variance of replay*

This is the proof of theorem 7 from section 3.5.6.

Proof.

$$\begin{aligned}Var\left(\frac{K}{T}\sum_{t=1}^{T}V_t r_t\right) &= \frac{K^2}{T^2}Var\left(\sum_{t=1}^{T}V_t\vec{r}_t\left[\pi\left(x_t\right)\right]\right) = \frac{K^2}{T^2}T\ Var\left(V.\vec{r}\left[\pi\left(x\right)\right]\right)\\ &= \frac{K^2}{T}\left[\mathbb{E}\left(V^2\right)\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]^2\right) - \mathbb{E}\left(V\right)^2\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K^2}{T}\left[\mathbb{E}\left(V\right)\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]^2\right) - \mathbb{E}\left(V\right)^2\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K^2}{T}\left[\frac{1}{K}\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]^2\right) - \frac{1}{K^2}\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K}{T}\left[\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]^2\right) - \frac{1}{K}\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K}{T}\left[\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]^2\right) - \mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2 + \mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2 - \frac{1}{K}\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K}{T}\left[Var\left(\vec{r}\left[\pi\left(x\right)\right]\right) + \frac{K-1}{K}\mathbb{E}\left(\vec{r}\left[\pi\left(x\right)\right]\right)^2\right]\\ &= \frac{K}{T}Var\left(\vec{r}\left[\pi\left(x\right)\right]\right) + \frac{K-1}{T}.g_\pi^2 \quad = \frac{K}{T}Var\left(\vec{r}\left[\pi\left(x\right)\right]\right) + O\left(\frac{Kg_\pi^2}{T}\right)\ .\end{aligned}$$

The two successive transformations of r on the first line are not trivial. They are explained at the beginning of the proof of theorem 3. The second line comes from the fact that the reward (r_t) is independent from the event that it is taken into account by the estimator (V_t).

This proves theorem 7 and the first part of theorem 9. □

F.6 Expected Squared Error (ESE) of replay

This is the proof of theorem 8 from section 3.5.6

Proof.

$$
\begin{aligned}
& \mathbb{E}\left[\left(\frac{\sum_{t=1}^{T} V_t r_t}{\sum_{t=1}^{T} V_t} - g_\pi\right)^2\right] \\
= \; & P\left(\sum_{t=1}^{T} V_t = 0\right) . \mathbb{E}\left[(0-g_\pi)^2\right] + \sum_{i=1}^{T} P\left(\sum_{t=1}^{T} V_t = i\right) . \mathbb{E}\left[\left(\frac{\sum_{t=1}^{T} V_t r_t}{\sum_{t=1}^{T} V_t} - g_\pi\right)^2 \,\middle|\, \sum_{t=1}^{T} V_t = i\right] \\
= \; & \left(\frac{K-1}{K}\right)^T g_\pi^2 + \sum_{i=1}^{T} P\left(X = i \,\middle|\, X \sim B\left(\frac{1}{K}, T\right)\right) \mathbb{E}\left[\left(\sum_{k=1}^{i} r_k - g_\pi\right)^2\right] \\
= \; & \left(\frac{K-1}{K}\right)^T g_\pi^2 + \sum_{i=t}^{T} \binom{T}{i} \left(\frac{1}{K}\right)^i \left(\frac{K-1}{K}\right)^{(T-i)} \frac{Var\left(\vec{r}\left[\pi\left(x\right)\right]\right)}{i} \\
= \; & \left(\frac{K-1}{K}\right)^T g_\pi^2 + Var\left(\vec{r}\left[\pi\left(x\right)\right]\right) \sum_{i=t}^{T} \binom{T}{i} \frac{(K-1)^{(T-i)}}{i.K^T} \; .
\end{aligned}
$$

The first equality comes from the law of total expectations. Then, in the second equality, the first term of the sum simplifies easily. In the second one, we can rewrite the expression of $\hat{g}_\pi^{(replay)}$ for when $\sum V_t = i$. Rewriting the denominator is straightforward: it is equal to i. For the numerator, we use the fact that V and the obtained reward (r) are independent (see the proof of theorem 3). It follows that the numerator is in fact the sum of i independent realizations of $\vec{r}[\pi(x)]$. Finally the third equality comes by noticing that the expectation that remains is in fact the variance of a sample mean on a sample of i *i.i.d.* random variables. The last one is a refactoring of the sum.

□

F.7 Comparison of the ESE of replay and replay*

This is the proof of theorem 9 from section 3.5.6.

The theorem states that asymptotically in T,

$$
\begin{aligned}
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay*)} - g_\pi\right)^2\right] &= \frac{KVar\left(\vec{r}[\pi(x)]\right)}{T} + O\left(\frac{1}{T}\right) \; , \\
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] &= \frac{KVar\left(\vec{r}[\pi(x)]\right)}{T} + O\left(\frac{1}{T^2}\right) \; .
\end{aligned}
$$

We also provide intuition justifying that the result could be proven for finite and relatively small values of T.

Proof. The first equality is straightforward and its proof is at the end of the proof of theorem 7.

In what follows, we shall decompose the expression of the ESE of replay that was proved in theorem 8, and provide a tight upper bound that meets the expected result.

Let us first introduce notations to lighten the calculations:

$$
Var(r) = Var\left(\vec{r}\left[\pi(x)\right]\right) .
$$

$B\left(\frac{1}{K}, T\right)$, the distribution of V_t converges toward $\mathcal{N}\left(\frac{\mathcal{T}}{\mathcal{K}}, \left(\frac{\sqrt{\mathcal{T}(\mathcal{K}-\infty)}}{\mathcal{K}}\right)^{\in}\right)$, whose probability density in x is denoted by $\phi_r(x)$.

The calculation then starts as follows.

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] &= \left(\frac{K-1}{K}\right)^T + Var(r)\sum_{i=1}^{T}\frac{P\left(X=i \mid X \sim B\left(\frac{1}{K},T\right)\right)}{i} \\
&= O\left(e^{-T}\right) + Var(r)\int_{-\infty}^{+infty}\frac{\phi_r(x)}{x}\mathrm{d}x \\
&= O\left(e^{-T}\right) + Var(r)\int_{\frac{T}{K}-a\frac{\sqrt{T(K-1)}}{K}}^{\frac{T}{K}+a\frac{\sqrt{T(K-1)}}{K}}\frac{\phi_r(x)}{x}\mathrm{d}x
\end{aligned}$$

where a is a finite number.

