Efficient, Safe and Adaptive Learning from User Interactions

Rolf Jagerman

Efficient, Safe and Adaptive Learning from User Interactions

Rolf Mathias Jagerman

Efficient, Safe and Adaptive Learning from User Interactions

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Aula der Universiteit
op 9 december 2020, te 17:00 uur

door

Rolf Mathias Jagerman

geboren te Zoetermeer

Promotiecommissie

Promotor:		
	Prof. dr. M. de Rijke	Universiteit van Amsterdam
Co-promotor:		
	Dr. I. Markov	Universiteit van Amsterdam
Overige leden:		
	Dr. C. Eickhoff	Brown University
	Prof. dr. H. Haned	Universiteit van Amsterdam
	Prof. dr. E. Kanoulas	Universiteit van Amsterdam
	Dr. A. Swaminathan	Microsoft
	Prof. dr. M. Worring	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was supported by Netherlands Organisation for Scientific Research (NWO) under project number 612.001.551.

Printed by Off Page, Amsterdam

ISBN: 978-94-93197-30-5

Acknowledgements

Pursuing a doctoral degree is a journey. I departed on this journey four years ago as a computer science graduate with the heart of a software engineer: I had, and still have, a passion for building things. The book you are reading is the culmination of four years of hard work. I could not have completed this book by myself and am thankful to all who have provided their guidance and support over the past four years.

First and foremost I want to thank my supervisor Maarten de Rijke. You have given me the opportunity to pursue the ideas I found interesting and your ability to expertly guide the whole research process, from conception to experimentation to writing, has made me a better researcher. Thank you Maarten!

Second I would like to thank my co-promotor Ilya Markov. We wrote several papers together on various topics and I learned a lot during these times. You helped me better understand how to turn an idea into a paper. Thank you Ilya!

Next, I would like to thank all my commitee members: Adith, Carsten, Evangelos, Hinda and Marcel. You have read my thesis in more depth than pretty much anyone else on this planet. Thank you!

During my four years at ILPS I met so many people with whom I shared insightful discussions and many fun evenings. In an order determined by a python shuffle on a Wednesday afternoon: Antonios, Vera, Boris, Sam, Wouter, Dat, Mahsa, Georgios, Julien, Krisztian, Zhaochun, Petra, Yangjun, Ziming, Bob, Kaspar, Tom, Jie, Isaac, Mariya, Chuan, Svitlana, Hamid, Ali A, Olivier, Yifan, Daan, Xiaohui, Mozhdeh, Marzieh, Julia, Chang, Praveen, Trond, Marlies, David S, Sami, Ana, Dilek, Edgar, Shaojie, Sebastian, Fei, Christof, Nikos, Artem, Christophe, Maurits, Ke, Pengjie, Amir, Alexey, Mohammad, Hendrik, Ali V, Mostafa, Spyretta, Dan, Arianna, Hosein, Peilei, Anna, Jin, Maartje, Xinyi, Arezoo, Wanyu, Harrie, Maarten M, Ridho, Anne, Jiahuan, David G, Manos and Katya. Thank you all!

Experimentation cannot be done without great infrastructure and I want to thank Auke, Derk, Jeroen and everyone else at FEIOG for keeping the servers up and running.

I was fortunate enough to be able to do two internships during my PhD. Weize, Michael, Rama, Zhen, Shuguang, Sebastian and everyone else at Google, thank you for an amazing summer in Mountain View! Max, Christophe, José, Drew, Tresi, Ernie, Xiaodan, Xinwei, Ethan, Hans, Tim and everyone else at Apple, thank you for all the engineering in Boston, I had a great time!

To all my online friends and everyone in [LotS], you kept me sane during years of writing papers: Ynea, Kinsmir, Acha, Archo, Zero, Iccana, Ody, Bee, Shiro, Foon, Lem, Ethan, Deathray, Duncan, Renown and Grumpy. Thank you!

The road leading up to me starting my PhD was greatly influenced by my father, Evert, who has supported my passion for computer science since an early age. Together with my mother, Marly, and my sister, Eva, I could not have wished for a more supportive family, you have always been there for me. Thank you!

Lastly, I thank my girlfriend Trúc. Your unwavering support and love has been my source of strength during my PhD. I could not have done this without you by my side. Thank you!

Contents

1 Introduction

Information Retrieval (IR) systems such as search engines or recommender systems face a complex challenge: their goal is to select a handful of useful items out of potentially billions of candidate items in order to satisfy the information need, shopping need or entertainment need of a user. To tackle this challenge, highly complex systems have been built that can rank items by relevance, placing relevant items at the top of the ranked list. For example, in web search a user may issue a query and the search engine responds with a search engine result page, usually containing a list of ten documents sorted by their relevance with respect to the query.

Modern IR systems employ a *ranking function* to determine the order in which items are shown to a user. Ranking functions assign a real-valued score to each item by combining many features, and then sort the items by their respective scores to produce a ranking. For example, a web search engine may consider features such as link analysis (Pagerank [125]), query-document lexical overlap (BM25 [140]), and many more. Supervised Learning to Rank (LTR) approaches can learn a ranking function from *annotated datasets*: datasets where it is known which items are relevant or not with respect to a query, user profile, or context. Traditionally, research in this area has focussed largely on improving the ability to optimize specific ranking metrics [21, 174] or to improve generalization by utilizing more powerful and expressive underlying models such as decision trees [20] and deep neural networks [34].

Supervised LTR has been successfully used in many different IR settings and tasks. However, the limitations of relying on annotated datasets have become more apparent in recent years: First, annotated datasets are not necessarily aligned with user preferences and may not capture the true preferences of the user [80]. Second, in some settings, for example in personal search, they are unethical or impossible to create [16]. Third, creating annotated datasets can be expensive [28, 131]. Fourth, since annotated datasets are expensive they are typically not updated frequently and cannot capture changes in user's preferences over time. Because of this, practicioners and researchers have started looking at *user interaction data* as an alternative source of data with which ranking models can be trained. In contrast to professional annotations, user interactions (a) capture the true user preferences more accurately, (b) can be used in settings where professional annotations are unethical or impossible, (c) are cheap to collect, and (d) can be collected in real-time and track non-stationary user preferences over time.

One of the major challenges in using user interaction data is *bias*. Some of the more prominent types of bias that make user interactions difficult to use are position bias [32] and selection bias [121]. Position bias is a phenomenon where users pay more attention to top-ranked items and consequently those items will receive more clicks than equally relevant items placed at lower ranks. Selection bias deals with the phenomenon where users only interact with a subset of the ranked list, for example just the top-10. In both cases the ranker that was used to log the interaction data has a great influence on the collected interaction data.

Recent work has focused on either online learning [120, 184] or counterfactual learning [81, 172] to perform *unbiased* LTR from user interactions [72, 123]. These methods can remove specific kinds of bias from the user interaction data, but do so at a cost: Online learning methods deal with bias by performing interventions: changing what rankings are shown to the user, often by randomizing some of the results shown. Hence they consequently risk exposing the user to poor rankings. On the other hand, counterfactual learning methods do not perform interventions but instead deal with bias by re-weighing historical interaction data. Doing so introduces significant amounts of variance, which can lead to inefficient learning. Furthermore, the lack of interventions means that counterfactual learning methods may lack interaction data on potentially high-quality items because they may never have been shown to the user during data collection.

Both online and counterfactual learning methods aim to solve the same problem: unbiased learning with biased feedback. As we will see in Chapter 2, counterfactual learning methods can underperform *empirically* in several unbiased LTR scenarios when compared to online learning approaches. This is despite the strong theoretical guarantees that counterfactual learning methods have for unbiased LTR. Therefore, our findings suggest that counterfactual learning methods can be improved. We identify three areas in which improvements could be made: *efficiency*, *safety* and *adaptiveness*. In the research chapters that follow we explore each of these areas in more depth.

First, we contrast counterfactual and online learning approaches to understand the strengths and the weaknesses of either approach (Chapter 2). Second, we propose a way to improve the convergence rate for counterfactual Stochastic Gradient Descent (SGD) approaches (Chapter 3). Third, we study how counterfactual learning methods can perform interventions, similar to online learning approaches, without the risk of exposing poor rankings to the user (Chapter 4). Fourth, we look at the properties of counterfactual estimators in situations where user-preferences are non-stationary and may change over time and propose new counterfactual estimators that have theoretically and empirically better performance in non-stationary settings (Chapter 5). Finally, we look beyond clicks to more comprehensive user interaction signals as the main source of user interaction data in the context of cloud-based file storage systems (Chapter 6).

1.1 Research Outline and Questions

There are two families of algorithms for dealing with bias when learning from user interactions: *online* learning and *counterfactual* learning. Online learning algorithms

learn to rank by interactively optimizing a ranking model. They deal with biases by performing *interventions*, effectively changing the rankings that are displayed to the user. In contrast, counterfactual LTR approaches deal with bias by treating clicks as relevance indicators and employing a form of re-weighing. To better understand the relative strengths and weaknesses of either approach, a comparison that contrasts these methods across a number of biased LTR scenarios is necessary. This brings us to the first research question:

RQ1 How should LTR practicioners choose which method to apply from either counterfactual or online LTR methodologies?

To answer this research question we contrast state-of-the-art online and counterfactual methods under different biased conditions in Chapter 2. We find that counterfactual learning has many theoretical advantages over online learning but underperform empirically in many experimental settings, especially when there is a large amount of bias or a large amount of noise.

The findings from RQ1 suggest that high levels of noise and bias can make counterfactual learning inefficient due to the large variance that is introduced with the inverse propensity scores. Counterfactual learning with SGD in these settings is slow and takes long to converge to a solution. The second research questions addresses this inefficiency problem:

RQ2 Can counterfactual learning from user interactions be made more efficient?

In Chapter 3 we introduce a sample-based SGD approach for counterfactual learning that has provably better convergence rate than standard Inverse Propensity Scoring (IPS)-weighted SGD that is commonly used for counterfactual learning. Furthermore we empirically show that this learning approach outperforms IPS-weighted SGD in many scenarios.

Going back to RQ1, another important finding is that counterfactual learning methods may suffer from selection bias, where only the top-10 results are displayed, whereas online methods do not suffer. We hypothesize that interventions performed by online methods enable them to change the top-10 results and are able to collect user interactions on documents that counterfactual methods could not. Based on this, we ask ourselves the third research question:

RQ3 Can counterfactual approaches perform interventions without harming the user experience?

Chapter 4 answers this question by introducing a new counterfactual learning algorithm that periodically deploys its learned model online, but only does so when it is *safe* to do so, i.e. when there is no risk of harming the user experience. To accomplish this we build on existing high-confidence off-policy estimators that can tell us when the learned policy outperforms the deployed policy.

In contrast to annotated datasets, user interactions can be collected in real-time and track user preferences over time. As such, it is important to understand the behavior

of counterfactual estimators when user preferences change. This leads us to the fourth research question:

RQ4 How can counterfactual approaches be adapted to deal with non-stationary environments?

To answer this question we look at the bias and variance of counterfactual estimators in non-stationary environments in Chapter 5. We find that counterfactual estimators can be biased when user preferences change over time and propose two estimators that have reduced bias in non-stationary settings.

In this thesis we largely focus on clicks on search results as the main source of user interaction data. However, there are settings where other types of interaction data are more abundant and where search logs are a limited resource. Other interaction data is not necessarily tied to relevance or ranking quality, which brings us to the final research question:

RQ5 Can activity-based user interaction signals improve ranking quality?

We answer this question in Chapter 6 where we study activity-based user interaction signals, such as opening or editing documents, in the context of cloud-based file storage search. We find that it is possible to leverage these signals to train semantic matching models that can further improve the ranking quality of the search component.

1.2 Main Contributions

This section describes a list of the main contributions in this thesis.

1.2.1 Algorithmic contributions

1. A sample-based SGD learning algorithm called CounterSample with an improved convergence rate for counterfactual LTR; see Chapter 3.
2. A Safe Exploration Algorithm (SEA) that uses counterfactual learning and off-policy evaluation to explore new actions while maintaining the user experience; see Chapter 4.
3. Sliding-window and exponential-decay counterfactual estimators for off-policy evaluation in non-stationary settings; see Chapter 5.
4. A Semantic Matching Model that learns from activity data instead of clicks; see Chapter 6.

1.2.2 Theoretical contributions

1. A proof that the time until convergence of IPS-weighted SGD scales with the maximum IPS score; see Theorem 3.4.1.

2. A proof that the time until convergence of COUNTERSAMPLE scales with the average IPS score; see Theorem 3.5.2.

3. A proof that the performance of SEA is always at least as good as a baseline ranker; see Theorem 4.4.1.

4. A proof that IPS estimators are biased in non-stationary settings; see Lemma 5.3.3.

5. Proofs for the bias of sliding-window and exponential-decay IPS estimators in non-stationary settings; see Theorems 5.3.4 and 5.3.5.

1.2.3 Empirical contributions

1. A large-scale comparison of online and counterfactual approaches to unbiased LTR; see Chapter 2.

2. Testing the convergence rate of counterfactual LTR across optimizers, batch sizes and different severities of position bias; see Chapter 3.

3. Experiments to test the safety of several bandit algorithms on text classification and document ranking; see Chapter 4.

4. A comparison of IPS estimators for off-policy evaluation in stationary, abrupt non-stationary and smooth non-stationary settings; see Chapter 5.

5. An empirical validation of the use of activity data and click data based on user interaction logs for cloud storage search; see Chapter 6.

1.2.4 Open source software contributions

1. **PyTorchLTR** (`https://github.com/rjagerman/pytorchltr`)
   A PyTorch library for running LTR experiments. Used to run experiments in Chapter 3.

2. **ChainerCB** (`https://github.com/rjagerman/chainercb`)
   A Chainer library for running contextual bandit experiments. Used to run experiments in Chapter 5.

3. **Shoelace** (`https://github.com/rjagerman/shoelace`)
   A Chainer library for running LTR experiments.

4. **Glint** (`https://github.com/rjagerman/glint`)
   An asynchronous parameter server for running large-scale machine learning experiments in Spark.

1.3 Thesis Overview

The dissertation starts with Chapter 1, which you are currently reading. In this chapter we introduce the main subject of the dissertation, which is about learning from user interactions. Next, in Chapter 2 we empirically compare two families of methods for unbiased LTR from user interactions: *online* and *counterfactual* approaches. This is followed by Chapter 3 which introduces a counterfactual SGD learning algorithm that enjoys provably faster convergence than existing counterfactual SGD learning algorithms. Furthermore, the proposed method is empirically validated across a range of biased LTR scenarios. Next, Chapter 4 addresses the lack of exploration that counterfactual LTR methods suffer from. We introduce a safe (meaning to not harm the user experience) mechanism for exploration in counterfactual learning.

Until this point, the dissertation has assumed stationary user preferences that do not change over time and we have only looked at clicks as a user interaction signal. In the following two chapters we tackle non-stationary user preferences and use non-search interaction signals. Chapter 5 relaxes the stationarity assumption by introducing counterfactual estimators that work well in non-stationary environments. Then, in Chapter 6 we go beyond clicks as the main source of interaction data, instead looking at activity data, such as opening or editing documents, in the context of cloud file storage platforms. Finally, we conclude the thesis in Chapter 7 and provide directions for future work.

Readers familiar with counterfactual LTR and online LTR can skip the majority of Chapter 2 but should read at least section 2.6 to understand how counterfactual approaches compares to online approaches. All chapters are based on separate articles. We aim to keep the articles in their original state as much as possible. Because of this, it is unavoidable to have some overlap in the description of some baseline methods or core notation.

1.4 Origins

In this section we list the publications that form the basis for each chapter:

Chapter 2 is based on the conference paper:

> R. Jagerman, H. Oosterhuis, and M. de Rijke. To model or to intervene: A comparison of counterfactual and online learning to rank from user interactions. In *SIGIR*, pages 15–24. ACM, 2019.
>
> The counterfactual LTR experiments were performed by Jagerman, the online LTR experiments were performed by Oosterhuis. All authors contributed equally to the text.

Chapter 3 is based on the conference paper:

> R. Jagerman and M. de Rijke. Accelerated convergence for counterfactual learning to rank. In *SIGIR*, page 469–478. ACM, 2020.
>
> The method and proofs were developed by Jagerman. The experiments were designed and run by Jagerman. Both authors contributed to the text. Jagerman did most of the writing.

Chapter 4 is based on the journal paper:

R. Jagerman, I. Markov, and M. de Rijke. Safe exploration for optimizing contextual bandits. *TOIS*, 38(3):Article 24, April 2020.

The method was developed by Jagerman. The proofs were written by Jagerman with help of Markov. The experiments were designed and run by Jagerman. All authors contributed to the text. Jagerman did most of the writing.