The third line comes from the fact that asymptotically, the binomial distribution is a normal distribution of mean $\mu = T/K$ and standard deviation $\sigma = \frac{\sqrt{T(K-1)}}{K}$. Note that by considering the entire integral (from $-\infty$ to $+\infty$) we reintroduce the case of $\sum V_t = i = 0$ but even for a finite T, this case has an exponentially low probability. The last line comes from the fact the tails (*i.e.* the integrals between $-\infty$ and $\mu - a\sigma$ and between $\mu + a\sigma$ and $+\infty$) are exponentially small. In other words, all the density of the normal distribution (minus an exponentially small quantity) is within a distance a times its standard deviation from its mean, where a is a finite number. We can thus remove them from the sum. The term $O\left(e^{-T}\right)$ accounts for them. Notice that this line of reasoning argument is most certainly true way before $T = +\infty$ and this is why we suspect that this result holds even in a non-asymptotic setting. Yet the setting allows a clear and nice proof which will suffice in this work.

We pursue by changing the index of the sum (1st line), use the symmetry of the normal distribution (2nd line) and simplify

$$\begin{aligned}
&\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] \\
&= O\left(e^{-T}\right) + Var(r)\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{+a\frac{\sqrt{T(K-1)}}{K}}\frac{\phi_r\left(\frac{T}{K}+j\right)}{\frac{T}{K}+j}\mathrm{d}x \\
&= O\left(e^{-T}\right) + Var(r)\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{0}\left(\frac{\phi_r\left(\frac{T}{K}+j\right)}{\frac{T}{K}+j} + \frac{\phi_r\left(\frac{T}{K}+j\right)}{\frac{T}{K}-j}\right)\mathrm{d}x \\
&= O\left(e^{-T}\right) + Var(r)\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{0}\frac{\phi_r\left(\frac{T}{K}+j\right)\left(\frac{T}{K}-j\right) + \phi_r\left(\frac{T}{K}+j\right)\left(\frac{T}{K}+j\right)}{\left(\frac{T}{K}\right)^2 - j^2}\mathrm{d}x \\
&= O\left(e^{-T}\right) + Var(r)\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{0}\frac{2\phi_r\left(\frac{T}{K}+j\right)\frac{T}{K}}{\left(\frac{T}{K}\right)^2 - j^2}\mathrm{d}x
\end{aligned}$$

Finally we coarsely upper bound the sum by replacing j (which is subtracted from the denominator) by its greatest value. Indeed, the greater this quantity, the smaller the denominator and the greater the whole term.

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] &< O\left(e^{-T}\right) + Var(r)\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{0}\frac{2\phi_r\left(\frac{T}{K}+j\right)\frac{T}{K}}{\left(\frac{T}{K}\right)^2 - \left(a\frac{\sqrt{T(K-1)}}{K}\right)^2} \\
&= O\left(e^{-T}\right) + \frac{Var(r)\frac{T}{K}}{\left(\frac{T}{K}\right)^2 - \frac{a^2T(K-1)}{K^2}}\int_{-a\frac{\sqrt{T(K-1)}}{K}}^{0} 2\phi_r\left(\frac{T}{K}+j\right)
\end{aligned}$$

The integral on the very right is simply the integral of the terms that are not in the tail of the distribution as defined earlier. Thus it is equal to $1 - O\left(e^{-T}\right)$ and can be removed here. We also simplify the fraction by $\frac{T}{K^2}$ which yields the following.

$$\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] \;<\; O\left(e^{-T}\right) + \frac{K.Var(r)}{T - a^2\left(K-1\right)}$$

Here it is already clear that replay's variance is bigger than $\frac{K.Var(r)}{T}$ since we subtract something from the denominator. In what follows, let us make that cost clearer.

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] &< O\left(e^{-T}\right) + \frac{K.Var(r)}{T - a^2\left(K-1\right)} + \frac{K.Var(r)}{T} - \frac{K.Var(r)}{T} \\
&= O\left(e^{-T}\right) + \frac{K.Var(r)}{T} + \frac{KTVar(r) - K.Var(r)\left(T - a^2\left(K-1\right)\right)}{T^2 - Ta^2\left(K-1\right)} \\
&= O\left(e^{-T}\right) + \frac{K.Var(r)}{T} + \frac{K\left(K-1\right)a^2Var(r)}{T^2 - Ta^2\left(K-1\right)} \\
&= \frac{K.Var(r)}{T} + O\left(\frac{K^2Var(r)}{T^2}\right) .
\end{aligned}$$

Also, because the result is asymptotic in T in this proof, we can write that

$$\mathbb{E}\left[\left(\hat{g}_\pi^{(replay)} - g_\pi\right)^2\right] < 1\frac{K.Var(r)}{T} + O\left(\frac{1}{T^2}\right) ,$$

which concludes the proof. □

Appendix G

Proofs of the theorems of chapter 5

Contents

G.1 Consistency and convergence of BRED

This is the proof of theorem 13 from section 5.4.

Proof. At each iteration of the bootstrap loop (indexed by b), BRED is estimating $g_A(T)$ using the replay method on a dataset of size $T' = K \times T$. As the actions in S were chosen uniformly at random, we have $\mathbb{E}(T^{(b)}) = T'/K = T$.

As the policy is fixed, we can use the multiplicative Chernoff bound as in Li *et al.* [11] to obtain for any bootstrap step b:

$$\mathbb{P}r\left(\left|T^{(b)} - T\right| \geq \gamma_1 T\right) \leq \exp\left(-\frac{T\gamma_1^2}{3}\right) ,$$

for any $\gamma_1 > 0$ (where $\mathbb{P}r(e)$ denotes the probability of event e). A similar inequality can be obtained with $\mathbb{E}(\widehat{G}_A) = Tg_A$:

$$\mathbb{P}r\left(\left|\widehat{G}_A - Tg_A\right| \geq \gamma_2 Tg_A\right) \leq \exp\left(-\frac{Tg_A\gamma_2^2}{3}\right) .$$

Thus with $\gamma_1 = \sqrt{\frac{3}{T}\ln\frac{4}{\delta}}$ and $\gamma_2 = \sqrt{\frac{3}{Tg_A}\ln\frac{4}{\delta}}$ using a union bound over probabilities, we have with probability at least $1-\delta$:

$$1 - \gamma_1 \leq \frac{T^{(b)}}{T} \leq 1 + \gamma_1$$

and

$$g_A 1 - \gamma_2 \leq \frac{\widehat{G}_A}{T} \leq g_A 1 + \gamma_2 ,$$

which implies:

$$\left|\frac{\widehat{G}_A}{T^{(b)}} - g_A\right| \leq \frac{(\gamma_1+\gamma_2)g_A}{1-\gamma_1} = O\left(\sqrt{\frac{g_a}{T}}\ln\frac{1}{\delta}\right) .$$

So with high probability the first moment of $\widehat{CTR}_A(T^{(b)})$ as estimated by the replay method admits an asymptotic expansion in $1/\sqrt{\mathbb{E}(T^{(b)})} = 1/T$.

Now we need to focus on higher order terms. All the moments are finite because the reward distribution over $\mathcal{S}$ is bounded. Recall that by hypothesis $\xi\left(CTR_A(T)\right)$ admits a term of order α:

$$p_\alpha = \mathop{\mathbb{E}}_{D}\left(CTR_A(T^{(b)})^\alpha\right) .$$

The Chernoff's bound can be applied to $\left|(T^{(b)})^\alpha - T^\alpha\right|$ and $\left|\widehat{G}_A^\alpha - T^\alpha g_A^\alpha\right|$ leading to:

$$\left|\frac{\widehat{G}_A^\alpha}{(T^{(b)})^\alpha} - g_A^\alpha\right| = O\left(\left(\frac{g_a}{T}\right)^{\frac{\alpha}{2}}\ln\frac{1}{\delta}\right) ,$$

with probability at least $1-\delta$.

So for a large enough $T^{(b)}$, $\xi\left(\widehat{CTR}_A(T^{(b)})\right)$ admits a expansion in polynomials of $1/\sqrt{T}$. Thus theorem 1 applies and the aggregation of all the $\hat{g}_A^{(b)}(T^{(b)})$ allows an estimation of $CTR_A(T)$. For a large enough number of bootstrap iterations (the value of B in BRED), we obtain a convergence speed in $O(1/T)$ with high probability, which concludes the proof. □

G.2 Unbiasedness of replay* with a non uniform logging policy

This is the proof of theorem 17 from section 5.8.2.

Proof.

$$\begin{aligned}
\mathbb{E}\left[\hat{g}_\pi^{(replay*)}\right] &= \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\frac{V_t r_t}{p\left(\pi_{LOG}\left(x_t\right)=a_t\right)}\right] \\
&= \frac{1}{T}T \mathop{\mathbb{E}}_{(x,a,r)\sim(\mathcal{C},\pi_{LOG}(x),R_{x,a})}\left[\frac{I\left(\pi\left(x\right)=a\right)\ .\ r}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right] \\
&= \sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}p(\pi_{LOG}(x)=a)\mathop{\mathbb{E}}_{r\sim R_{x,a}}\left[\frac{I\left(\pi\left(x\right)=a\right)\ .\ \vec{r}[\pi(x)]}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right] \\
&= \sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}\frac{p(\pi_{LOG}(x)=a)}{p(\pi_{LOG}(x)=a)}\mathop{\mathbb{E}}_{r\sim R_{x,a}}[r]\,.\,\mathbb{E}\left[I\left(\pi\left(x\right)=a\right)\right] \\
&= \sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}p\left(\pi\left(x\right)=a\right)\mathop{\mathbb{E}}_{r\sim R_{x,a}}[r] \\
&= g_\pi.
\end{aligned}$$

The second line uses the fact that all T iterations are independent (π and π_{LOG} are stationary). The third line comes the law of total expectations. The fourth uses the independence between $\pi(x)$ and r ($\sim R_{x,a}$) given x. Indeed π only depends on x, which is fixed and can therefore not be influenced by r (or anything else for that matter). Reciprocally, r depends on x which is fixed and $a = \pi_{LOG}(x)$ which also only depends on x. Finally the expression simplifies and yields g_π which concludes the proof of unbiasedness of the estimator. □

Notice that RED$^*\infty$ is the mean of E independent realizations of replay* (given the data). It is thus unbiased as well.

G.3 Variance of replay* with a non uniform logging policy

This is the proof of theorem 19 from section 5.8.2.

Proof.

$$\begin{aligned}
Var\left(\hat{g}_\pi^{(replay^*)}\right) &= Var\left(\frac{1}{T}\sum_{t=1}^{T}\frac{V_t r_t}{p\left(\pi_{LOG}\left(x_t\right)=a_t\right)}\right)\\
&= \frac{1}{T^2}T \underset{(x,a,r)\sim(\mathcal{C},\pi_{LOG}(x),R_{x,a})}{Var}\left(\frac{I\left(\pi\left(x\right)=a\right)\ .\ r}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right)\\
&= \frac{1}{T}\left(\underset{\substack{x\sim\mathcal{C}\\ a\sim\pi_{LOG}(x)\\ r\sim R_{x,a}}}{\mathbb{E}}\left[\left(\frac{I\left(\pi\left(x\right)=a\right)\ .\ r}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right)^2\right]-\underset{\substack{x\sim\mathcal{C}\\ a\sim\pi_{LOG}(x)\\ r\sim R_{x,a}}}{\mathbb{E}}\left[\frac{I\left(\pi\left(x\right)=a\right)\ .\ r}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right]^2\right)\\
&= \frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}p\left(\pi_{LOG}(x)=a\right)\underset{r\sim R_{x,a}}{\mathbb{E}}\left[\frac{I\left(\pi\left(x\right)=a\right)\ .\ r^2}{p\left(\pi_{LOG}\left(x\right)=a\right)^2}\right]-g_\pi^2\right)\\
&= \frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}\frac{p\left(\pi_{LOG}(x)=a\right)}{p\left(\pi_{LOG}\left(x\right)=a\right)^2}\mathbb{E}\left[I\left(\pi\left(x\right)=a\right)\right]\underset{r\sim R_{x,a}}{\mathbb{E}}\left[r^2\right]-g_\pi^2\right)\\
&= \frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}\frac{p\left(\pi(x)=a\right)}{p\left(\pi_{LOG}\left(x\right)=a\right)}\underset{r\sim R_{x,a}}{\mathbb{E}}\left[r^2\right]-g_\pi^2\right)\ .
\end{aligned}$$

The second line uses the fact that all T iterations are independent (π and π_{LOG} are stationary). The right hand side term of the third line is the squared expectation $\hat{g}_\pi^{(replay*)}$ when $T=1$. It is thus equal to g_π^2. Notice in the fourth line that the indicator function is not squared. Indeed, it is either equal to 0 or to 1 and is thus equal to its square. The fifth line uses the independence between $\pi(x)$ and $r\ (\sim R_{x,a})$ given x (same arguments as in the proof of theorem 17). Finally the expression simplifies to the expected result which concludes the proof. $\square$

G.4 Variance of RED* with a non uniform logging policy

This is the proof of theorem 20 from section 5.8.2.