Chapter 5 is based on the conference paper:

R. Jagerman, I. Markov, and M. de Rijke. When people change their mind: Off-policy evaluation in non-stationary recommendation environments. In *WSDM*, pages 447–455. ACM, 2019.

The estimators and methods were developed by Jagerman. The proofs were written by Jagerman with help of Markov. The experiments were designed and run by Jagerman. All authors contributed to the text. Jagerman did most of the writing.

Chapter 6 is based on the paper:

R. Jagerman, W. Kong, R. K. Pasumarthi, Z. Qin, M. Bendersky, and M. Najork. Improving cloud storage search with activity data. In *Under Review*, 2020.

This work was done as part of an internship at Google Research in Mountain View, California. Kong was the supervisor for this work. The method and models were developed by Jagerman and Kong, with helpful discussions and feedback from Pasumarthi, Qin and Bendersky. The experiments were run by Jagerman and Kong. All authors contributed to the text. Jagerman did most of the writing.

Furthermore, the thesis builds indirectly on the following work:

- R. Jagerman, C. Eickhoff, and M. de Rijke. Computing web-scale topic models using an asynchronous parameter server. In *SIGIR*, pages 1337–1340. ACM, 2017
- R. Jagerman, J. Kiseleva, and M. de Rijke. Modeling label ambiguity for neural list-wise learning to rank. In *SIGIR-NeuIR Workshop*. ACM, 2017.
- R. Jagerman, K. Balog, P. Schaer, J. Schaible, N. Tavakolpoursaleh, and M. de Rijke. Overview of TREC OpenSearch 2017. In *TREC*, 2017.
- R. Jagerman, K. Balog, and M. de Rijke. OpenSearch: Lessons learned from an online evaluation campaign. *JDIQ*, 10(3):Article 13, 2018.
- C. Lucchese, F. M. Nardini, R. K. Pasumarthi, S. Bruch, M. Bendersky, X. Wang, H. Oosterhuis, R. Jagerman, and M. de Rijke. Learning to rank in theory and practice: From gradient boosting to neural networks and unbiased learning. In *SIGIR*, pages 1419–1420. ACM, 2019.
- H. Oosterhuis, R. Jagerman, and M. de Rijke. Unbiased learning to rank: Counterfactual and online approaches. In *WWW*, pages 299–300. ACM, 2020.

2

A Comparison of Counterfactual and Online Learning to Rank

2.1 Introduction

Interest in Learning to Rank (LTR) approaches that learn from user interactions has increased recently [16, 59, 82, 184]. Compared to learning from annotated datasets [105], implicit feedback obtained through user interactions matches user preferences more closely [80]. Furthermore, gathering interactions is much less costly than expert annotations [28, 131]. Additionally, unlike LTR from annotated datasets, LTR from user interactions can respect privacy-sensitive settings [16]. However, a big disadvantage of user interactions is that they often contain different types of bias and noise. Hence, LTR methods that learn from user interactions mainly focus on removing bias from the learning process [16, 82, 119].

There are two main families of algorithms for *unbiased* LTR from user interactions:

1. Counterfactual Learning to Rank (CLTR) [82]: These algorithms learn a ranking model from a historical interaction log, often collected using a production system. They usually treat clicks as absolute relevance indicators and employ a form of reweighing in order to debias interaction data. Counterfactual methods have no experimental control; they avoid the risks associated with online interventions where untested rankings may be displayed. A disadvantage is that they cannot explore and are limited to rankings displayed by the production system.

2. Online Learning to Rank (OLTR) [184]: This class of algorithms interactively optimize and update a ranking model after every interaction. They combat bias by *interventions*, i.e., by displaying slightly modified rankings. This type of experimental control allows the learner to assess and learn novel rankings. Clearly, experimental control comes with a risk: untested rankings may hurt the user experience.

For practitioners the decision whether to use counterfactual or online LTR is important for practical deployment and user satisfaction with their ranking system. E.g., if there are situations where CLTR and OLTR methods provide the same performance, the risks of interventions can be avoided. However, if under some conditions CLTR methods

This chapter was published as [72].

are unable to reach the same performance as online interventions promise to bring, an OLTR method may be preferred. Currently, there has not been a study comparing methods across the two methodologies. As a result, it is currently unclear when which methods may be preferred, what benefits either methodology provides, and the scope of these benefits. Direct comparisons between CLTR and OLTR are required to help advance the field of LTR and inform its uptake.

A direct and fair comparison of counterfactual and online LTR algorithms is non-trivial for several reasons. First, CLTR methods do not affect the user experience as they learn from historical data; in contrast, the user experience is a vital part of the evaluation of OLTR methods. Second, unlike OLTR methods, CLTR methods assume there is no selection bias, and proofs of their unbiasedness depend on this assumption. Finally, the optimization problems for OLTR and CLTR methods are formulated differently – therefore they may not be optimizing the same metrics and observed differences could be a consequence of this difference.

To the best of our knowledge, this is the first study to provide a direct comparison of CLTR and OLTR methods. Our main goal in this chapter is to answer the following research question:

> **RQ1** How should LTR practicioners choose which method to apply from either counterfactual or online LTR methodologies?

In order to enable informed answers to this question, we address multiple aspects that are important to practitioners of both large-scale and small-scale LTR systems. First, we evaluate whether both approaches converge at the same level of performance, in other words, whether both approaches capture the true user preferences equally well. Furthermore, we investigate how the learning outcomes are affected by different levels of selection bias, position bias and interaction noise. Second, we evaluate how well the user experience is maintained during learning, since OLTR methods could potentially deter users with inappropriate interventions. Thirdly, we investigate the effect of interventions by allowing counterfactual methods to execute periodic deployments; this simulates multiple steps of optimization and deployment as one would see in practice.

The research questions we address are:

RQ1.1 Do state-of-the-art counterfactual and online LTR methods converge to the same level of performance?

RQ1.2 Is the user experience the same for counterfactual methods as for online methods?

RQ1.3 When do online interventions help the learning to rank algorithm?

In this chapter we present the first direct comparison between CLTR and OLTR methods. Our comparison leads to valuable insights as it reveals that, depending on the experimental conditions, a different methodology should be preferred. In particular, our results show that OLTR methods are more robust to selection bias, position bias and interaction noise. However, under low levels of bias and noise CLTR methods can obtain a significantly higher performance. Furthermore, to our surprise we find that some properties asserted to pertain to CLTR or OLTR methods in previously published

Table 2.1: Notation used throughout the chapter.

Notation	Description
d	document
D	set of documents
R	ranked list
R_i	document placed at rank i in ranked list R
$f_\theta(\cdot)$	ranking model with parameters θ
c_i	1 if document at rank i was clicked, 0 otherwise
p_i	The propensity score at rank i
$relevance(d)$	1 if document d is relevant, 0 otherwise

work appear to be lacking when tested. For instance, in contrast with previously published expectations [119] OLTR is not substantially faster at learning than CLTR, and while always assumed to be safe [173], CLTR may be detrimental to the user experience when deployed under high-levels of noise.

Our findings reveal areas where future LTR work could make important advances, and moreover, allow practitioners to make an informed decision on which LTR methodology to apply.

2.2 Counterfactual Learning to Rank

Counterfactual Learning to Rank (CLTR) [4, 9, 82] aims to learn a ranking model from historical interaction data. Employing an offline approach has many benefits compared to an online one. First, it is possible to try and iterate many different learning algorithms without needing to deploy them online. Furthermore, it avoids the pitfalls and engineering overhead of having to deploy an online learning system. Finally, models that are learned offline can be tested before actually being deployed online, alleviating some of the safety concerns surrounding OLTR, such as the aggressive exploration of online methods that may place irrelevant items at high ranked positions [99, 173].

A straightforward approach to LTR from historical user interactions is to collect clicks and treat them as signals of relevance [79]. This is referred to as *partial information feedback* because it only conveys information about the documents that the user *has seen* and clicked on, but not other documents that the user *could have* seen and clicked on. Traditional supervised learning algorithms expect data to be in a "full information" form, where it is exactly known which documents are relevant and which ones are not. This is never the case in user interactions due to biases and noise. As a solution, CLTR provides a way to deal with partial information feedback.

2.2.1 Unbiased LTR with biased feedback

Joachims et al. [82] introduce a method to utilize interaction data in LTR, by casting the problem as a counterfactual learning problem [156]. In [82], it is assumed that the user does not examine all documents in a ranked list, and is more likely to observe

documents at the top of the list than at the bottom; this is referred to as *position bias*. After a document is observed, a user will either judge it as relevant resulting in a click, or judge it as non-relevant. More formally, the user observes document d_i at rank i with some probability p_i, called the *propensity*.[1]

If the propensity is known, it is possible to modify an existing learning algorithm and simply re-weigh interaction data according to the propensity scores using *Inverse Propensity Scoring* (IPS). Joachims et al. [82] take the SVMRank algorithm and modify it to optimize a re-weighted objective, where each click is re-weighted according to whether the click appeared at the top of the ranked list (thus with high propensity) or lower in the ranked list (thus with lower propensity). Samples with high propensity are weighted less than samples with low propensity and vice versa. We will assume the propensities are known a priori and discuss related work dealing with propensity estimation in Section 2.2.3.

To formalize CLTR, we consider a ranking function $f_{\text{production}}$ that produces a ranked list R to be shown to the user in response to a query q. When a user clicks on a document in this ranking, they are revealing to us that this document is relevant. We denote a user's clicks by a 0/1 vector c:

$$c_i = \begin{cases} 1 & \text{if document } d_i \text{ was observed and judged relevant,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that it is possible for a user to click on more than one document during a session or click on no documents. Since a user is more likely to observe top-ranked documents than lower-ranked ones, we are more likely to observe relevance signals of the top-ranked documents. We denote the probability that a user observes the document at rank i with p_i; this is usually called the *propensity* of the observation.

We record a click log $\mathcal{D} = \{(R^{(j)}, c^{(j)})\}_{j=1}^{n}$, containing rankings of documents R and clicks c according to the procedure in Algorithm 1. For brevity we drop the superscript notation $\cdot^{(j)}$ for the session identifier. We now derive the learning objective of [82], a modified version of the SVMRank training objective that minimizes the average rank of relevant results, weighted by the inverse propensity scores:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(R,c)\in\mathcal{D}} \sum_{\{i:c_i=1\}} \frac{\text{rank}(R_i \mid f_\theta)}{p_i}. \tag{2.1}$$

It can be shown that the above training objective is unbiased and can be solved via a hinge loss formulation [82]. We will refer to this method as *Counterfactual SVMRank* (CF-RANK).

2.2.2 Unbiased LTR with additive metrics

The counterfactual learning framework described in the previous section can be adapted to optimize additive metrics [4]. For example, we can modify the training objective so it optimizes DCG [75], a common metric in the evaluation of rankings:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(R,c)\in\mathcal{D}} \sum_{\{i:c_i=1\}} \frac{\lambda(\text{rank}(R_i \mid f_\theta))}{p_i}, \tag{2.2}$$

[1] The notation we use in the chapter is listed in Table 2.1.

Algorithm 1 Data collection for Counterfactual Learning to Rank.

1: **Input**: production ranker: $f_{\text{production}}$; log size n;
2: **Output**: a session log $\mathcal{D}$
3: $\mathcal{D} = \emptyset$
4: **for** $t \leftarrow 1 \dots n$ **do**
5: $\quad q^{(t)} \leftarrow receive\_query(t)$
6: $\quad D^{(t)} \leftarrow preselect\_documents(q^{(t)})$
7: $\quad R^{(t)} \leftarrow rank\_documents(f_{\text{production}}, D^{(t)})$
8: $\quad c^{(t)} \leftarrow receive\_clicks(R^{(t)})$
9: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(R^{(t)}, c^{(t)})\}$
10: **end for**

where $\lambda(r) = \frac{-1}{\log(1+r)}$. This objective is both continuous and sub-differentiable, making it possible to solve using existing gradient descent techniques. We will refer to the DCG-optimizing counterfactual method as *Counterfactual DCGRank* (CF-DCG).

The counterfactual LTR algorithm is described in Algorithm 2. As input, the algorithm takes a set of weights (typically initialized to 0), a scoring function f, a learning rate μ and a click log $\mathcal{D}$. The algorithm runs for a fixed number of epochs which trades off computation time for convergence. The gradient is calculated on line 8 where a clicked document is compared against every other document. The gradient is computed as a λ-modified hinge loss: with $\lambda(r) = r$ this is the CF-RANK method, which attempts to minimize the rank of relevant results; and with $\lambda(r) = \frac{-1}{\log(1+r)}$ we obtain the CF-DCG method, which attempts to maximize DCG. Finally, the ranking model is updated via stochastic gradient descent on line 11.

2.2.3 Propensity estimation methods

Recent work in CLTR has focused on estimating propensities from data [9, 172, 173]. As the aim of our work is to compare counterfactual and online LTR approaches, we consider propensity estimation beyond the scope of this chapter and assume the propensity scores are known a priori. This is a reasonable assumption, as practitioners typically first perform a randomization experiment to measure the observation probabilities before applying a counterfactual learning algorithm [16].

Our experimental setup allows us to measure the difference of counterfactual methods and online methods without confounding our results by the accuracy of the propensity estimator.

2.3 Online Learning to Rank

Online Learning to Rank (OLTR) [59, 119, 144, 184] aims to learn by directly interacting with users. OLTR algorithms affect the data gathered during the learning process because they have control over what is displayed to users. These interventions potentially allow for more efficient learning by requiring less user interactions to reach a certain level of performance. Yet, an OLTR algorithm has to simultaneously

Algorithm 2 Counterfactual Learning to Rank (CLTR).

Input: initial weights: θ; scoring function: f; learning rate μ; click log $\mathcal{D}$; number of epochs: E.
for $e \leftarrow 1 \dots E$ **do**
  for $(R, c) \in \mathcal{D}$ **do**
    for $R_i : c_i = 1$ **do**
      $\nabla f_\theta \leftarrow \mathbf{0}$ *// initialize gradient*
      for $R_j : R_j \neq R_i$ **do**
        $\nabla f_\theta \leftarrow \nabla f_\theta + \nabla \left[\lambda(hinge(f_\theta(R_i) - f_\theta(R_j)))\right]$
        // (modified) hinge-loss gradient
      end for
      $\nabla f_\theta \leftarrow \frac{\nabla f_\theta}{p_i}$
      $\theta \leftarrow \theta + \mu \nabla f_{\theta_t}$ *// update the ranking model*
    end for
  end for
end for

provide good results to the user and learn from their interactions with the displayed results [117]. Thus, besides unbiasedly learning from user interactions, the user experience during learning should also be maintained. A great advantage of the online approach is that learned behavior is immediately applied. However, this high level of responsiveness also means that an unreliable OLTR method can decrease the user experience immediately, making it a potential risk. Therefore, it is important for OLTR methods to be reliable, i.e. unbiased and robust to noise.

In contrast to CLTR, OLTR methods do not explicitly model user behavior, i.e., they do not estimate observance probabilities. Instead, they use stochasticity in the displayed results to handle selection and position biases. In addition, their properties are only based on simple assumptions about user behavior [119, 184], e.g., *a relevant document is more likely to be clicked than a non-relevant document*. Thus, in cases where users are hard to model, OLTR may have an advantage over CLTR. In other areas of machine learning [147], online (or active) approaches tend to be more efficient than algorithms without control over data gathering w.r.t. data requirements. However, CLTR and OLTR methods have never been compared directly, thus currently we do not know if this advantage also generalizes to LTR problems.

2.3.1 Dueling bandit gradient descent

Dueling Bandit Gradient Descent (DBGD) [184] is the earliest OLTR method and is based on interleaving: an unbiased online evaluation method. Interleaving methods unbiasedly compare two ranking systems in the online setting [58, 77, 134]. Therefore, interleaving can be used to recognize an improvement to a ranking system. At each iteration DBGD compares its current model with a sampled variation using interleaving. If a preference towards the variation is inferred, the current model is updated in its direction. Over time this process estimates a gradient descent and the model should oscillate towards an optimum w.r.t. user preference. Most OLTR methods published to date are

extensions of DBGD. This includes, for instance, Multileave Gradient Descent [145], which compares multiple variations per iteration using multileaving [118, 144]; other methods reuse historical interactions to guide exploration [59, 187]. All of these extensions improve on the initial learning speed of DBGD. However, no extension has been shown to improve long-term performance [119, 122, 145]. Moreover, even under ideal circumstances and with a very large number of iterations, DBGD is unable to reach levels of performance comparable to LTR from labeled datasets [117, 119]. Despite these shortcomings, DBGD is a key method in the field of OLTR.