Proof. In this proof, we use the compressed form of RED$^*\infty$ that was shown in section 5.7.7. Then everything unfolds the same way as in the proof of the variance of replay* (theorem 19, proved in appendix G.3).

$$\begin{aligned}
Var\left(\hat{g}_\pi^{(RED^*\infty)}\right) &= Var\left(\frac{1}{T}\sum_{t=1}^{T}\frac{p\left(\pi\left(x_t\right)=a_t\right)r_t}{p\left(\pi_{LOG}\left(x_t\right)=a_t\right)}\right)\\
&= \frac{1}{T^2}T\underset{(x,a,r)\sim(\mathcal{C},\pi_{LOG}(x),R_{x,a})}{Var}\left(\frac{p\left(\pi\left(x\right)=a\right)\ .\ r}{p\left(\pi_{LOG}\left(x\right)=a\right)}\right)\\
&= \frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}p\left(\pi_{LOG}\left(x\right)=a\right)\underset{r\sim R_{x,a}}{\mathbb{E}}\left[\frac{r^2.p\left(\pi\left(x\right)=a\right)^2}{p\left(\pi_{LOG}\left(x\right)=a\right)^2}\right]-g_\pi^2\right)\\
&= \frac{1}{T}\left(\sum_{x\in\mathcal{X}}p(x)\sum_{a=1}^{K}\frac{p\left(\pi\left(x\right)=a\right)^2}{p\left(\pi_{LOG}\left(x\right)=a\right)}\underset{r\sim R_{x,a}}{\mathbb{E}}\left[r^2\right]-g_\pi^2\right)\ .
\end{aligned}$$

□

G.5 Convexity of the problem to find the minimum variance logging policy

This is the proof of theorem 21 from section 5.8.3

Proof. We use the following notations:

$$\begin{aligned}
p_\pi &= p(\pi=1), & x &= p(\pi_{LOG}=1)\\
a_{replay*} &= p_\pi.\mu_1, & b_{replay*} &= (1-p_\pi)\mu_2\\
a_{RED*\infty} &= p_\pi^2.\mu_1 & b_{RED*\infty} &= (1-p_\pi)^2\mu_2
\end{aligned}$$

The proof is the same for replay* and RED*∞. Only the values for a and b change. Therefore the proof is proposed using the notations a and b (without subscript) which can be replaced by the values corresponding to the estimator of our choice.

The variance of $\hat{g}_\pi$ evaluated by replay* or RED*∞ is expressed as follows:

$$Var(\hat{g}_\pi)=\frac{1}{T}\left(\frac{a}{x}+\frac{b}{1-x}-g_\pi^2\right).$$

When π is fixed the variance varies with

$$v(x)=\frac{a}{x}+\frac{b}{1-x},$$

which is the function that we shall study. The goal here is to prove that $v(x)$ is convex on $]0,1[$, the domain of definition of x, the logging policy. We then want to find $p_{min}\in]0,1[$, the value of x that minimizes $v(x)$.

We first calculate the first and second order derivative of v.

$$\begin{aligned}
v'(x) &= \frac{-a}{x^2}+\frac{b}{(1-x)^2},\\
v''(x) &= \frac{2a}{x^3}+\frac{-2b}{(x-1)^3}.
\end{aligned}$$

Because $0<x<1$, $a>0$ and $b>0$, it is trivial that $v''(x)$ is a sum of two positive terms (the second term is a ratio of two negative terms) and so we have $v''(x)>0$ on $]0,1[$. As a consequence, v' is strictly increasing and v is a convex function of x in the domain $]0,1[$ we are interested in, which proves the first part of the theorem.

Now we study the sign of $v'(x)$ between 0 and 1. We have that

$$\begin{aligned} v'(x) &= \frac{-a}{x^2} + \frac{b}{(1-x)^2}, \\ v'(x) &= \frac{(b-a)x^2 + 2ax - a}{x^2(1-x)^2}. \end{aligned}$$

Since the denominator is positive, $v'(x)$ has the sign of:

$$w(x) = (b-a)x^2 + 2ax - a.$$

If $a = b$, $w(x) = 2ax - a$ which is an increasing function that nullifies in $x = \frac{1}{2}$. Therefore we have the minimum we sought:

x	0		$\frac{1}{2}$		1
$w(x)$	$\Vert$	$-$	0	$+$	$\Vert$
$v(x)$	$\Vert$	$\searrow$	min	$\nearrow$	$\Vert$

Otherwise ($a \neq b$) w is a quadratic polynomial in x of discriminant $\Delta = 4ab$ and with an extremum $x_e = \frac{a}{a-b}$. Its roots are:

$$x_1 = \frac{-a-\sqrt{ab}}{b-a} = \frac{-1-\sqrt{\frac{b}{a}}}{\frac{b}{a}-1} = \frac{1+\sqrt{\frac{b}{a}}}{1-\frac{b}{a}}, \qquad x_2 = \frac{-a+\sqrt{ab}}{b-a} = \frac{\sqrt{\frac{b}{a}}-1}{\frac{b}{a}-1} = \frac{1-\sqrt{\frac{b}{a}}}{1-\frac{b}{a}}.$$

We then consider the two cases $a < b$ and $a > b$ and determine what happens in $]0,1[$.

<u>If $a < b$,</u> w's leading term $(b-a)$ is positive. Therefore $w(x)$ is first positive (until x_1) then negative (until x_2) and finally positive again.

Also we have that:

$$\frac{b}{a} > \sqrt{\frac{b}{a}} > 1.$$

Consequently:

$$\frac{b}{a} > 1 \quad\Rightarrow\quad 1 - \frac{b}{a} < 0 \quad\Rightarrow\quad x_1 \frac{1+\sqrt{\frac{b}{a}}}{1-\frac{b}{a}} < 0\,,$$

$$\frac{b}{a} > \sqrt{\frac{b}{a}} > 1 \quad\Rightarrow\quad \frac{b}{a} - 1 > \sqrt{\frac{b}{a}} - 1 > 0 \quad\Rightarrow\quad 0 < x_2 = \frac{\sqrt{\frac{b}{a}}-1}{\frac{b}{a}-1} < 1\,,$$

This yields the following (with $c = \frac{b}{a}$):

x	$-\infty$		$x_1 = \frac{1+\sqrt{c}}{1-c}$		0		$x_2 = \frac{1-\sqrt{c}}{1-c}$		1		$+\infty$
$w(x)$		$+$	0	$-$		$-$	0	$+$		$+$	
$v(x)$		$\nearrow$		$\searrow$		$\searrow$	min	$\nearrow$		$\nearrow$	

If $a > b$, w's leading term $(b-a)$ is negative. Therefore $w(x)$ is first negative (until x_2) then positive (until x_1) and finally negative again.