2.3.2 Pairwise differentiable gradient descent

In reaction to the shortcomings of DBGD, recent work has introduced the Pairwise Differentiable Gradient Descent (PDGD) algorithm [119]. In contrast to DBGD, PDGD does not depend on sampling model variations or online evaluation methods. Instead, PDGD constructs a pairwise gradient at each interaction, from inferred user preferences between documents. Algorithm 3 describes the PDGD method in formal detail. At each iteration, the algorithm waits until a query is issued by the user (line 3). Then PDGD creates a probability distribution over documents by applying a Plackett-Luce model with parameter $\tau \in \mathbb{R}_{>0}$ to the scoring function:

$$P(d|D,\theta) = \frac{e^{\tau f_\theta(d)}}{\sum_{d' \in D} e^{\tau f_\theta(d')}}. \tag{2.3}$$

We introduce the τ parameter to control the *sharpness* of the initial distribution, which indicates the confidence we have in the initial model. Previous work only considered cold-start situations thus did not require this parameter [119, 120]. From this distribution a result list is sampled (Line 5) and displayed to the user. Then PDGD infers preferences between the clicked documents and the first unclicked document and every unclicked document preceding a clicked document, a longstanding pairwise assumption [76]. With $d_i >_\mathbf{c} d_j$ denoting an inferred preference of d_i over d_j, PDGD estimates the model gradient as:

$$\nabla f_\theta \approx \sum_{d_i >_\mathbf{c} d_j} \rho(d_i, d_j, R, \theta) \nabla P(d_i \succ d_j | \theta), \tag{2.4}$$

where ρ is a weighing function used to deal with biases (line 8) and $P(d_i \succ d_j \mid \theta)$ is the probability of sampling d_i before d_j.[2] Finally, the scoring function is updated according to the estimated gradient (line 11), and the process repeats with the updated model.

The ρ function makes use of a *reverse pair ranking function* $R^*(d_i, d_j, R)$, which returns the same ranking as R with the position of document d_i and d_j swapped. Then, the value of ρ is determined by the ratio between the probabilities of the two rankings:

$$\rho(d_i, d_j, R, \theta) = \frac{P(R^*(d_i, d_j, R)|\theta)}{P(R|\theta) + P(R^*(d_i, d_j, R)|\theta)}. \tag{2.5}$$

[2] $P(d_i \succ d_j \mid \theta) = \frac{e^{\tau f_\theta(d_i)}}{e^{\tau f_\theta(d_i)} + e^{\tau f_\theta(d_j)}}$

PDGD assumes that if a user considers both d_i and d_j equally relevant, then inferring the preference in $d_i >_{\mathbf{c}} d_j$ in R is equally probable as inferring the reverse preference $d_j >_{\mathbf{c}} d_i$ in $R^*(d_i, d_j, R)$. Furthermore, if a user prefers one of the documents, inferring the corresponding preference is more likely than the reverse. These assumptions can formulated as:

$$\begin{aligned} &\textit{sign}(\textit{relevance}(d_i) - \textit{relevance}(d_j)) = \\ &\quad \textit{sign}\left(P(d_i \succ_{\mathbf{c}} d_j|R) - P(d_j \succ_{\mathbf{c}} d_i|R^*(d_i, d_j, R))\right). \end{aligned} \tag{2.6}$$

Intuitively, this means that relative relevance differences can be inferred by swapping document pairs without changing the rest of the ranking. A similar approach is used by counterfactual methods to estimate propensities [82], conversely, PDGD uses it to directly optimize its ranking model. In the original paper, Oosterhuis and de Rijke prove that the gradient estimation of PDGD is unbiased w.r.t. document pair preferences. This means that the expected gradient of PDGD can be written as a sum over all document pairs:

$$E[\nabla f(\cdot, \theta)] = \sum_{(d_i, d_j) \in D} \alpha_{ij} \left(f'(d_i, \theta) - f'(d_j, \theta)\right), \tag{2.7}$$

where α_{ij} is a unique weight for every document pair in the collection. PDGD is unbiased in the sense that the sign of α_{ij} matches the user preferences between d_i and d_j:

$$\textit{sign}(\alpha_{ij}) = \textit{sign}\left(\textit{relevance}(d_i) - \textit{relevance}(d_j)\right). \tag{2.8}$$

Thus, in expectation PDGD will perform an unbiased update towards the pairwise preferences of the user.

Recent work has extensively compared DBGD with PDGD [119, 120]; PDGD performs considerably better in terms of final convergence, user experience during optimization, and learning speed. These findings generalize from settings with no to moderate levels of position bias and noise [119] to circumstances with extreme levels of bias and noise [120]. PDGD is the new state-of-the-art for OLTR, and we will therefore not consider DBGD in our comparison.

2.4 Expectations from Previous Work

This section will discuss several expectations about the qualitative differences between CLTR and OLTR based on previous work. Subsequently, Section 2.5 describes the experiments that have been run to test these expectations and Section 2.6 their outcomes. By discussing existing expectations here, we can later contrast them with our observations. Whether and how our results match our expectations can reveal how well our understanding of LTR from user interactions is.

Expectation 1 – The performance at convergence. As described in Section 2.2, it has been proven that CLTR can unbiasedly optimize additive metrics [4], for instance using CF-DCG, when the observation probabilities of the user are correctly known.

Algorithm 3 Pairwise Differentiable Gradient Descent (PDGD).

Input: initial weights: θ_1; scoring function: f; learning rate μ.
for $t \leftarrow 1 \dots \infty$ **do**
  $q^{(t)} \leftarrow$ *receive_query*(t) *// obtain a query from a user*
  $D^{(t)} \leftarrow$ *preselect_documents*$(q^{(t)})$ *// preselect doc. for query*
  $\mathbf{R}^{(t)} \leftarrow$ *sample_list*$(f_\theta, D^{(t)})$ *// sample list according to Eq. 2.3*
  $\mathbf{c}^{(t)} \leftarrow$ *receive_clicks*$(\mathbf{R}^{(t)})$ *// show result list to the user*
  $\nabla f_\theta \leftarrow \mathbf{0}$ *// initialize gradient*
  for $d_i >_\mathbf{c} d_j \in \mathbf{c}^{(t)}$ **do**
    $\nabla f_\theta \leftarrow \nabla f_\theta + \rho(d_i, d_j, R) \nabla P(d_i \succ d_j | \theta)$
  end for
  $\theta \leftarrow \theta + \mu \nabla f_\theta$ *// update the ranking model*
end for

Conversely, for PDGD there is no known proof that it optimizes any metric unbiasedly. Therefore, we expect CLTR methods like CF-DCG to reach a higher level of performance than PDGD if the propensities are known, since CLTR can guarantee that the performance metric is optimized, while for PDGD it is unclear whether its pairwise gradient will optimize the metric precisely.

Expectation 2 – The user experience during learning. The field of OLTR has long claimed that their methods provide the most responsive experience [59, 119, 145] because OLTR methods apply their learned model instantly. However, noise may cause a method to decrease model quality (temporarily) and exploration adds stochasticity to the results, thus risking a worsened user experience. As a result, we expect an OLTR method to provide an experience worse than the initial ranker at the start, but as learning continues the user experience should eventually exceed that of the initial model. In contrast, CLTR methods do not affect the user experience during learning as they work with historical data, and therefore, also cannot improve it. Nevertheless, this approach completely avoids the risks of degrading the user experience. Therefore, we expect OLTR to provide a worse experience than under click gathering for CLTR initially, yet eventually the experience under OLTR should exceed that of CLTR. The question is whether the long-term improvements of OLTR outweigh the initial decrease.

Expectation 3 – The effect of interventions. Interventions are expected to greatly reduce the data requirements for learning [147], as they allow algorithms to gather data that is more useful for their current state. Correspondingly, OLTR methods are expected to learn faster [59, 145], in other words, they should require less user interactions to reach a decent level of performance than CLTR methods [119]. Similarly, allowing CLTR methods to intervene, e.g., by deploying a current model should make them more efficient as well.

This concludes the key expectations regarding the performance differences between CLTR and OLTR methods. While these expectations are based on previously published literature on CLTR and OLTR [4, 119, 147], they have never directly been tested. To the best of our knowledge, our study is the first to confirm or challenge them with hard experimental facts.

Table 2.2: Click probabilities after observing a document in the result list for different user models.

	$P(\text{click} = 1 \mid \text{observed} = 1, \text{rel}(d))$				
$\text{rel}(d)$	0	1	2	3	4
Perfect	0.00	0.20	0.40	0.80	1.00
Binarized	0.10	0.10	0.10	1.00	1.00
Near-Random	0.40	0.45	0.50	0.55	0.60

2.5 Experiments

Our experiments evaluate the user experience of several methods at different time-steps and a multitude of conditions with varying levels of interaction noise, position bias, and selection bias. Due to the scope of this comparison and the varying requirements, we rely on a synthetic setting based on an existing LTR dataset and simulated user behavior. Our setup is an extension of the synthetic experiments common in both OLTR [59, 119, 145] and CLTR [9, 82].

2.5.1 Optimization setup

We use the *Yahoo! Webscope* dataset [28]; it contains a set of queries with a unique set of preselected documents for each query. The dataset provides a train, validation and test split. We use the train partition during optimization of the methods, the validation set for tuning hyperparameters and the test partition to report our results. Each query-document pair is represented by a feature vector and a relevance label, the relevance labels are in a five-degree scale ranging from *not relevant* (0) to *perfectly relevant* (4).

A baseline ranker is trained to serve as a logging ranker for the CLTR methods, and an initial ranker to warm-start the OLTR method. To create the baseline ranker, we follow the setup of Joachims et al. [82] and train an SVMRank ranker on 1% of the queries in the training data. This setup is chosen as it reflects a common real-world scenario: it is possible to manually annotate a small amount of data to learn an initial ranker, and then use a large amount of logged interaction data, either online or counterfactually, to further improve this ranker.

Finally, the gathering of click-data is simulated using the following steps: First, a user-issued query is simulated by uniformly sampling a query from the training partition of the dataset. Then, the corresponding documents are ranked according to the applied LTR method, i.e., by the logging policy for CLTR methods or by the algorithm itself in OLTR. Subsequently, the ranked results are displayed to a simulated user who then clicks on any number of documents (including none); Section 2.5.2 details the behavior models we applied. Lastly, the resulting clicks are presented to the LTR method, which may now use the interaction for optimization.

2.5.2 Simulating user behavior

We simulate user behavior by modelling three aspects of user behavior in search: interaction noise, position bias and selection bias.

First, *interaction noise* affects the probability of a user clicking on a document after observing it. The probability of clicking is conditioned on the relevance label of a document, as users are more likely to click on more relevant documents. Table 2.2 provides the click probabilities for three different click behavior models: *Perfect* click behavior has probabilities proportional to the relevance and never clicks on a non-relevant document, simulating an *ideal* user. *Binarized* click behavior acts on only two levels of relevance and is affected by position-bias; this simulated behavior has been used in previous work on CLTR [4, 9, 82]. And *Near-Random* behavior clicks very often, and only slightly more frequently on more relevant documents than on less relevant documents; this behavior simulates very high levels of click noise.

Second, *position bias* is modelled by observation probabilities; for a document at rank i the probability of being observed is determined by the parameter γ and formula:

$$P(observed = 1 \mid i) = \left(\frac{1}{i}\right)^{\gamma}. \tag{2.9}$$

Again this follows previous work on CLTR [4, 9, 82]. We apply this position bias to the *Binarized* and *Near-Random* user models; the *Perfect* user observes all documents every time and thus has no position bias. In our experiments we use $\gamma = 1$ and $\gamma = 2$ to model different levels of position bias.

Thirdly, we simulate *selection bias*, which occurs when not all documents can be displayed and thus also not observed. In practice it also occurs because users never look past certain positions, for instance, users rarely look beyond the initial page of many multi-page result displays. We model selection bias by giving a zero observance probability to documents beyond rank 10. This is common practice in OLTR experiments [59, 119, 145]; in contrast, CLTR methods assume that no selection bias is present. To investigate the effect of selection bias, our experiments both contain simulations with and without it.

In conclusion, we can apply selection bias, have two levels of position bias, and three levels of interaction noise. In total, we apply ten different types of user behavior: *Perfect* click behavior with and without selection bias, the *Binarized* and *Near-Random* click behaviors with two levels of position bias, with and without selection bias. To the best of our knowledge this is the most extensive set of types of behavior used for evaluating CLTR and OLTR methods, in addition to being the first comparison between the two methodologies.

2.5.3 Evaluation

To measure the performance of a ranker at any time step, we evaluate it on held-out annotated test data using the $nDCG@10$ metric [75]. We use the learned models without any exploration when evaluating performance at convergence. To evaluate the user experience during learning for OLTR we apply the algorithm with exploration to

the held-out dataset; this measures the performance for a previously unseen query that appears during optimization.

To test for statistical significance we average the results over multiple runs and apply a two-tailed t-test. In Section 2.6, we indicate for each comparison, whether the observed difference is statistically significant with $p < 0.01$.

2.5.4 Methods and interventions

Our comparisons concern one OLTR method and several CLTR methods, in addition to CLTR methods with periodic deployments during the optimization process.

The OLTR method in the comparison is PDGD (Section 2.3); the parameters ($\mu = 0.01$, $\tau = 10$) were tuned on the validation set.

The CLTR methods we apply are CF-RANK (Section 2.2.1) and CF-DCG (Section 2.2.2); the former is the original CLTR method while the latter optimizes DCG corresponding to our evaluation metric. For each run, the CLTR methods are given the propensity scores of the actual user models applied; this guarantees that the CLTR methods optimize unbiasedly. Furthermore, to investigate the effect of interventions we also run these methods with *periodic deployment*. For these runs we replaced the logging policy with the (then) current ranking model after every 200,000 iterations, thus simulating a situation where the learned model is deployed periodically. The parameters for the CLTR were optimized for every instance of user behavior and number of interactions on the validation set, thus results at different time-steps may use different parameter settings. The complete set of hyper parameter settings that were used is released alongside our source code; see Section 2.8 for details.

2.6 Results and Analysis

This section will answer the research questions posed in Section 2.1, using our experimental results displayed in Figure 2.1, 2.2, 2.3, and 2.4.

2.6.1 Ranking performance

We investigate the ranking performance of the OLTR and CLTR methods under different experimental conditions to answer RQ1.1:

> Do state-of-the-art counterfactual and online LTR methods converge to the same level of performance?

As discussed in Section 2.4 we expect CF-DCG to have better performance when the assumptions for unbiased CLTR are met, as unlike PDGD, CF-DCG is then proven to optimize the DCG metric. Figure 2.1 displays the performance of the counterfactual and online LTR methods over 1,000,000 sessions, where a session consists of a query, a ranking of documents and all corresponding clicks generated by the click simulation. These results are grouped by the user behavior under which they were optimized, which varies in the amount of selection bias, position bias, and interaction noise.

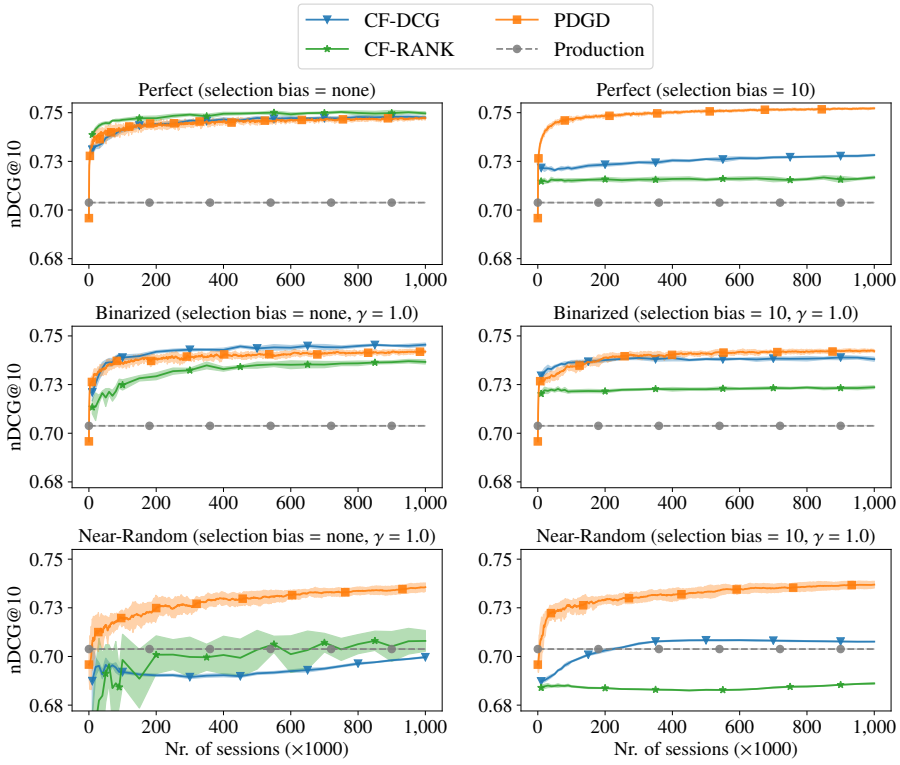

Figure 2.1: Performance of online and counterfactual methods under perfect, binarized, and near-random user models. In the left column no selection bias is present; in the right column, a selection of 10 is used.