Also we have that:

$$0 < \frac{b}{a} < \sqrt{\frac{b}{a}} < 1.$$

Consequently:

$$1 + \sqrt{\frac{b}{a}} > 1 \ \ \textit{and} \ \ 0 < 1 - \frac{b}{a} < 1 \quad \Rightarrow \quad x_1 = \frac{1+\sqrt{\frac{b}{a}}}{1-\frac{b}{a}} > 1 \ ,$$

$$0 < \frac{b}{a} < \sqrt{\frac{b}{a}} < 1 \quad \Rightarrow \quad 0 < 1 - \sqrt{\frac{b}{a}} < 1 - \frac{b}{a} < 1 \quad \Rightarrow \quad 0 < x_2 = \frac{1-\sqrt{\frac{b}{a}}}{1-\frac{b}{a}} < 1 \ .$$

This yields the following (with $c = \frac{b}{a}$):

x	$-\infty$		0		$x_2 = \frac{1-\sqrt{c}}{1-c}$		1		$x_1 = \frac{1+\sqrt{c}}{1-c}$		$+\infty$
$w(x)$		$-$		$-$	0	$+$		$+$	0	$-$	
$v(x)$		$\searrow$		$\searrow$	min	$\nearrow$		$\nearrow$		$\searrow$	

In both cases ($a < b$ and $a > b$), we find the same configuration in $]0,1[$ with the same unique value for x: $\frac{1-\sqrt{c}}{1-c}$ that minimizes the variance of $\hat{g}_\pi$. This concludes the proof, which we recall is valid for both replay* and RED*∞.

$\square$

Bibliography

[1] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[2] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web (WWW-2010), Raleigh, NC, USA*, pages 661–670. ACM, 2010.

[3] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[4] Michael A. Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[5] Olivier Nicol, Jérémie Mary, and Philippe Preux. Icml exploration & exploitation challenge: Keep it simple! volume 26 of *JMLR Proceedings*, pages 62–85. JMLR.org/Microtome Publishing, 2012.

[6] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel*, pages 13–20. Omnipress, 2010.

[7] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web (WWW-2010), Raleigh, NC, USA*, pages 661–670. ACM, 2010.

[8] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[9] James Bennett and Stan Lanning. The Netflix prize. In *Proceedings of the KDD Cup Workshop 2007, San Jose, CA, USA (at ACM SIGKDD-2007)*, pages 3–6. ACM, 2007.

[10] John Langford, Alexander Strehl, and Jennifer Wortman. Exploration scavenging. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland*, volume 307 of *ACM International Conference Proceeding Series*, pages 528–535. ACM, 2008.

[11] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Proceedings of the 4th International Conference on Web*

Search and Web Data Mining (WSDM-2011), Hong Kong, China, pages 297–306. ACM, 2011.

[12] Yahoo! Researsh. R6b - yahoo! front page today module user click log dataset. `http://webscope.sandbox.yahoo.com/`, 2012.

[13] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, May 2002.

[14] Bradley Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, 7(1):1–26, 1979.

[15] B. W. Silverman and G. A. Young. The bootstrap: To smooth or not to smooth? *Biometrika*, 74(3):469–479, 1987.

[16] Blaise Pascal. *Pensées*. Dezobry et E. Magdeleine, 1852. (1st edition in 1670).

[17] Daniel Bernoulli. Exposition of a new theory on the measurement of risk. *Econometrica: Journal of the Econometric Society*, pages 23–36, 1954. (1st edition in 1738).

[18] Abraham Wald. Contributions to the theory of statistical estimation and testing hypotheses. *Annals of Mathematical Statistics*, 10(4):299–326, 1939.

[19] Warren B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2007.

[20] Drazen Prelec and George Loewenstein. Decision making over time and under uncertainty: A common approach. *Management science*, 37(7):770–786, 1991.

[21] Helen M. Regan, Yakov Ben-Haim, Bill Langford, William G. Wilson, Per Lundberg, Sandy J. Andelman, and Mark A. Burgman. Robust decision-making under severe uncertainty for conservation management. *Ecological Applications*, 15(4):1471–1477, 2005.

[22] Chih-Chun Wang, S. R. Kulkarni, and H. V. Poor. Bandit problems with side observations. *IEEE Transactions on Automatic Control*, 50(3):338–355, 2005.

[23] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS-2007), Vancouver, British Columbia, Canada*. Curran Associates, Inc., 2008.

[24] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 1 edition, September 2010.

[25] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM (CACM)*, 35(12):61–70, 1992.

[26] The Internet Society. History of the internet society. `http://www.internetsociety.org/history`, 2014.

[27] Internet Systems Consortium. Isc domain survey. `http://tools.ietf.org/html/rfc1296`, January 1992.

[28] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW-94), Chapel Hill, North Carolina, USA*, pages 175–186. ACM, 1994.

[29] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[30] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[31] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[32] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system - a case study. In *Workshop on Web Mining for E-Commerce – Challenges and Opportunities (WEBKDD-2000), Boston, MA, USA (at ACM SIGKDD-2000) - without proceedings, see `http://robotics.stanford.edu/~ronnyk/WEBKDD2000`*. ACM, 2000.

[33] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In Wendy A. Kellogg and Steve Whittaker, editors, *Proceeding of the ACM 2000 Conference on Computer Supported Cooperative Work (CSCW-2000), Philadelphia, PA, USA*, pages 241–250. ACM, 2000.

[34] David McSherry. Explanation in recommender systems. *Artificial Intelligence Review*, 24(2):179–197, 2005.

[35] Wei Zeng, Ming-Sheng Shang, and Tie-Yun Qian. Useful acquiring ratings for collaborative filtering. In *2009 IEEE Youth Conference on Information, Computing and Telecommunication (YC-ICT-2009), Beijing, China*, pages 483–486. IEEE Computer Society, 2009.

[36] William T. Glaser, Timothy B. Westergren, Jeffrey P. Stearns, Jonathan M. Kraft, et al. Consumer item matching method and system, February, 21 2006. US Patent 7,003,515 - (Pandora).

[37] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5–6):393–408, 1999.

[38] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. Feature weighting in content based recommendation system using social network analysis. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th international conference on World Wide Web (WWW-2008), Beijing, China*, pages 1041–1042. ACM, 2008.