First, we consider the results without selection bias displayed in the left column of Figure 2.1. Under *Perfect* and *Binarized* click behavior the performance of the CLTR methods and PDGD are quite comparable, with CF-DCG performing better than PDGD in the *Binarized* case ($p < 0.01$). In contrast, performance of the CLTR methods drops below the production ranker under *Near-Random* click behavior, and does not exceed it within 1,000,000 iterations ($p < 0.01$). This goes against our expectations, as CLTR methods in the *Near-Random* case should, in expectation, find the best ranking function because the relative ordering of ranking functions is preserved [82]. PDGD, on the other hand, is much less affected and reaches much higher levels of performance. Because the *Binarized* and *Near-Random* behaviors have the same position bias, the difference in performance must be due to the increased interaction noise in the latter. Thus, it appears that the CLTR methods are much less robust to noise than PDGD, yet with low levels of noise CLTR methods outperform the OLTR method.

Second, we look at the results with selection bias displayed in the right column in Figure 2.1. For each user model the performance of the CLTR methods is worse than without selection bias. Except under *Near-Random* click behavior where CF-DCG

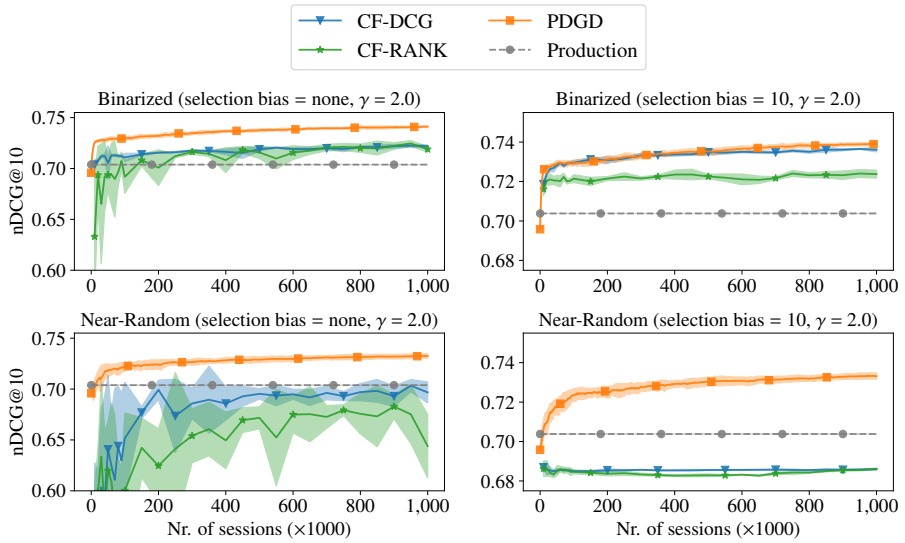

Figure 2.2: Performance of online and counterfactual methods under very strong position bias ($\gamma = 2$). The scale of the y-axis of the plots in the left column has been modified to be able to show the large variance. In the left column no selection bias is present; in the right column, a selection of 10 is used.

now performs slightly better than the production ranker. Unbiased CLTR does not consider selection bias [82] which could explain this unexpected result. In contrast, the performance of PDGD is affected very little in comparison and is now better than both CLTR methods under all click behaviors ($p < 0.01$). Thus, it appears that PDGD is preferable when selection bias is present.

Third, to understand the effect of position bias we look at the results in Figure 2.2, where strong position bias is simulated with $\gamma = 2$. It is clear that all methods are negatively affected by strong position bias. Unexpectedly, PDGD now outperforms the CLTR methods in all cases ($p < 0.01$), even though the *Binarized* click behavior without selection bias provides the exact circumstances for which CLTR was designed [82]. Therefore, we attribute the negative effect on CLTR to high variance since the methods are still proven to be unbiased in this case. This may further explain why selection bias has a positive effect on the CLTR methods under the *Binarized* click behavior: it removes documents with low propensities that lead to high variance. Clearly, we see that OLTR is better at handling high levels of position bias than CLTR.

In conclusion, we answer RQ1.1 negatively: online and counterfactual methods do not converge to the same level of performance. However, which method reaches the best performance depends heavily on the conditions under which they are deployed. In the presence of selection bias or under high levels of position bias or interaction noise OLTR reaches the highest performance. However, when there is no selection bias, little position bias and little interaction noise, then CLTR reaches a level of performance that OLTR is unable to obtain. Counter to our expectations, even when the assumptions of

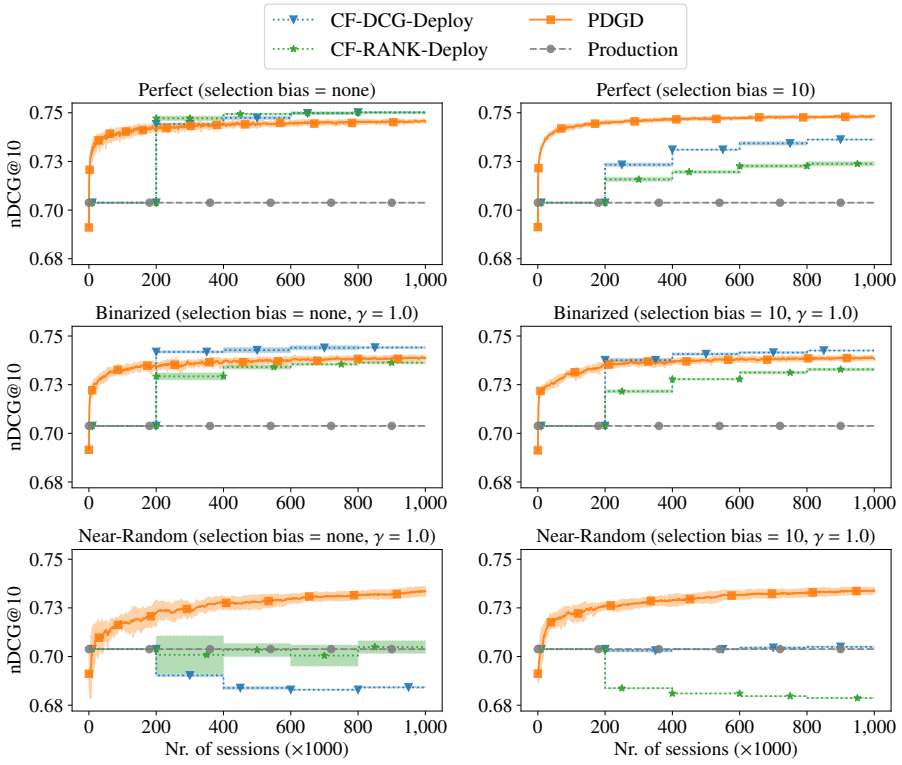

Figure 2.3: Display performance during training, indicating user experience. In the left column no selection bias is present; in the right column, a selection of 10 is used.

CLTR are true, the CLTR methods are still not robust to noise. Thus, to be able to make the best decision between the CLTR and OLTR methodologies, a practitioner should first measure the severity of different types of bias and noise in their search scenario.

2.6.2 User experience

In this section, we examine the quality of displayed rankings in order to answer RQ1.2:

> Is the user experience the same for counterfactual methods as for online methods?

Figure 2.3 shows the quality of rankings displayed by the PDGD method during optimization and of the *Production* ranker used to gather click-logs for the CLTR methods. For clarity: we are not discussing the *CF-DCG-Deploy* and *CF-RANK-Deploy* results for this research question, they will be discussed in Section 2.6.3.

In Section 2.4 we stated the expectation that OLTR methods start with a user experience worse than the production ranker due to exploration. However, OLTR is expected to overtake the production ranker as it continues to learn from interactions. The results

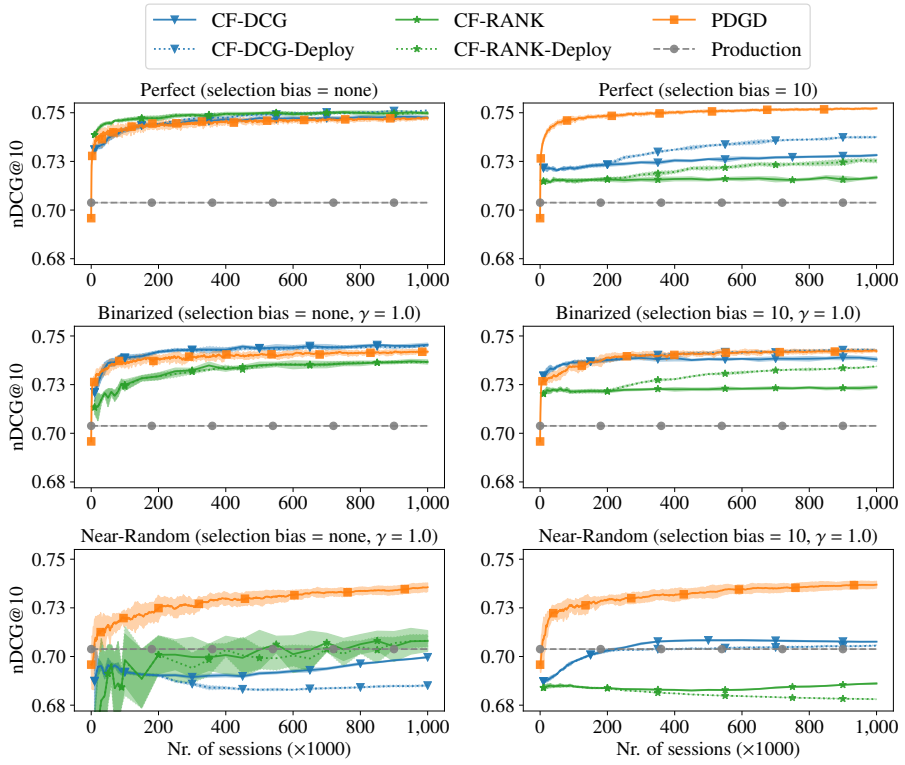

Figure 2.4: Performance of counterfactual methods with a deployment every 200,000 sessions. In the left column no selection bias is present; in the right column, a selection of 10 is used.

in Figure 2.3 confirm this expectation. Across all types of user behavior, we see that the initially displayed performance is substantially worse than the production ranker ($p < 0.01$). PDGD provides considerably better rankings than the production ranker within 1,000, 2,000 and 21,000 sessions for *Perfect*, *Binarized* and *Near-Random* click behavior, respectively ($p < 0.01$). Thus, we conclude that PDGD provides a better user experience than CLTR methods overall, with a decrease in quality for a limited initial period.

Therefore, we answer RQ1.2 negatively: OLTR does not provide the same user experience as CLTR. Besides a limited initial period, OLTR provides a more responsive user experience during optimization than CLTR. However, it is up to practitioners to decide whether the initial worse period is worth it, or whether they prefer the constant user experience in the click gathering for CLTR.

2.6.3 The power of interventions

In this section we investigate whether performing interventions helps the learning performance, so as to answer RQ1.3:

> When do online interventions help the learning to rank algorithm?

To answer this question we consider the performance of the optimized models in Figure 2.4, and the user experience during click gathering in Figure 2.3.

In Section 2.4 we stated the expectation that interventions significantly speed up the learning process. In Figure 2.4 the performance of CLTR methods diverge at the first moment of deployment: after 200,000 sessions. We see that only in cases with high interaction noise, i.e., *Near-Random* click behavior, periodic deployment leads to worse performance than without ($p < 0.01$). For *Perfect* and *Binarized* click behavior, periodic deployment has no negative effects, moreover, when selection bias is present it substantially increases performance ($p < 0.01$). Thus it appears that interventions cause CLTR methods to reach higher levels of performance and especially help in dealing with selection bias.

Then we examine Figure 2.3, which displays the user experience during click gathering. Here we see that interventions allow users to benefit from improvements earlier, or suffer from deteriorations sooner. The same trend appears: a worse experience under high interaction noise, a better experience with little noise. Furthermore, CF-DCG with periodic deployment is capable of providing a better user experience than PDGD when little noise is present ($p < 0.01$). Unlike PDGD, CF-DCG does not perform exploration, which seems to be a crucial advantage in these cases.

Lastly, we discuss the expectation that interventions speed up learning, in particular that OLTR methods require significantly less data. None of our results indicate that OLTR learns faster than CLTR methods. While in many cases OLTR reaches higher performance levels than CLTR, when they reach comparable levels they do so after similar numbers of interactions. We suspect the reason to be that PDGD does not reiterate over previous interactions, where the CLTR methods perform numerous epochs. Nonetheless, despite expectations in previous work, our results do not indicate that the interventions of OLTR reduce data requirements.

To answer RQ1.3: interventions help CLTR methods in circumstances where they already improve over the production ranker. Moreover, their effect is substantial when dealing with selection bias. Unfortunately, deployment in difficult circumstances, i.e., with high levels of noise, can decrease performance even further and negatively affect the user experience considerably. Thus, practitioners should realize that a repeated cycle of optimization and deployment with CLTR can be quite harmful to the user experience. Counterfactual evaluation [101, 158, 163] could estimate whether the deployment of a model improves the experience, before deployment. The question is whether this evaluation is reliable and sensitive enough to prevent harmful changes.

2.7 Related Work

In this section we discuss previous work that concerns large-scale comparisons of LTR methods.

Liu [105] provides a comprehensive overview of the (then) state-of-the-art in LTR from labeled data but does not include a large-scale empirical comparison of methods. Tax et al. [161] do compare LTR algorithms that use manually annotated training data in a large-scale cross-benchmark comparison. They show that there is no single optimal LTR algorithm and provide a selection of supervised LTR methods that are pareto-optimal. In this chapter we compare two different families of LTR algorithms: *online* and *counterfactual* LTR methods, neither of which learn from manually annotated data; both types of method utilize user interactions. As such, the algorithms we compare are not supervised in the traditional sense [82].

A systematic comparison of CLTR methods appears to be lacking at this moment in time. Joachims and Swaminathan [78] seem to have provided the first comprehensive overview of counterfactual methods for LTR aimed at the information retrieval community, but the authors do not include a large-scale experimental comparison. More recently, Ai et al. [10] provide an overview of existing approaches to CLTR and describe both the theory and detailed instructions on how to deploy CLTR in practice. Furthermore, their work also contrasts CLTR with click models [30] but it does not contrast CLTR and OLTR methods.

Similarly, a systematic comparison of OLTR methods appears to be lacking too. The comprehensive survey due to Grotov and de Rijke [50] is several years old; it does not provide a large-scale experimental comparison nor does it contrast CLTR and OLTR methods; modern OLTR algorithms such as PDGD are also not included. In a more recent tutorial on OLTR, Oosterhuis [116] does provide a theoretical and experimental comparison of OLTR methods based on Dueling Bandit Gradient Descent and PDGD.

Our aim in this study is to gain an understanding in what situations counterfactual and online LTR approaches are appropriate to be used. To the best of our knowledge, there is no prior work that systematically compares counterfactual and online LTR approaches, or answers this question.

2.8 Conclusion

The goal of our study in this chapter was to answer RQ1:

> How should LTR practicioners choose which method to apply from either counterfactual or online LTR methodologies?

The choice between OLTR and CLTR is important as there are large differences between the results obtained by the two methodologies. We recognize three factors that determine which approach should be preferred: selection bias, position bias, and interaction noise. CLTR reaches a higher level of performance than OLTR in the absence of selection bias, and when there is little position bias or interaction noise. In contrast, OLTR outperforms CLTR in the presence of selection bias, high position bias, high interaction noise, or any combination of these three. Surprisingly, CLTR methods can decrease performance w.r.t., the production ranker when high levels of noise are present, even in situations where they are proven to be unbiased. We conclude that

OLTR is more robust to different types of bias and noise than CLTR. Therefore, practitioners should be well aware of the levels of bias and noise present in their search setting to choose between the two methodologies.

Unlike OLTR, CLTR does not need to intervene and can use data collected by the production ranker, which prevents a negative impact on the user experience. OLTR initially provides an experience worse than the production ranker but very quickly substantially improves over it. We have not observed OLTR having large negative effects on the user experience, even under high levels of interaction noise. However, practitioners should consider whether they are willing to risk the initial worsened user experience in order to get long term gains.

We observed that cycles of optimization and deployment with CLTR methods can have harmful effects on performance and user experience. High levels of interaction noise can severely worsen model quality for CLTR; if, subsequently, such a model is deployed, it can worsen the next model even further. Thus, practitioners should realize that CLTR brings risks to the user experience and evaluate models before deploying them, for instance using offline or counterfactual evaluation [101, 158, 163].

Our comparison is not without limitations. In our experiments, the CLTR methods were provided with the exact propensities; in realistic settings these values are not known and have to be estimated [9]. Thus we do not consider how errors in propensity estimation affect the comparison. Additionally, our comparative study considers only a single metric on a single dataset. Although the dataset and metric we use are widely accepted as a benchmark in both OLTR and CLTR, we would like to extend our study to multiple datasets, measuring across various dimensions and utilizing real user behavior from deployed systems in future work. Furthermore, we have seen that CLTR methods suffer in situations with high position bias. In Chapter 3 we will consider a more efficient CLTR learning algorithm that can deal with more extreme IPS weights and as a result can more reliably handle situations with high position bias. Future work should also consider heuristic methods such as propensity clipping; these methods reduce variance but make CLTR biased. Finally, our findings also reveal the importance of safety in LTR: naively deploying counterfactually learned models can have detrimental effects on model performance. We consider this topic in more depth in Chapter 4 where we propose a method that can automatically deploy a counterfactually learned model when it is safe to do so. This approach can provide a better user experience with little effort from practicioners.

3

Accelerated Convergence for Counterfactual Learning to Rank

3.1 Introduction

Learning to Rank (LTR) from *user interactions* [50], as opposed to learning from *annotated datasets* [105], has seen increased research interest due to its immense practical value. As we have argued in Chapter 1, learning from implicit feedback enjoys several advantages over learning from professional annotations:

1. User interaction signals are available at large scale and cost much less than professional annotations.

2. Implicit feedback captures the user's true interest more accurately.

3. Interaction data can be utilized in domains where professional annotations are impractical, unethical, or impossible, for example in personal search.

However, learning from user interactions is not without difficulty and one of the major challenges is the biased nature of user interactions [32, 79]. For example, in LTR, one of the most important types of bias that affects user interaction data is *position bias*, a phenomenon where users observe, and as a result click on, top-ranked items more than lower ranked ones.

Recent work has focused on removing bias by applying methods from counterfactual learning [82]. Most notably, Inverse Propensity Scoring (IPS) is commonly used to perform unbiased learning. A widely used approach for unbiased learning is to treat inverse propensity scores as weights and solve a weighted optimization problem via Stochastic Gradient Descent (SGD) [6, 10, 83].

A major challenge of learning with IPS-weighted SGD is that the inverse propensity scores introduce a large amount of variance in the gradients. This effect is especially severe in scenarios where the propensity scores can take on extreme values. For example, in product search, the set of candidate results tends to be large [13] and the query distribution can be heavily skewed [53] (e.g., due to periodic or seasonal

This chapter was published as [66].

influences), which means that some items get very little exposure and as a result have extreme IPS weights.

In this chapter we investigate the relationship between IPS weights and the convergence rate of IPS-weighted SGD. We prove that IPS-weighted SGD suffers from a slow convergence rate when the propensity weights are large. We also argue that as long as stochastic gradients are scaled with IPS-weights, this slowdown *cannot* be improved. This means that for many practical LTR scenarios, learning a ranking model is inefficient and convergence is slow.

We overcome the above limitation with a novel sample-based learning algorithm, called CounterSample, that samples learning instances proportional to their IPS weight instead of weighting learning instances by their IPS weight. Because of this strategy, we are able to control the variance, which, in turn, leads to accelerated convergence. We show that this new approach provably enjoys faster convergence while remaining unbiased and computationally cheap. We complement these theoretical findings with extensive experiments. CounterSample consistently converges faster, and in some cases learns a better ranker than IPS-weighted SGD, in a number of biased LTR scenarios – across optimizers, across batch sizes and for different severities of position bias.

Our main contributions in this chapter are:

- We analyze the convergence rate of IPS-weighted SGD algorithms and formalize the relationship between IPS weights and the convergence rate.
- We introduce a novel learning algorithm for Counterfactual Learning to Rank called CounterSample that enjoys provably faster convergence than IPS-weighted SGD approaches.
- We empirically show that CounterSample converges faster than competing methods in a number of biased LTR scenarios.

3.2 Related Work

3.2.1 Counterfactual learning to rank

Recent work on unbiased Learning to Rank (LTR) uses counterfactual learning [155, 156] to remove different types of bias such as position bias from click data to improve ranking performance [82, 172]. These methods typically use Inverse Propensity Scoring (IPS) to enable unbiased learning. A popular approach for solving IPS-weighted learning problems is to use Stochastic Gradient Descent (SGD) algorithms [6, 10, 83]. Existing approaches accomplish this by scaling the loss (and as a result, the gradients) with IPS weights. Although the empirical success of such approaches has been well documented in the literature [6, 10, 83], the impact of IPS weights on the *convergence rate* of SGD is not well understood.

In this chapter we address this problem by investigating the relationship between IPS weights and the convergence rate of SGD. Furthermore, we introduce a novel learning method that, unlike previous approaches, does *not* scale the loss or gradients, but instead guarantees unbiasedness by employing a sampling procedure.

3.2.2 Position bias

An important aspect of unbiased LTR is estimating the observation probabilities (often called *propensities*), which are necessary to apply IPS-weighting. For example, Wang et al. [173] propose result randomization strategies for obtaining observation probabilities under a position bias user model. Recent work has focused on *intervention harvesting*, a less invasive method for propensity estimation [6, 43].

In our work, we do *not* focus on the propensity estimation aspect of unbiased LTR, instead assuming the propensity scores are known a priori, because our goal is to study the *convergence rate* of IPS-weighted SGD algorithms.

3.2.3 Convergence rates for stochastic gradient descent

There is a significant amount of work studying upper bounds for the convergence rate of SGD [139] under varying assumptions of convexity and smoothness of the optimization objective [54, 55, 137, 149]. The convergence rate proofs presented in this chapter build on proofs provided by Shalev-Shwartz and Ben-David [148]. However, unlike previous work, our work investigates the role of IPS weights in optimization problems. We note that there is work investigating the use of importance sampling for SGD algorithms [11, 88, 115, 186]. These approaches all start from an unbiased dataset and then improve the convergence rate of SGD by manually perturbing the sampling distribution during learning, which introduces bias, and then applying IPS-weighting to remove the introduced bias.

Our work is different from these approaches because we learn from an *already biased* click log dataset, where the source of bias is outside of our control (e.g., position bias coming from users). This means we do not assume control over how the dataset is generated nor do we assume control over the propensity scores.

Finally, the idea of breaking large importance weights into smaller ones was previously considered by Karampatziakis and Langford [87]. Our work is different from [87] because: (a) we focus on the LTR scenario, and, (b) we propose a *sampling procedure* whereas [87] propose modifications to the update rule based on importance weights.

3.3 Background

We consider the problem of Counterfactual LTR as described in [82]. First, we introduce our notation for LTR with additive metrics. After that, we describe Counterfactual LTR from biased click feedback and present the IPS-weighted SGD algorithm that is commonly used for Counterfactual LTR.

3.3.1 Learning to rank with additive metrics

Let $f_{\boldsymbol{w}}(q, d)$ be a scoring function, parameterized by $\boldsymbol{w}$, that, for a given query q and item $d \in D_q$, produces a real-valued ranking score, where D_q is a set of candidate items for query q. Let $rel(q, d) \in \{0, 1\}$ be the relevance of item d to query q, where we assume binary relevance for simplicity. We write $rank(d \mid q, D_q, f_{\boldsymbol{w}})$ for the rank

of item $d \in D_q$ after sorting all items D_q by their respective scores using the scoring function $f_{\boldsymbol{w}}$. We consider the class of additive ranking metrics, as described in [5]:

$$\Delta(f_{\boldsymbol{w}} \mid q) = \sum_{d \in D_q} \lambda(rank(d \mid q, D_q, f_{\boldsymbol{w}})) \cdot rel(q, d), \tag{3.1}$$

where λ is a weighting function of the rank of an item that can capture different ranking metrics such as Average Relevant Rank, DCG, Precision@k and more [5]. For simplicity we will assume $\lambda(x) = x$, but note that our findings hold for any convex, (sub)differentiable and monotonically increasing weighting function λ. It is common practice to make Equation 3.1 differentiable by upper bounding the $rank(d \mid q, D_q, f_{\boldsymbol{w}})$ term with a pairwise hinge loss [5, 76, 82]:

$$\begin{aligned} &rank(d \mid q, D_q, f_{\boldsymbol{w}}) \leq \\ &\quad 1 + \sum_{d' \in D_q} \max(0, 1 - (f_{\boldsymbol{w}}(q, d) - f_{\boldsymbol{w}}(q, d'))). \end{aligned} \tag{3.2}$$

Now, suppose we are given a sample Q of i.i.d. queries $q \sim P(q)$. The goal in Learning to Rank (LTR) is to learn a scoring function $f_{\boldsymbol{w}}$ that minimizes risk:

$$\begin{aligned} &\arg\min_{\boldsymbol{w}} R(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \int_q \Delta(f_{\boldsymbol{w}} \mid q) dP(q) \\ &= \arg\min_{\boldsymbol{w}} \mathbb{E}_Q \left[\frac{1}{|Q|} \sum_{q \in Q} \sum_{d \in D_q} \lambda(rank(d \mid q, D_q, f_{\boldsymbol{w}})) \cdot rel(q, d) \right]. \end{aligned} \tag{3.3}$$

In most practical settings we cannot directly observe the relevance $rel(q, d)$, but only partial feedback in the form of clicks collected on rankings produced by a deployed production ranker. As a result, we cannot directly minimize the empirical risk associated with Equation 3.3. Furthermore, unlike online LTR approaches and bandit algorithms, we assume that we do not have any control over the deployed production ranker (i.e., we cannot perform *interventions*). In Counterfactual LTR we instead focus on minimizing an unbiased estimate of the empirical risk using a historical click log.

3.3.2 Counterfactual learning to rank with biased feedback

Suppose a deployed production ranker is collecting user interactions, i.e., clicks, as follows:

- a user issues a query $q \sim P(q)$;
- the user is presented with a ranking of the candidate items D_q for the issued query; and
- the user observes and clicks on an item $d \in D_q$ with some probability $P(c(d) = 1)$, where $c(d) \in \{0, 1\}$ indicates a click on item d.

In line with existing work [32, 82] we assume that the examination hypothesis holds, i.e., that clicks can only occur on observed items. In other words, a click on an item depends on the probability that the user observes the item *and* decides to click on it:

$$P(c(d) = 1) \stackrel{\text{def}}{=} P(o(d) = 1) \cdot P(c(d) = 1 \mid o(d) = 1), \tag{3.4}$$

where $o(d) \in \{0, 1\}$ indicates that item d was observed by the user. Finally, for any two observed items d and d' it is more likely that a user clicks on a relevant item than a non-relevant item. More formally:

$$\begin{aligned} &rel(q, d) > rel(q, d') \\ &\implies P(c(d) = 1 \mid o(d) = 1) > P(c(d') = 1 \mid o(d') = 1). \end{aligned} \tag{3.5}$$

Under these assumptions, a click does not necessarily indicate relevance, nor does a non-click necessarily indicate non-relevance. In general $c(d) \neq rel(q, d)$, and naively optimizing the empirical risk using clicks would be a suboptimal strategy [82]. Instead, it is necessary to correct for the observation probabilities $P(o(d) = 1)$, often called the *propensity*. This motivates the use of Inverse Propensity Scoring (IPS), where inversely weighing the propensities of clicked items can debias the click data.

For our work we assume that the propensities $P(o(d) = 1)$ are known a priori. In practice, the propensities are commonly estimated via A/B testing [172] or through intervention harvesting [6, 43], usually under some assumption of a user behavior model such as the position-bias model [32]. Modeling and estimating these propensity scores falls outside the scope of this chapter as our aim is to understand and improve the *convergence rate* of IPS-weighted optimization for LTR. We refer to existing work [6, 10, 24, 27, 43] for more detailed information about how the propensity scores can be modelled and/or estimated.

We now reach the main approach for Counterfactual LTR, which is to apply Inverse Propensity Scoring (IPS) to debias our optimization objective. Suppose we are given a *click log dataset* containing n clicks:

$$\mathcal{D} = \{(q_i, D_{q_i}, d_i, p_i)\}_{i=1}^{n}, \tag{3.6}$$

where:
- $q_i \sim P(q_i)$ is the issued query;
- D_{q_i} is the set of candidate items;
- $d_i \in D_{q_i}$ is the clicked item (i.e., $c(d_i) = 1$); and
- $p_i = P(o(d_i) = 1)$ is the propensity of item d_i.

Our goal now is to solve the following optimization problem:

$$\begin{aligned} \arg\min_{\boldsymbol{w}} R_{IPS}(\boldsymbol{w}) &= \arg\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i} \lambda(\mathit{rank}(d_i \mid q_i, D_{q_i}, f_{\boldsymbol{w}})) \\ &= \arg\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i} \mathcal{L}_i(\boldsymbol{w}). \end{aligned} \tag{3.7}$$

It is easy to show that $R_{IPS}(\boldsymbol{w})$ is an unbiased estimate of $R(\boldsymbol{w})$, i.e., that $\mathbb{E}[R_{IPS}(\boldsymbol{w})] = \mathbb{E}[R(\boldsymbol{w})]$, as long as $P(o(d) = 1) > 0$ for all d, using the proof of Section 4 in [82]. We note that the minimization problem in Equation 3.7 permits stochastic optimization and can be efficiently solved with a variety of SGD approaches. A common approach for counterfactual learning is to treat the inverse propensity scores as weights and multiply the gradient $\nabla \mathcal{L}_i(\boldsymbol{w})$ with $\frac{1}{p_i}$ to obtain an unbiased gradient estimate [5, 10, 72, 83]. We display this IPS-weighted SGD approach in Algorithm 4.

Algorithm 4 IPS-weighted Stochastic Gradient Descent (SGD)

$\boldsymbol{w}_1 = \mathbf{0}$
for $t \leftarrow 1, \ldots, T$ **do**
  $i_t \sim \mathit{Uniform}(0, n)$ ▷ sample i_t from uniform distribution
  $\boldsymbol{g}_t = \frac{1}{p_{i_t}} \nabla \mathcal{L}_{i_t}(\boldsymbol{w}_t)$ ▷ compute IPS-weighted gradient
  $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \boldsymbol{g}_t$ ▷ SGD update step
end for
$\bar{\boldsymbol{w}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}_t$
return $\bar{\boldsymbol{w}}$

Scaling the gradients with IPS weights, as is done in Algorithm 4, is a common technique for unbiased LTR [5, 9, 16, 72, 173]. It is known that high-variance gradients can cause poor convergence for general SGD-style algorithms [148]. However, to the best of our knowledge, the exact impact of IPS weights on the convergence rate of SGD algorithms for LTR has not been studied. In other words, the nature of the relationship between the IPS weights and the convergence rate of SGD algorithms is an open problem. To address this problem, we will analyze Algorithm 4 and provide results describing the relationship between IPS weights and the convergence rate in the next section.

3.4 Convergence of IPS-weighted SGD

In this section our aim is to better understand the convergence rate of Algorithm 4 and analyze the impact of IPS weights on the convergence rate.