[39] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, pages 325–341. Springer, 2007.

[40] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[41] George Lekakos and Petros Caravelas. A hybrid approach for movie recommendation. *Multimedia tools and applications*, 36(1-2):55–70, 2008.

[42] comScore. Cookie-based counting overstates size of web site audiences. *PR Newswire* `http://www.prnewswire.com/news-releases/cookie-based-counting-overstates-size-of-web-site-audiences-58362982.html`, April 2007.

[43] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning Series. MIT Press, 1998.

[44] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[45] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.

[46] T. L. Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[47] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[48] Apostolos N. Burnetas and Michael N. Katehakis. Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17(2):122–142, 1996.

[49] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, April 2009.

[50] Sarah Filippi, Olivier Cappé, and Aurélien Garivier. Optimism in reinforcement learning and kullback-leibler divergence. In *48th Annual Allerton Conference on Communication, Control, and Computing (ALLERTON-2010), Monticello, Illinois, USA*, pages 115–122. IEEE, 2010.

[51] Aurélien Garivier and Olivier Cappé. The KL-UCB algorithm for bounded stochastic bandits and beyond. In Sham M. Kakade and Ulrike von Luxburg, editors, *The 24th Annual Conference on Learning Theory (COLT-2011), Budapest, Hungary*, volume 19 of *JMLR Proceedings*, pages 359–376. JMLR.org, 2011.

[52] Olivier Cappé, Aurélien Garivier, Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. Kullback–leibler upper confidence bounds for optimal sequential allocation. *Annals of Statistics*, 41(3):1516–1541, 2013.

[53] Steffen Grünewälder, Jean-Yves Audibert, Manfred Opper, and John Shawe-Taylor. Regret bounds for gaussian process bandit problems. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS-2010), Chia Laguna Resort, Sardinia, Italy*, volume 9 of *JMLR Proceedings*, pages 273–280. JMLR.org/Microtome Publishing, 2010.

[54] John C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–177, 1979.

[55] Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson Sampling: An Asymptotically Optimal Finite Time Analysis. In Nader H. Bshouty, Gilles Stoltz, Nicolas Vayatis, and Thomas Zeugmann, editors, *Algorithmic Learning Theory - Proceedings of the 23rd International Conference (ALT-2012), Lyon, France*, volume LNCS 7568, pages 199–213. Springer, 2012.

[56] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS-2011), Granada, Spain.*, pages 2249–2257. Curran Associates, Inc., 2011.

[57] Steven L. Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[58] A.G. Barto and P. Anandan. Pattern-recognizing stochastic learning automata. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-15(3):360–375, May 1985.

[59] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. Business Data Processing: A Wiley Series. John Wiley & Sons, 3rd edition, 2011. (1st edition in 1979).

[60] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[61] Judit Bar-Ilan. Google bombing from a time perspective. *Journal of Computer-Mediated Communication*, 12(3):910–938, 2007.

[62] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web (WWW-2004), New York, NY, USA*, pages 393–402. ACM, 2004.

[63] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Effective attack models for shilling item-based collaborative filtering systems. In Olfa Nasraoui, Osmar R. Zaïane, Myra Spiliopoulou, Bamshad Mobasher, Brij M. Masand, and Philip S. Yu, editors, *Proceedings of the 7th International Workshop on Knowledge Discovery on the Web (WebKDD-2005), Chicago, IL, USA (at ACM SIGKDD-2005)*, volume 4198 of *Lecture Notes in Computer Science*, pages 15–23. Springer, 2006.

[64] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)*, 7(4):23, 2007.

[65] Bamshad Mobasher, Robin Burke, Chad Williams, and Runa Bhaumik. Analysis and detection of segment-focused attacks against collaborative recommendation. In Olfa Nasraoui, Osmar R. Zaïane, Myra Spiliopoulou, Bamshad Mobasher, Brij M. Masand, and Philip S. Yu, editors, *Proceedings of the 7th International Workshop on Knowledge Discovery on the Web (WebKDD-2005), Chicago, IL, USA (at ACM SIGKDD-2005)*, volume 4198 of *Lecture Notes in Computer Science*, pages 96–118. Springer, 2006.

[66] Sheng Zhang, Yi Ouyang, James Ford, and Fillia Makedon. Analysis of a low-dimensional linear model under recommendation attacks. In Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors, *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2006), Seattle, Washington, USA*, pages 517–524. ACM, 2006.

[67] Paolo Massa and Paolo Avesani. Trust metrics on controversial users: Balancing between tyranny of the majority. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(1):39–64, 2007.

[68] Runa Bhaumik, Robin D. Burke, and Bamshad Mobasher. Crawling attacks against web-based recommender systems. In Robert Stahlbock, Sven F. Crone, and Stefan Lessmann, editors, *Proceedings of the 2007 International Conference on Data Mining (DMIN-2007), Las Vegas, Nevada, USA*, pages 183–189. CSREA Press, 2007.

[69] Elazar J. Pedhazur and Liora Pedhazur Schmelkin. *Measurement, design, and analysis: An integrated approach.* Lawrence Erlbaum Associates, 1991.

[70] Matthew B. Miles and A. Michael Huberman. *Qualitative data analysis: An expanded sourcebook.* SAGE, 2nd edition, 1994. (1st edition in 1984).

[71] John W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* SAGE, 4th edition, 2009. (1st edition in 1994).

[72] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M. Henne. Controlled experiments on the web: Survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1):140–181, 2009.

[73] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-1995), Montreal, Quebec, Canada*, volume 14, pages 1137–1145. Morgan Kaufmann, 1995.

[74] Steven L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and knowledge discovery*, 1(3):317–328, 1997.

[75] Michael A. Babyak. What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models. *Psychosomatic medicine*, 66(3):411–421, 2004.

[76] Tariq Mahmood and Francesco Ricci. Learning and adaptivity in interactive recommender systems. In Maria L. Gini, Robert J. Kauffman, Donna Sarppo, Chrysanthos Dellarocas, and Frank Dignum, editors, *Proceedings of the 9th international conference on Electronic commerce (ICEC-2007), Minneapolis, MN, USA*, volume 258 of *ACM International Conference Proceeding Series*, pages 75–84. ACM, 2007.

[77] Gerhard Fischer. User modeling in human–computer interaction. *User Modeling and User-Adapted Interaction (UMUAI)*, 11(1–2):65–86, 2001.

[78] Yi Zhang and Jonathan Koren. Efficient bayesian hierarchical user modeling for recommendation system. In Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando, editors, *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2007), Amsterdam, The Netherlands*, pages 47–54. ACM, 2007.