3.4.1 Convergence rate analysis

Let $R_{IPS}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i} \mathcal{L}_i(\boldsymbol{w})$ be the function that we wish to minimize using Algorithm 4. We assume that each $\mathcal{L}_i(\boldsymbol{w})$ is convex and, consequently, that R_{IPS} is convex since each $\frac{1}{p_i} > 0$. In LTR this is a reasonable assumption because the loss can often be convex, for example the pairwise hinge loss formulation presented in Equation 3.2 is convex. Furthermore, the rank weighting functions $\lambda(\cdot)$ presented in Section 3.3.1 are convex for, e.g., the average relevant rank or DCG weighting schemes. Additionally, we assume R_{IPS} is minimized at some point $\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w}:\|\boldsymbol{w}\| \leq B} R_{IPS}(\boldsymbol{w})$. We denote with M the maximum IPS weight in the dataset: $M = \max_i \frac{1}{p_i}$. As in previous analyses of regular SGD [148], the goal of our analysis is to bound the suboptimality of the solution produced by Algorithm 4:

$$\mathbb{E}[R_{IPS}(\bar{\boldsymbol{w}}) - R_{IPS}(\boldsymbol{w}^*)]. \tag{3.8}$$

Theorem 3.4.1. Let $\mathcal{L}_i(\boldsymbol{w})$ be a convex function for each i and let $\boldsymbol{w}^*$ be a minimizer of $R_{IPS}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i} \mathcal{L}_i(\boldsymbol{w})$ such that $\|\boldsymbol{w}^*\| \leq B$. Assume that $\|\nabla \mathcal{L}_{i_t}(\boldsymbol{w}_t)\| \leq G$ for all t and let $\bar{\boldsymbol{w}}$ be the solution produced by running Algorithm 4 for T iterations

with $\eta = \sqrt{\frac{B^2}{(MG)^2 T}}$. Then:

$$\mathbb{E}[R_{IPS}(\bar{\boldsymbol{w}}) - R_{IPS}(\boldsymbol{w}^*)] \leq \frac{B(MG)}{\sqrt{T}}. \tag{3.9}$$

Proof. This convergence rate proof is a variant of the proof of Theorem 14.8 from [148], where we use IPS-weighted gradients $\boldsymbol{g}_t$ and construct a bound on the gradient variance in Equation 3.15. Since our notation deviates slightly from the notation in [148], we include the full proof here for clarity. Denote with $i_{1:T}$ the sequence of random indices $i_1, \ldots, i_T$, then:

$$\begin{aligned}
&\mathbb{E}_{i_{1:T}}[R_{IPS}(\bar{\boldsymbol{w}}) - R_{IPS}(\boldsymbol{w}^*)] \\
&= \mathbb{E}_{i_{1:T}}\left[R_{IPS}\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{w}_t\right) - R_{IPS}(\boldsymbol{w}^*)\right] && \text{(3.10a)} \\
&\leq \mathbb{E}_{i_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}R_{IPS}(\boldsymbol{w}_t) - R_{IPS}(\boldsymbol{w}^*)\right] && \text{(3.10b)} \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:T}}[R_{IPS}(\boldsymbol{w}_t) - R_{IPS}(\boldsymbol{w}^*)]. && \text{(3.10c)}
\end{aligned}$$

Here, (3.10a) follows from the definition of $\bar{\boldsymbol{w}}$, (3.10b) is due to Jensen's inequality [142], and, finally, (3.10c) is obtained by applying linearity of expectation. Since $\boldsymbol{w}_t$ depends only on the indices $i_{1:t-1}$ we get:

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:T}}[R_{IPS}(\boldsymbol{w}_t) - R_{IPS}(\boldsymbol{w}^*)] \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:t-1}}[R_{IPS}(\boldsymbol{w}_t) - R_{IPS}(\boldsymbol{w}^*)]. && \text{(3.11)}
\end{aligned}$$

Once the indices $i_{1:t-1}$ are known, the value of $\boldsymbol{w}_t$ is no longer random. Furthermore, since each i_t is uniformly sampled from the dataset, it follows that $\boldsymbol{g}_t$ is an unbiased estimate of $\nabla R_{IPS}(\boldsymbol{w}_t)$:

$$\mathbb{E}_{i_t}[\boldsymbol{g}_t \mid i_{1:t-1}] = \mathbb{E}_{i_t}[\boldsymbol{g}_t \mid \boldsymbol{w}_t] = \nabla R_{IPS}(\boldsymbol{w}_t). \tag{3.12}$$

Continuing from Equation 3.11, we have:

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:t-1}}[R_{IPS}(\boldsymbol{w}_t) - R_{IPS}(\boldsymbol{w}^*)] \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:t-1}}[\langle \boldsymbol{w}_t - \boldsymbol{w}^*, \nabla R_{IPS}(\boldsymbol{w}_t)\rangle] && \text{(3.13a)} \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{i_{1:t-1}}[\langle \boldsymbol{w}_t - \boldsymbol{w}^*, \mathbb{E}_{i_t}[\boldsymbol{g}_t \mid i_{1:t-1}]\rangle] && \text{(3.13b)}
\end{aligned}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{i_{1:t-1}} \mathbb{E}_{i_t}[\langle \boldsymbol{w}_t - \boldsymbol{w}^*, \boldsymbol{g}_t \rangle \mid i_{1:t-1}] \tag{3.13c}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{i_{1:t}}[\langle \boldsymbol{w}_t - \boldsymbol{w}^*, \boldsymbol{g}_t \rangle], \tag{3.13d}$$

where (3.13a) follows from the convexity of R_{IPS}, (3.13b) can be obtained by using Equation 3.12 ($\nabla R_{IPS}(\boldsymbol{w}_t) = \mathbb{E}[\boldsymbol{g}_t \mid i_{1:t-1}]$), (3.13c) is due to the linearity of expectation, and, finally, (3.13d) is obtained by applying the law of total expectation. We can now use Lemma 14.1 from [148] since Equation 3.13d is of the required form and obtain:

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{i_{1:t}}[\langle \boldsymbol{w}_t - \boldsymbol{w}^*, \boldsymbol{g}_t \rangle] \leq \frac{1}{T}\left(\frac{\|\boldsymbol{w}^*\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T} \|\boldsymbol{g}_t\|^2 \right). \tag{3.14}$$

Here is where we deviate from the standard proof of convergence for SGD and upper bound the IPS-weighted gradients $\boldsymbol{g}_t$ as follows:

$$\|\boldsymbol{g}_t\| = \left\| \frac{1}{p_{i_t}} \nabla \mathcal{L}_{i_t}(\boldsymbol{w}_t) \right\| = \frac{1}{p_{i_t}} \|\nabla \mathcal{L}_{i_t}(\boldsymbol{w}_t)\| \leq MG. \tag{3.15}$$

Next, we can plug this upper bound on $\|\boldsymbol{g}_t\|$ into Equation 3.14, and use the assumption that $\|\boldsymbol{w}^*\| \leq B$ to obtain:

$$\frac{1}{T}\left(\frac{\|\boldsymbol{w}^*\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T} \|\boldsymbol{g}_t\|^2 \right) \leq \frac{1}{T}\left(\frac{B^2}{2\eta} + \frac{\eta}{2} T (MG)^2 \right). \tag{3.16}$$

Finally, by plugging in $\eta = \sqrt{\frac{B^2}{(MG)^2 T}}$ and applying algebraic manipulations we obtain:

$$\frac{1}{T}\left(\frac{B^2}{2\eta} + \frac{\eta}{2} T (MG)^2 \right)$$

$$= \frac{1}{T}\left(\frac{B^2}{2\sqrt{\frac{B^2}{(MG)^2 T}}} + \frac{\sqrt{\frac{B^2}{(MG)^2 T}}}{2} T (MG)^2 \right) \tag{3.17a}$$

$$= \frac{1}{T}\left(\frac{B(MG)\sqrt{T}}{2} + \frac{B(MG)\sqrt{T}}{2} \right) \tag{3.17b}$$

$$= \frac{B(MG)\sqrt{T}}{T} \tag{3.17c}$$

$$= \frac{B(MG)}{\sqrt{T}}. \quad \square \tag{3.17d}$$

Theorem 3.4.1 combines several important quantities that determine how fast Algorithm 4 will converge. First, we have $1/\sqrt{T}$, which indicates that, as the number of

iterations T grows, the solution $\bar{\boldsymbol{w}}$ gets closer to the optimal solution $\boldsymbol{w}^*$. Second, we have B, which tells us how far away $\boldsymbol{w}^*$ is from the starting point $\mathbf{0}$. Clearly, for large B, the optimum $\boldsymbol{w}^*$ is far away and we require more iterations to converge. Finally, we have the gradient variance term (MG). When the gradients have potentially large variance, we need to correspondingly set a small learning rate to prevent divergent behavior and, as a result, it takes longer to converge. As a consequence of Theorem 3.4.1, it is clear that in order to achieve an error of at most ϵ, it suffices to run Algorithm 4 for T iterations where:

$$T \geq \frac{B^2(MG)^2}{\epsilon^2}. \tag{3.18}$$

3.4.2 Discussion

The key insight that our analysis provides is that, for IPS-weighted SGD algorithms, the number of iterations required to achieve an ϵ-optimal solution grows with a factor $(MG)^2$. We note that there are known variations of SGD that can improve the convergence rate presented in Theorem 3.4.1 from $O\left(1/\sqrt{T}\right)$ to $O(1/T)$, for example see [54, 55, 137, 149]. However, their analyses are considerably more complex and do not remove the dependency on $\|\boldsymbol{g}_t\|^2$, which for the IPS-weighted SGD case remains upper bounded by $(MG)^2$. This means that despite having faster convergence in terms of T, these methods do not improve the slowdown introduced by the IPS-weights.

Moreover, Agarwal et al. [3] show that, for strongly convex and Lipschitz smooth functions, the term $\|\boldsymbol{g}_t\|^2$, and consequently the term $(MG)^2$ in Equation 3.18, cannot be improved for *any SGD algorithm* (for sufficiently large T). In practice, the value of M can be very large, for example in cases with small propensities p_i (e.g., in situations with a significant amount of position bias such as when the candidate set D_q is very large). The above facts lead us to hypothesize that, as long as the gradients are scaled with IPS weights, the convergence rate is severely slowed by the magnitude of the IPS weights. Our experiments in Section 3.7 support this hypothesis.

3.5 Improved Convergence with Weighted Sampling

3.5.1 CounterSample: SGD with IPS-proportional sampling

Theorem 3.4.1 shows that the convergence of IPS-weighted SGD is slowed by a factor M^2. Moreover, as described in Section 3.4.2, as long as gradients $\boldsymbol{g}_t$ are scaled by IPS weights, this dependency on M cannot be improved [3]. Clearly, for situations where M is large, this can lead to slow convergence and make learning inefficient.

To overcome this problem, we propose a sampling-based SGD strategy. The key idea is to debias our optimization objective via *sampling* instead of *weighting*. As we will prove below, this sampling-based approach similarly guarantees unbiasedness of the optimization objective but has a better convergence rate. We call our approach CounterSample and it is displayed in Algorithm 5.

CounterSample functions as follows. First, we assign each data point i in our

Algorithm 5 COUNTERSAMPLE: SGD with IPS-proportional sampling

$\boldsymbol{w}_1 \leftarrow \mathbf{0}$
$\bar{M} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i}$
for $t \leftarrow 1, \ldots, T$ **do**
  $i_t \sim P(i_t \mid \mathcal{D})$ ▷ sample i_t according to Equation 3.19
  $\boldsymbol{g}_t = \bar{M} \nabla \mathcal{L}_{i_t}(\boldsymbol{w}_t)$ ▷ compute gradient
  $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}_t$ ▷ SGD update step
end for
$\bar{\boldsymbol{w}} \leftarrow \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}_t$
return $\bar{\boldsymbol{w}}$

dataset $\mathcal{D}$ the following probability of being sampled:

$$P(i \mid \mathcal{D}) = \frac{\frac{1}{p_i}}{\sum_{j=1}^{n} \frac{1}{p_j}}. \tag{3.19}$$

We then proceed exactly like regular SGD, where instead of sampling datapoints uniformly from the dataset, they are sampled using Equation 3.19. Furthermore, the algorithm does not scale the gradients by the IPS-weights, but by a constant factor:

$$\bar{M} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{p_i}, \tag{3.20}$$

which is necessary to guarantee unbiasedness (see Section 3.5.2).

In practice, one would tune the learning rate η in Algorithm 5, making the inclusion of $\bar{M}$ unnecessary as it merely scales the gradients by a constant, which can be offset by the particular η chosen. Therefore, for practical implementations, it is not necessary to include this constant. We include $\bar{M}$ here for the purposes of guaranteeing unbiasedness and analyzing the convergence rate.

3.5.2 Unbiasedness

To show that Algorithm 5 minimizes the unbiased objective $R_{IPS}(\boldsymbol{w})$, it is sufficient to show that, in expectation, the gradient $\boldsymbol{g}_t$ is an unbiased estimate of $\nabla R_{IPS}(\boldsymbol{w_t})$ for any t.

Theorem 3.5.1. Let $R_{IPS}(\boldsymbol{w})$ be the function to be optimized and let $\boldsymbol{g}_t$ be the gradient at time t as computed by Algorithm 5, then:

$$\mathbb{E}_{i_t}[\boldsymbol{g}_t \mid \boldsymbol{w}_t] = \nabla R_{IPS}(\boldsymbol{w_t}). \tag{3.21}$$

Proof. The proof uses the definition of $\boldsymbol{g}_t$ (3.22a), linearity of expectation (3.22b) and the definition of expectation (3.22c) where we use Equation (3.19) as the sampling probability. Using the definition of $\bar{M}$ and algebraic manipulations completes the proof

((3.22d) and (3.22e)):

$$\mathbb{E}_{i_t}[\boldsymbol{g}_t \mid \boldsymbol{w}_t] = \mathbb{E}\big[\bar{M}\ \nabla\mathcal{L}_{i_t}(\boldsymbol{w}_t)\big] \tag{3.22a}$$
$$= \bar{M}\ \mathbb{E}[\nabla\mathcal{L}_{i_t}(\boldsymbol{w}_t)] \tag{3.22b}$$
$$= \bar{M}\sum_{i=1}^{n} P(i \mid \mathcal{D})\nabla\mathcal{L}_i(\boldsymbol{w}_t) \tag{3.22c}$$
$$= \frac{1}{n}\left(\sum_{i=1}^{n}\frac{1}{p_i}\right)\sum_{i=1}^{n}\frac{\frac{1}{p_i}}{\sum_{j=1}^{n}\frac{1}{p_j}}\nabla\mathcal{L}_i(\boldsymbol{w}_t) \tag{3.22d}$$
$$= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_i}\nabla\mathcal{L}_i(\boldsymbol{w}_t) = \nabla R_{IPS}(\boldsymbol{w}_t). \quad \square \tag{3.22e}$$

3.5.3 Convergence rate

We wish to understand the convergence rate of the proposed method CounterSample (Algorithm 5). Similar to Section 3.4.1, the goal of our analysis is to bound the suboptimality of the solution produced by Algorithm 5:

$$\mathbb{E}[R_{IPS}(\bar{\boldsymbol{w}}) - R_{IPS}(\boldsymbol{w}^*)]. \tag{3.23}$$

Theorem 3.5.2. Let $\mathcal{L}_i(\boldsymbol{w})$ be a convex function for each i and let $\boldsymbol{w}^*$ be a minimizer of $R_{IPS}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_i}\mathcal{L}_i(\boldsymbol{w})$ such that $\|\boldsymbol{w}^*\| \leq B$. Assume that $\|\nabla\mathcal{L}_{i_t}(\boldsymbol{w}_t)\| \leq G$ for all t and let $\bar{\boldsymbol{w}}$ be the solution produced by running Algorithm 5 for T iterations with $\eta = \sqrt{\frac{B^2}{(\bar{M}G)^2 T}}$. Then:

$$\mathbb{E}[R_{IPS}(\bar{\boldsymbol{w}}) - R_{IPS}(\boldsymbol{w}^*)] \leq \frac{B(\bar{M}G)}{\sqrt{T}}. \tag{3.24}$$

Proof. First, we note that in Algorithm 5, the gradients are bounded as follows:

$$\|\boldsymbol{g}_t\| = \big\|\bar{M}\ \nabla\mathcal{L}_{i_t}(\boldsymbol{w}_t)\big\| \leq \bar{M}G. \tag{3.25}$$

Next, we follow the proof of Theorem 3.4.1, replacing Equation 3.15 with Equation 3.25, using the result of Theorem 3.5.1 to ensure $\mathbb{E}_{i_t}[\boldsymbol{g}_t \mid \boldsymbol{w}_t] = \nabla R_{IPS}(\boldsymbol{w_t})$ and plugging in $\eta = \sqrt{\frac{B^2}{(\bar{M}G)^2 T}}$ to give us the desired result. $\square$

As a result of Theorem 3.5.2, we can conclude that CounterSample provides significant advantages in terms of convergence rate over standard IPS weighting. Specifically, to obtain an error of at most ϵ, it is sufficient to run CounterSample for:

$$T \geq \frac{B^2(\bar{M}G)^2}{\epsilon^2} \tag{3.26}$$

iterations. This is strictly better than the bound that was obtained for Algorithm 4 in nearly all cases. The only case where the two methods have the same convergence rate is when $\bar{M} = M$, which can only happen when all the propensity scores are

the same, i.e., when $p_i = p_j$ for all i, j. However, this can only be the case when the click log itself is already unbiased, thus negating the need to do counterfactual learning in the first place. Therefore, for any practical Counterfactual LTR scenario, COUNTERSAMPLE is strictly better in terms of convergence rate than naively scaling the gradients with IPS weights.