[79] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research (JMLR)*, 10:2935–2962, 2009.

[80] Giuseppe Carenini and Rita Sharma. Exploring more realistic evaluation measures for collaborative filtering. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the 19th National Conference on Artificial Intelligence, 16th Conference on Innovative Applications of Artificial Intelligence (AAAI-2004), San Jose, California, USA*, pages 749–754. AAAI Press / The MIT Press, 2004.

[81] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In Gary M. Olson and Robin Jeffries, editors, *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems (CHI-2006), Montréal, Québec, Canada*, pages 1097–1101. ACM, 2006.

[82] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating “word of mouth”. In Irvin R. Katz, Robert L. Mack, Linn Marks, Mary Beth Rosson, and Jakob Nielsen, editors, *Proceedings of the 2005 Conference on Human Factors in Computing Systems (CHI-1995), Denver, Colorado, USA*, pages 210–217. ACM/Addison-Wesley, 1995.

[83] S. Girgin, J. Mary, Ph. Preux, and O. Nicol. Advertising campaigns management: Should we be greedy? In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *The 10th IEEE International Conference on Data Mining (ICDM-2010), Sydney, Australia*, pages 821–826. IEEE Computer Society, 2010.

[84] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces (IUI-2001), Santa Fe, NM, USA*, pages 33–40. ACM, 2001.

[85] Hua Zheng, Dong Wang, Qi Zhang, Hang Li, and Tinghao Yang. Do clicks measure recommendation relevancy?: An empirical user study. In *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys-2010), Barcelona, Spain*, pages 249–252. ACM, 2010.

[86] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{tm}}$: A bayesian skill rating system. In B Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS-2006), Vancouver, British Columbia, Canada*, pages 569–576. MIT Press, 2007.

[87] Xavier Amatriain and Justin Basilico. Netflix recommendations: Beyond the 5 stars (part 1). *The Netflix Tech Blog* - `http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html`, April 2012.

[88] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[89] Frank R. Kschischang, Brendan J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[90] Geoffrey J. Mc Lachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley series in Probability and Statistics. John Wiley & Sons, 1$^{\text{st}}$ edition, 1997.

[91] Christophe Salperwyck and Tanguy Urvoy. Stumping along a Summary for Exploration & Exploitation Challenge 2011. volume 26 of *JMLR Proceedings*, pages 86–97. JMLR.org/Microtome Publishing, 2012.

[92] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix. *The Annals of Mathematical Statistics*, 20(4):620–624, 12 1949.

[93] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 03 1950.

[94] Ye Chen, Pavel Berkhin, Bo Anderson, and Nikhil R Devanur. Real-time bidding algorithms for performance-based display ad allocation. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1307–1315. ACM, 2011.

[95] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1 edition, 1957.

[96] Bennett L. Fox. Finite horizon behavior of policies for two-arm bandits. *Journal of the American Statistical Association*, 69(348):963–965, 1974.

[97] Alexander L. Strehl, John Langford, Lihong Li, and Sham Kakade. Learning from logged implicit exploration data. pages 2217–2225, 2010.

[98] Léon Bottou, Jonas Peters, Joaquin Qui nonero Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research (JMLR)*, 14(1):3207–3260, 2013.

[99] Alina Beygelzimer and John Langford. The offset tree for learning with partial labels. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2009), Paris, France*, pages 129–138. ACM, 2009.

[100] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. pages 1097–1104, 2011.

[101] Francis Maes, Louis Wehenkel, and Damien Ernst. Automatic discovery of ranking formulas for playing with multi-armed bandits. In Scott Sanner and Marcus Hutter, editors, *Recent Advances in Reinforcement Learning - 9th European Workshop (EWRL-2011), Athens, Greece*, volume 7188 of *Lecture Notes in Computer Science*, pages 5–17. Springer, 2012.

[102] Przemysław Kazienko and Michał Adamski. Adrosa—adaptive personalization of web advertising. *Information Sciences*, 177(11):2269–2295, 2007.

[103] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, Nitin Motgi, Seung-Taek Park, Raghu Ramakrishnan, Scott Roy, and Joe Zachariah. Online models for content optimization. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS-2008), Vancouver, British Columbia, Canada*, pages 17–24. Curran Associates, Inc., 2009.

[104] José Antonio Martín H and Ana M. Vargas. Linear bayes policy for learning in contextual-bandits. *Expert Systems with Applications*, 40(18):7400–7406, 2013.

[105] Jérémie Mary and Olivier Nicol. Challenge exploration and exploitation 3. *INRIA website* `https://explochallenge.inria.fr/`, 2012.

[106] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[107] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Information Science and Statistics. Springer, 2008.

[108] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI-98 workshop on learning for text categorization, Madison, Wisconsin, USA*, pages 41–48. AAAI Press, 1998.

[109] Bradley Efron and Robert Tibshirani. *An introduction to the bootstrap*, volume 57 of *Monographs on Statistics and Applied Probability.* Springer, 1993.

[110] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. Adaptive bootstrapping of recommender systems using decision trees. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Proceedings of the 4th International Conference on Web Search and Web Data Mining (WSDM-2011), Hong Kong, China*, pages 595–604. ACM, 2011.

[111] Maurice H. Quenouille. Problems in plane sampling. *The Annals of Mathematical Statistics*, 20(3):355–375, 1949.

[112] Maurice H. Quenouille. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956.

[113] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. The big data bootstrap. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12), Edinburgh, Scotland, GB*, pages 1759–1766. Omnipress, 2012.

[114] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. Spatio-temporal models for estimating click-through rate. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 18th international conference on World wide web (WWW-2010), Madrid, Spain*, pages 21–30. ACM, 2009.

[115] Herman J. Ader, Gideon J. Mellenbergh, and David J. Hand. *Advising on Research Methods: a consultant's companion.* Johannes van Kessel Publishing, 2008.

[116] Q. Li and Suojin Wang. A simple consistent bootstrap test for a parametric regression function. *Journal of Econometrics*, 87(1):145–165, 1998.

[117] Bradley Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[118] Bradley Efron and Robert Tibshirani. Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.

[119] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[120] Roberto Esposito and Lorenza Saitta. Monte carlo theory as an explanation of bagging and boosting. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), Acapulco, Mexico*, pages 499–504. Morgan Kaufmann, 2003.

[121] Joel L. Horowitz. The bootstrap. *Handbook of econometrics*, 5:3159–3228, 2001.