3.5.4 Efficiency

Finally, despite the advantages of COUNTERSAMPLE in terms of convergence rate, these benefits may not be useful if they come at the cost of worse computational complexity. A straightforward but naive implementation for sampling from Equation 3.19 would result in a $O(Tn)$ time complexity for Algorithm 5, which is significantly worse than the $O(T)$ complexity obtained by standard SGD approaches such as Algorithm 4.

Fortunately, sampling from Equation 3.19 can be done with an amortized $O(1)$ cost using the alias method [170, 171]. To achieve this, there is a one time cost of constructing the alias table, done in $O(n)$. Furthermore, we also need to compute the constant $\bar{M}$ which is also a one time operation of $O(n)$. We note that both of these steps can take place during data pre-processing and are for most practical implementations easily achieved (e.g., PyTorch [127] and Tensorflow [1] both support efficient sampling from a weighed multinomial distribution using the alias method). Overall, this means that the complexity of COUNTERSAMPLE is $O(n + T)$, which is acceptable when $n < T$.

3.5.5 Illustrative example

To illustrate the difference in convergence rates between COUNTERSAMPLE and standard IPS-weighted SGD we have created a simple toy example learning problem in Figure 3.1. We chose two optimal weights $\boldsymbol{w}^* = [w_1, w_2]$ and synthesize an IPS-weighted regression dataset.[1] Notice that, for large learning rates, the IPS-weighted SGD approach leads to unstable learning and diverges from the optimum. The learning rate for IPS-weighted SGD needs to be small enough to ensure that training samples with large IPS weights do not cause too large a step, possibly leading to divergent behavior. As a result, it is necessary to reduce the learning rate to ensure stable learning, however doing so naturally increases the time until convergence for IPS-weighted SGD because samples that do not have extreme IPS weights can only make small progress to the optimum. On the other hand, COUNTERSAMPLE can reliably handle large learning rates because the gradients are not scaled with potentially large IPS weights. Instead, COUNTERSAMPLE samples training instances with high IPS weights more frequently. Overall, this leads to much faster convergence.

[1]The synthesized dataset comprises 50 training samples: $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{50}\}$ where each $\boldsymbol{x}_i = [x_{i,1}, x_{i,2}]$ and each $x_{i,j} \sim \mathcal{N}(0, 1)$. We choose $\boldsymbol{w}^* = [0.973, 1.144]$ and set targets $y_i = \langle \boldsymbol{x}_i, \boldsymbol{w}^* \rangle$. For each $\boldsymbol{x}_i$ we generate a propensity $p_i \sim \textit{Uniform}(0.05, 1.0)$ and use the IPS-weighted squared loss as our optimization objective: $R_{\textit{IPS}}(\boldsymbol{w}) = \frac{1}{50} \sum_{i=1}^{50} \frac{1}{p_i} (\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle - y)^2$.

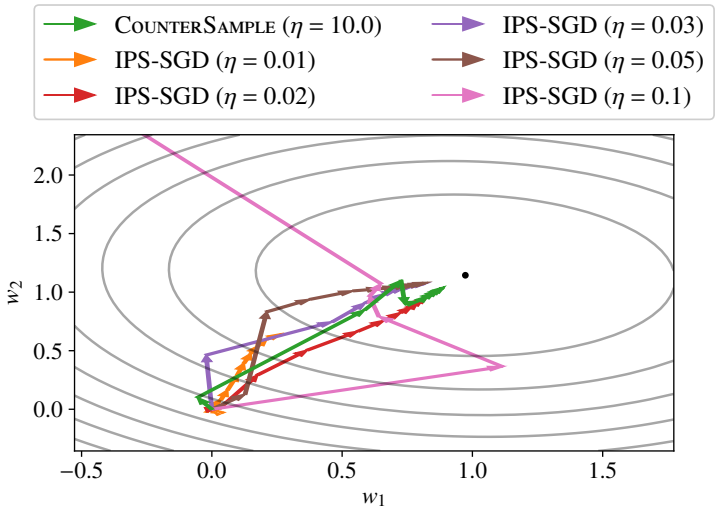

Figure 3.1: Illustration of the convergence of CounterSample versus IPS-weighted SGD on a synthetic learning example with two weights w_1 and w_2. The algorithms are run for $T = 50$ iterations. Best viewed in color.

Table 3.1: Datasets used for our experiments.

Dataset	Queries	Avg. docs per query	$rel(q, d) = 0$
Yahoo [28]	36,251	23	26%
Istella-s [106]	33,118	103	89%

3.6 Experimental setup

Our experimental setup is aimed at assessing the convergence rate of CounterSample and to answer the following research question:

> **RQ2** Can counterfactual learning from user interactions be made more efficient?

We use the standard experimental setup for Counterfactual LTR, first described in [82]. This means that we use a *fully supervised* LTR dataset and simulate a biased *click log* according to a position-based user behavior model.

3.6.1 Datasets

We use two supervised LTR datasets in our experiments: Yahoo [28] and Istella-s [106]. We choose these two datasets as they complement each other in the number of items per query, which is large for Istella-s and small for Yahoo, and the sparsity of relevance feedback, which is high for Istella-s and low for Yahoo (see Table 3.1).

Both LTR datasets are collected on a large set of queries $\mathcal{Q} = \{q_1, \ldots, q_m\}$. For each query q the dataset provides a set of candidate items D_q, where each item $d \in D_q$ is given in the form of a feature vector $x_{(q,d)}$ representing a query-item pair. Furthermore, for each query q the relevance grades $rel(q, d)$ are known for all $d \in D_q$. The relevance grades are scaled from 0 to 4, where 0 indicates no relevance and 4 indicates highly relevant. We note that this violates the binary relevance assumption made in Section 3.3.1. However, as we will see in Section 3.6.2, during click simulation the relevance grades are reduced to binary form which is in line with existing experimental setups for Counterfactual LTR [82].

3.6.2 Simulation setup

We simulate clicks using the setup of [82]. In this setup, we repeatedly sample a query q uniformly from the dataset. The candidate items D_q for the sampled query are then sorted by a scoring function f_0, called the *logging policy* (see Section 3.6.3 for how f_0 is chosen). The simulation introduces position bias: items that are highly ranked by f_0 have a higher probability of being observed and thus clicked. Furthermore, the simulation has some noise: for every observed item, the probability of it being clicked is 1 if the item is relevant ($\textit{rel}(q, d) \in \{3, 4\}$) and 0.1 if the item is not relevant ($\textit{rel}(q, d) \in \{0, 1, 2\}$). More formally, for every item $d \in D_q$, clicks are sampled from a Bernoulli distribution with probability:

$$P(c(d) = 1) = \begin{cases} P(o(d) \mid q, D_q, f_0) & \text{if } \textit{rel}(q, d) \in \{3, 4\}, \\ P(o(d) \mid q, D_q, f_0) \cdot 0.1 & \text{if } \textit{rel}(q, d) \in \{0, 1, 2\}, \end{cases} \tag{3.27}$$

where

$$P(o(d) \mid q, D_q, f_0) = \left(\frac{1}{\textit{rank}(d \mid q, D_q, f_0)} \right)^{\gamma}, \tag{3.28}$$

and $\gamma \geq 0$ is a parameter controlling the severity of position bias. The above formulation is identical to the setup used in [82]. For all our experiments we simulate 1,000,000 clicks. Unless otherwise specified, we use $\gamma = 1$ as the position bias parameter. Table 3.2 shows the values of M and $\bar{M}$ for the simulated clicks on each of the datasets. We note that even under mild position bias ($\gamma = 0.5$), there is a significant difference between M and $\bar{M}$. The difference between these quantities becomes substantially larger as γ increases.

Simulating clicks from supervised datasets has several advantages over using existing click logs. First, by simulating clicks we can explicitly control the severity of position bias (by controlling the value of γ) and therefore test its impact in a controlled environment. Second, we can evaluate the learned rankers on the *true relevance labels* as they are provided by the supervised datasets. This means that we do not have to resort to performance estimation techniques that may be unreliable.

3.6.3 Choice of logging policy

We need to build a logging policy f_0 that can be used to rank items for our click simulation. A good candidate logging policy is one that can produce rankings of sufficient

Table 3.2: $\bar{M}$ and M for varying levels of position bias (γ).

Position bias (γ):	0.5	0.75	1.0	1.25	1.5
Yahoo: $\bar{M}$	3.14	5.12	7.92	11.71	16.66
Yahoo: M	11.79	40.70	129.00	388.91	1265.55
Istella-s: $\bar{M}$	4.21	7.42	12.02	18.12	25.94
Istella-s: M	13.04	48.53	177.00	645.60	2081.04

quality to generate a useful number of relevant clicks, but not perfectly optimal so that learning can still occur. To do so we train a linear ranker with full supervision (using the pairwise hinge loss formulation of Equation 3.3) on 0.1% of the queries for each of the datasets. Building a logging policy in this manner represents a realistic deployment scenario: practitioners of LTR systems would typically train a ranker on a small amount of manually annotated data before deploying it to collect a large amount of click data.

3.6.4 Evaluation

To measure the performance of the rankers learned by the various algorithms, we use nDCG@10 [75] on held-out test data. We denote with $\textit{nDCG@10}(\boldsymbol{w})$, the average nDCG@10 on held-out test data when items are ranked using the scoring function $f_{\boldsymbol{w}}$.

We are interested in measuring the *convergence rate* of the learning algorithms. To do so, we measure average regret in terms of nDCG@10 (with respect to the optimal model $\boldsymbol{w}^*$):

$$\textit{Regret}(T) = \frac{1}{T} \sum_{t=1}^{T} \left(\textit{nDCG@10}(\boldsymbol{w}^*) - \textit{nDCG@10}(\bar{\boldsymbol{w}}_t) \right), \tag{3.29}$$

where $\bar{\boldsymbol{w}}_t = \frac{1}{t} \sum_{t'=1}^{t} \boldsymbol{w}_{t'}$ is the learned model after t iterations. To obtain the gold standard $\boldsymbol{w}^*$ we train a linear ranker with full supervision (using the relevance labels of the LTR dataset).[2]

Measuring regret should help us confirm our theoretical results about convergence rates since lower values indicate faster convergence to the optimal solution $\boldsymbol{w}^*$. For each of our results we consider statistical significance with a t-test ($p < 0.01$).

3.6.5 Methods to compare

In our experiments we compare the following methods:

- COUNTERSAMPLE: our sample-based method (Algorithm 5);
- IPS-SGD: IPS-weighted SGD (Algorithm 4) [5, 82]; and
- Biased-SGD: naive SGD without any propensity weighting.

[2] We assume that $\textit{nDCG@10}(\boldsymbol{w}^*) \geq \textit{nDCG@10}(\boldsymbol{w}')$ for any $\boldsymbol{w}'$ that $\bar{\boldsymbol{w}}_t$ may converge to.

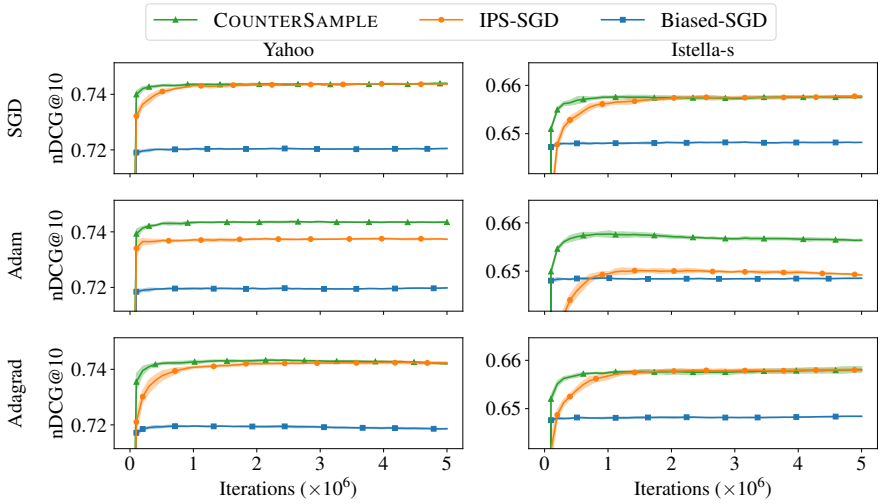

Figure 3.2: The learning performance on held-out test data for different optimizers (left column is Yahoo, right column is Istella-s). CounterSample is significantly faster to converge in all scenarios. For the Adam optimizer, the IPS-weighted SGD approach produces a suboptimal performance.

We use a linear scoring function for our experiments, as this more closely matches the convexity assumptions made in our analysis:

$$f_{\boldsymbol{w}}(q, d) = \langle \boldsymbol{w}, x_{q,d} \rangle. \tag{3.30}$$

For each experiment and each method we tune the learning rate η to minimize regret (see Section 3.6.4) on held-out validation data, where we try the following values of η:

$$\eta \in \{1 \times 10^{-10}, 3 \times 10^{-10}, 1 \times 10^{-9}, \ldots, 1 \times 10^{0}, 3 \times 10^{0}\}. \tag{3.31}$$

3.7 Experimental Results

3.7.1 Effect of optimizer

First, we investigate the impact of the optimizer on the convergence rate of the different Counterfactual LTR approaches. We consider three commonly used optimization methods: Regular SGD, Adam [92] and Adagrad [39]. We apply these methods by replacing the update rule in Algorithms 4 and 5 with either the update rule from Adam or Adagrad.

In Figure 3.2 we plot the learning curves on held-out test data for both the Yahoo and Istella-s dataset. Interestingly, IPS-SGD does not work well with Adam, converging to a suboptimal solution, whereas CounterSample is able to converge to a

Table 3.3: Average regret ($\times 100$) for different optimizers. Smaller values indicate faster convergence. Statistically significantly lower and higher regret compared to IPS-SGD is denoted with $\triangledown$ and $\triangle$ respectively.

Optimizer:	SGD	ADAM	ADAGRAD
Yahoo			
Biased-SGD	$2.64^{\triangle}$	$2.72^{\triangle}$	$2.76^{\triangle}$
IPS-SGD	0.41	0.97	0.64
COUNTERSAMPLE	$0.33^{\triangledown}$	$0.35^{\triangledown}$	$0.44^{\triangledown}$
Istella-s			
Biased-SGD	$2.07^{\triangle}$	$2.04^{\triangledown}$	$2.06^{\triangle}$
IPS-SGD	1.33	2.16	1.24
COUNTERSAMPLE	$1.19^{\triangledown}$	$1.24^{\triangledown}$	$1.15^{\triangledown}$

much higher level of performance. This result is surprising as both COUNTERSAMPLE and IPS-SGD optimize the same unbiased objective. Recent work has shown that ADAM is not guaranteed to converge to the optimal solution for some convex optimization problems and this may in part explain the behavior we observe here [138]. Regardless, we observe that in all cases COUNTERSAMPLE converges significantly faster than IPS-SGD. We note that the naive Biased-SGD approach converges to a lower level of performance, which is as expected since it ignores the impact of position bias. We confirm these findings by reporting the average regret in Table 3.3, observing a significantly lower regret for COUNTERSAMPLE than IPS-SGD.

Our results indicate that COUNTERSAMPLE is superior to IPS-weighting across all optimizers. We find that regular SGD outperforms other optimizers in the majority of cases. A possible reason for this behavior is that our scoring function is linear. Optimizers such as ADAM and ADAGRAD may not provide significant benefits over SGD when applied to linear functions as opposed to non-linear functions (e.g., deep neural networks). We leave studying other scoring functions such as neural networks as future work.

3.7.2 Impact of batch size

In this section we investigate the effect of the batch size. We hypothesize that large batch sizes reduce the variance of individual update steps, as many gradients are averaged in a single update step, and as a result the convergence rate of COUNTERSAMPLE and IPS-SGD should be comparable. We try batch sizes 10, 20 and 50.

We plot the learning curves for varying batch sizes in Figure 3.3. Once again we find that, unsurprisingly, Biased-SGD converges to a suboptimal solution. For both datasets we observe that COUNTERSAMPLE is able to converge faster than IPS-SGD, regardless of the chosen batch size. For the Istella-s dataset, COUNTERSAMPLE is able to converge to a slightly higher level of performance than IPS-SGD when using a batch size of 50. The average regret in Table 3.4 suggests that the convergence rate of COUNTERSAMPLE is not affected by batch size; it is able to converge faster than IPS-SGD in all cases.