[122] Petri Koistinen and Lasse Holmström. Kernel regression and backpropagation training with noise. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4 (NIPS-1991), Denver, Colorado, USA*, pages 1033–1039. Morgan Kaufmann, 1992.

[123] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Incorporated, 1995.

[124] Peter Hall. The bootstrap and edgeworth expansion. 1992.

[125] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[126] O. Nicol, J. Mary, and P. Preux. Improving offline evaluation of contextual bandit algorithms via bootstrapping techniques. In *Proceedings of the 31th International Conference on Machine Learning (ICML-2014), Beijing, China*, volume 32 of *JMLR Proceedings*, pages 172–180. JMLR.org, 2014.

[127] Yvonne M. M. Bishop, Stephen E. Fienberg, and Paul W. Holland. *Discrete multivariate analysis: theory and practice.* Theory and Applications. Springer, 2007.

[128] Peter J. Bickel and David A. Freedman. Some asymptotic theory for the bootstrap. *The Annals of Statistics*, 9(6):1196–1217, 1981.

[129] Evarist Giné and Joel Zinn. Bootstrapping general empirical measures. *The Annals of Probability*, 18(2):851–869, 1990.

[130] Rabi N. Bhattacharya and Jayanta K. Ghosh. On the validity of the formal edgeworth expansion. *Annals of Statistics*, 6(2):434–451, 1978.

[131] P. J. Bickel, F. Götze, and W. R. van Zwet. Resampling fewer than n observations: gains, losses, and remedies for losses. *Statistica Sinica*, 7(1):1–32, 1997.

[132] Erich Leo Lehmann and George Casella. *Theory of point estimation*, volume 31 of *Texts in Applied Mathematics.* Springer, 2nd edition, 1998. (1st edition in 1983).

[133] Vassiliy A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158, 1969.

[134] M. P. Wand and M. C. Jones. *Kernel smoothing*, volume 60 of *Monographs on Statistics and Applied Probability.* Springer, 1995.

[135] Bernard W. Silverman. *Density estimation for statistics and data analysis*, volume 26 of *Monographs on Statistics and Applied Probability.* Springer, 1986.

[136] M. Chris Jones, James S. Marron, and Simon J. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996.

[137] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* Wiley Series in Probability and Statistics. John Wiley & Sons, 1992.

[138] David Edgar Liebke. Bootstrapping: estimating confidence intervals, standard errors, and bias. *Data sorcery blog* `http://data-sorcery.org/category/bootstrapping/`, July 2009.

[139] Peter M. Robinson. Nonparametric estimators for time series. *Journal of Time Series Analysis*, 4(3):185–207, 1983.

[140] Simon J. Sheather and Michael C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society, Series B*, 53(3):683–690, 1991.

[141] Adrian W. Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71(2):353–360, 1984.

[142] Peter Hall, J. S. Marron, and Byeong U. Park. Smoothed cross-validation. *Probability Theory and Related Fields*, 92(1):1–20, 1992.

[143] J Langford, L Li, and A Strehl. Vowpal wabbit online learning project, 2007.

[144] James A Hanley and Brenda MacGibbon. Creating non-parametric bootstrap samples using poisson frequencies. *computer methods and programs in biomedicine*, 83(1):57–62, 2006.

[145] Amir Sani, Alessandro Lazaric, and Rémi Munos. Risk-aversion in multi-armed bandits. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-2012), Lake Tahoe, Nevada, USA*, pages 3284–3292. Curran Associates, Inc., 2012.

[146] Nicolas Galichet, Michèle Sebag, and Olivier Teytaud. Exploration vs exploitation vs safety: Risk-aware multi-armed bandits. 29:245–260, 2013.

[147] Peter Auer and Ronald Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

[148] Jean-Yves Audibert and Sébastien Bubeck. Regret bounds and minimax policies under partial monitoring. *The Journal of Machine Learning Research (JMLR)*, 11:2785–2836, 2010.

[149] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 22(140):1–55, 1932.

[150] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI-2003), Ft. Lauderdale, Florida, USA*, pages 585–592. ACM, 2003.

[151] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The adaptive web, Methods and Strategies of Web Personalization*, Lecture Notes in Computer Science, pages 291–324. Springer, 2007.

[152] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-1999), Berkeley, CA, USA*, pages 230–237. ACM, 1999.

[153] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithm for collaborative filtering. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-1998), Madison, Wisconsin, USA*, pages 43–52. Morgan Kaufmann, 1998.

[154] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval*, 5(4):287–310, 2002.

[155] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the 10th International Conference on World Wide Web (WWW-2001), Hong Kong, China*, pages 285–295. ACM, 2001.

[156] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science and Technology (JASIST)*, 41(6):391–407, 1990.

[157] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis (SINUM)*, 2(2):205–224, 1965.

[158] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[159] Simon Funk. Netflix update: Try this at home. *Project Sifter* `http://sifter.org/~simon/journal/20061211.html`, December 2006.

[160] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*, pages 9–42. Cambridge University Press, 1998. (revised, oct 2012).

[161] Krzysztof C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

[162] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM-2007), Omaha, Nebraska, USA*, pages 43–52. IEEE Computer Society, 2007.

[163] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In Rudolf Fleischer and Jinhui Xu, editors, *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM-2008), Shanghai, China*, Lecture Notes in Computer Science, pages 337–348. Springer, 2008.

[164] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM-2008), Pisa, Italy*, pages 263–272. IEEE Computer Society, 2008.

[165] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning (ICML-2007), Corvallis, Oregon, USA*, volume 227 of *ACM International Conference Proceeding Series*, pages 791–798. ACM, 2007.

[166] Qiang Yang, Irwin King, Qing Li, Pearl Pu, and George Karypis, editors. *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys-2013), Hong Kong, China.* ACM, 2013.

[167] Charles Sutton and Quim Castella. Recommendation track of the 2012 international conference in machine learning (icml-2012). Session 8D `http://icml.cc/2012/whatson/`, 2012.

[168] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - Tampere, Finland*, pages 253–260. ACM, 2002.

[169] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication (ICUIMC-2008), Suwon, Korea*, pages 208–211. ACM, 2008.

[170] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In Irvin R. Katz, Robert L. Mack, Linn Marks, Mary Beth Rosson, and Jakob Nielsen, editors, *Proceedings of the 2005 Conference on Human Factors in Computing Systems (CHI-1995), Denver, Colorado, USA*, pages 194–201. ACM/Addison-Wesley, 1995.

[171] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web (WWW-2005), Chiba, Japan*, pages 22–32. ACM, 2005.

[172] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.