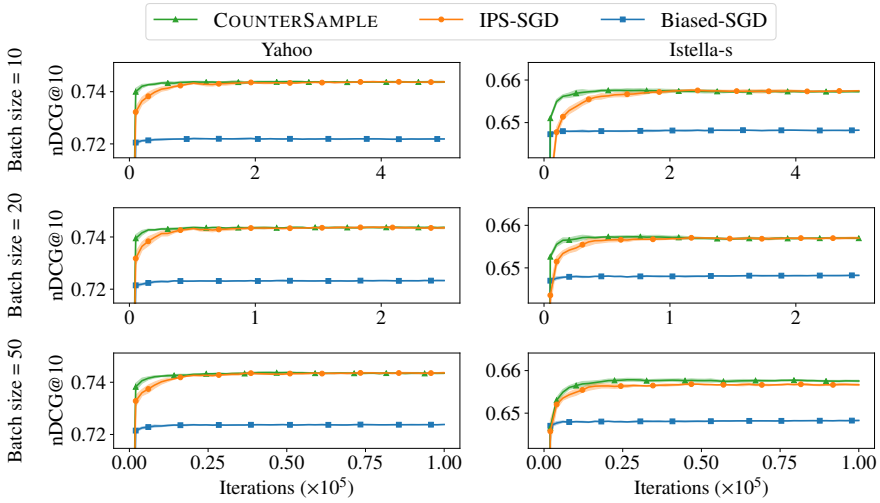

Figure 3.3: The learning performance on held-out test data for different batch sizes (left column is Yahoo, right column is Istella-s). CounterSample is faster to converge in all scenarios, however the differences are less pronounced for larger batch sizes.

Table 3.4: Average regret ($\times 100$) for different batch sizes. Statistical significance is denoted the same as Table 3.3.

Batch size:	10	20	50
Yahoo			
Biased-SGD	$2.49^{\triangle}$	$2.36^{\triangle}$	$2.31^{\triangle}$
IPS-SGD	0.41	0.42	0.44
CounterSample	$0.33^{\triangledown}$	$0.34^{\triangledown}$	$0.37^{\triangledown}$
Istella-s			
Biased-SGD	$2.07^{\triangle}$	$2.08^{\triangle}$	$2.07^{\triangle}$
IPS-SGD	1.34	1.31	1.32
CounterSample	$1.20^{\triangledown}$	$1.21^{\triangledown}$	$1.21^{\triangledown}$

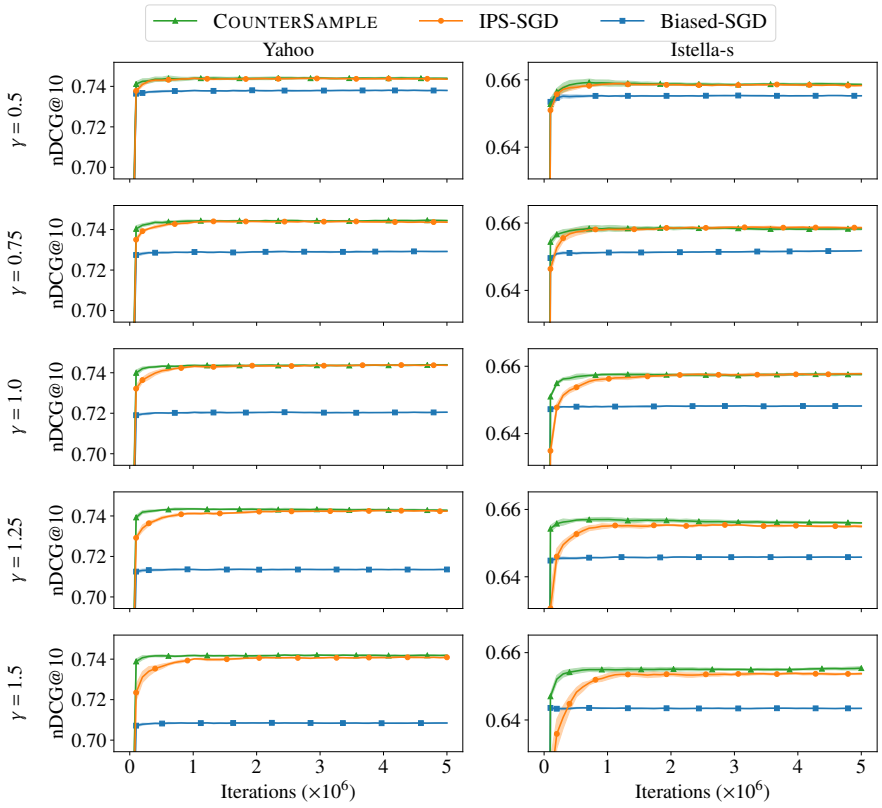

Figure 3.4: The learning performance on held-out test data for varying levels of position bias (left column is Yahoo, right column is Istella-s). CounterSample's convergence rate is robust to larger values of γ, whereas IPS-SGD suffers when γ is large.

3.7.3 Severity of position bias

Finally, we look at the impact of position bias, controlled by the position bias parameter γ. Position bias has an effect on the nature of the clicks collected and changes the distribution of propensity scores (see Table 3.2). For large γ, the propensity scores will be heavily skewed: the majority of observations and propensities will be on the top-ranked items while lower-ranked items are only very rarely observed and clicked, resulting in more extreme IPS weights for those clicks. Conversely, a small γ makes the propensity scores more heavy tailed, generating more observations on lower ranked items and consequently more clicks on those items with less extreme IPS weights. We expect that, as we increase γ, CounterSample should outperform IPS-SGD in terms of convergence rate since in this case $M \gg \bar{M}$. Conversely, for smaller values of γ we expect that the methods perform comparably.

Table 3.5: Average regret ($\times 100$) for different levels of position bias γ. Statistical significance is denoted the same as Table 3.3.

Position bias (γ):	0.5	0.75	1.0	1.25	1.5
Yahoo					
Biased-SGD	$0.89^{\triangle}$	$1.78^{\triangle}$	$2.64^{\triangle}$	$3.32^{\triangle}$	$3.83^{\triangle}$
IPS-SGD	0.34	0.35	0.41	0.56	0.75
CounterSample	0.30	$0.27^{\triangledown}$	$0.33^{\triangledown}$	$0.38^{\triangledown}$	$0.51^{\triangledown}$
Istella-s					
Biased-SGD	$1.37^{\triangle}$	$1.75^{\triangle}$	$2.07^{\triangle}$	$2.30^{\triangle}$	$2.53^{\triangle}$
IPS-SGD	1.09	1.12	1.33	1.53	1.79
CounterSample	1.05	1.08	$1.19^{\triangledown}$	$1.26^{\triangledown}$	$1.44^{\triangledown}$

Figure 3.4 provides learning curves for various levels of γ. We observe that the performance of Biased-SGD goes up as γ goes down, which is in line with our expectations since smaller values of γ result in less position bias. In all cases IPS-SGD and CounterSample perform strictly better than Biased-SGD. The convergence of CounterSample is comparable to IPS-SGD for smaller γ, but as γ grows, CounterSample is significantly faster to converge than IPS-SGD. This confirms our expectation that CounterSample is able to reliably handle situations where $M \gg \bar{M}$, i.e. when there are more extreme IPS weights. Table 3.5 confirms these findings in terms of average regret: for larger values of γ, CounterSample is able to obtain significantly lower regret than competing approaches.

3.7.4 Discussion

Finally, we reflect on RQ2:

> Can counterfactual learning from user interactions be made more efficient?

We answer our research question positively: CounterSample consistently converges faster than IPS-SGD – across optimizers, batch sizes and different levels of position bias (γ). These findings support the theoretical results we obtained in Sections 3.4 and 3.5. In some scenarios, for example when using the Adam optimizer, CounterSample is not only able to converge faster but able to converge to a higher level of performance than IPS-SGD.

3.8 Conclusion

In this chapter, we have studied the convergence rate for Stochastic Gradient Descent (SGD) approaches in Counterfactual Learning to Rank (LTR). A common approach to Counterfactual LTR is IPS-weighted SGD, where the loss or gradients are scaled by IPS weights. We prove that, for IPS-weighted SGD, the IPS weights play an important

role in the convergence rate: the time to converge is slowed by a factor $O(M^2)$ where M is the maximum IPS weight in the dataset.

To overcome the slow convergence of IPS-weighted SGD, we propose a sample-based Counterfactual LTR learning algorithm called COUNTERSAMPLE. We prove that COUNTERSAMPLE reduces the convergence rate slowdown from $O(M^2)$ to $O(\bar{M}^2)$, where $\bar{M}$ is the average IPS weight in the dataset. When $M \gg \bar{M}$, this improvement leads to significantly faster convergence of the learning algorithm.

We support our theoretical findings with extensive experimentation across a number of biased LTR scenarios, comparing COUNTERSAMPLE to SGD with and without IPS weighting. In all cases COUNTERSAMPLE is able to converge faster than standard IPS-weighted SGD. In some scenarios COUNTERSAMPLE is even able to converge to a better level of performance than IPS-weighted SGD.

There are several directions for future work: First, the convexity assumptions made in the analysis may not hold in practice, particularly when implementing deep neural networks. Showing the convergence rate of IPS-weighted SGD for non-convex problems remains an open problem. Second, optimizing an IPS-weighted objective is arguably the simplest approach to Counterfactual LTR and in future work we would like to consider more sophisticated objectives such as self-normalized IPS [157] and variance regularization [156]. Third, our experiments are conducted on click simulations, giving us experimental control to test our hypotheses. We leave applying COUNTERSAMPLE to large-scale industrial click logs as future work. Fourth, our work assumes that propensity scores are known a priori which is not always realistic. Robustness against misspecified propensity scores remains an open problem. Finally, in this Chapter we only consider *position bias* and do not consider the impact of *selection bias*: the phenomenon where users only interact with a subset of the ranked items, for example only the top 10. Recall that in Chapter 2 we found that online LTR algorithms can more effectively deal with selection bias. Furthermore, we found that interventions play a big role in overcoming selection bias. In Chapter 4 we will introduce a counterfactual learning algorithm that also performs interventions but does so safely: without harming the user experience.

4

Safe Exploration for Optimizing Contextual Bandits

4.1 Introduction

In this chapter we consider the problem of *safe exploration*. Recall from Chapter 2 that counterfactual LTR may not perform well in cases where selection bias is present. Selection bias is a phenomenon where users only interact with a fixed subset of the ranked list (for example the top 10). We found that performing interventions by frequently deploying the learned ranker, effectively exploring new rankings of documents, allowed counterfactual LTR approaches to overcome selection bias. However, we also saw that blindly deploying a learned ranker may lead to suboptimal performance in situations with a significant amount of click noise. In this chapter we study how interventions can *safely* be performed by counterfactual learning algorithms in the context of the contextual bandit framework.

A *multi-armed bandit* problem is a problem in which a limited number of resources must be allocated between alternative choices so as to maximize their expected gain. In *contextual* bandit problems, when making a choice, a representation of the context is available to inform the decision. Contextual bandit problems are a natural framework to capture a range of IR tasks [2, 49, 57]. Example tasks to which contextual bandits have been applied include news recommendation [100], text classification [156], ad placement [95], and online learning to rank [56]. For example, in online LTR (see Figure 4.1): (1) a user issues a query; (2) a search engine presents a ranked document list; and (3) clicks on the documents are recorded as feedback. The interactive nature of this problem makes it an ideal application for the contextual bandit framework. The LTR model is a decision-making *policy* π, the ranked document lists it produces are *actions*, the user is the *context* or *environment*, and clicks are the *reward* signal.

There are two major classes of algorithms to maximize the reward of a contextual bandit policy π. The first are online algorithms, which optimize a policy while it is being executed [7, 84, 100, 176]. The second are counterfactual algorithms [81, 155, 156], which optimize a policy based on data that was collected using an existing logging policy, often called π_0 [151, 156].

This chapter was published as [74].

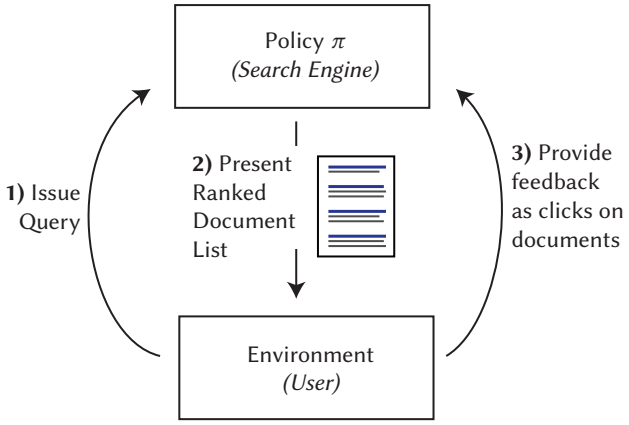

Figure 4.1: Online learning to rank viewed as a contextual bandit problem. A user issues a query, the search engine responds with a ranked list of documents, and the user provides implicit feedback in the form of clicks on documents.

Online learning methods for contextual bandits are widely studied and there are many known algorithms to solve this problem. E.g., popular algorithms include ϵ-greedy, LinUCB [100] and Thompson Sampling [7]. Despite their attractive properties, the adoption of online learning methods for contextual bandits in production systems has been limited. Especially in the early stages of learning, online algorithms may perform actions that are suboptimal and, thus, hurt the user experience. E.g., in online LTR it is risky to present suboptimal rankings of documents [117, 173].

Counterfactual learning from logged feedback [51, 59, 151, 156] has been proposed as a solution to this problem. Using an existing and already deployed logging policy, one only takes actions from that logging policy to collect bandit feedback. Using this collected data, a new policy is then learned offline in an unbiased manner. This learning process is *safe* in the sense that the new policy is not executed and, thus, the user experience is not affected. The drawback of counterfactual learning, however, is that it relies on the deployed logging policy, which never changes by design. Due to this, there may be areas in the action space left unexplored, i.e., bandit feedback may be missing for those areas. In IR terms, potentially high quality document rankings might never be presented to users and the corresponding user interactions will never be observed.

In this chapter, we propose a method that performs *exploration* of the action space in contextual bandit problems, but does this exploration *safely*, that is, without hurting the user experience. In our proposed solution, called Safe Exploration Algorithm (SEA), we use an already deployed logging policy which is safe, static but suboptimal, as a *warm-start* for learning a new policy. As soon as SEA is confident that the performance of the new policy does not fall below that of the logging policy, it starts executing actions from the new policy, thus, exploring the action space. In the context of LTR, this means that SEA is capable of exploring and presenting new rank-

ings to users, gathering feedback for rankings that might otherwise have never been presented. This enables SEA to trade off various strengths and weaknesses of both *online* and *counterfactual* learning.

We note that periodic deployment of a policy, after a successfull evaluation on held-out data, is a common practice in industry. In our experiments we include a boundless version of SEA, called BSEA, which represents such a deployment pipeline. What SEA adds over and above a standard deployment procedure is: (1) it comes with a formal proof of safety, i.e., with high probability the performance of SEA will be at least as good as that of a baseline policy (Section 4.4.4); and (2) we introduce a computationally efficient manner for computing high-confidence off-policy bounds (Section 4.4.3).

The main research question we address in this chapter is:

> **RQ3** Can counterfactual approaches perform interventions without harming the user experience?

This chapter addresses the above research question by dividing it into three subquestions:

RQ3.1 Is SEA safe? I.e., does it always perform at least as good as a baseline policy?

RQ3.2 Does SEA provide a better user experience during training than counterfactual or online learning methods? I.e., does it accumulate a higher reward during training?

RQ3.3 Does SEA, which explores the action space, learn a more effective policy compared to purely counterfactual learning methods, which do not perform exploration?

Our key technical contributions in this chapter are: (1) we introduce SEA and show it to be safe: its performance never falls below that of a baseline policy; (2) we show that SEA improves the user experience: it provides higher cumulative reward during training than both counterfactual and online methods; and (3) we show that a policy learned with SEA, which is capable of exploring new actions, outperforms that of purely counterfactual methods, which are incapable of exploration.

4.2 Related Work

The idea of deploying automated decision making systems in IR is not new. The contextual bandit framework has been used in news recommendation [100], ad placement [95], and online learning to rank [56]. Contextual bandit formulations of IR problems allow insights and methods from the bandit literature to be applied and extended to address these problems [57]. For research on contextual bandits this connection has opened up an application area where new approaches can be evaluated on large-scale datasets [101].

4.2.1 Learning in contextual bandits

Contextual bandit algorithms have been widely studied in the *online* and *counterfactual* learning settings [7, 100, 156]. A key challenge in contextual bandit problems is the exploration-vs-exploitation tradeoff. On the one hand we want to explore new actions so as to find favorable rewards. On the other hand, we want to exploit existing knowledge about actions so as to maximize the total reward. The online learning methods we consider in this chapter deal with this tradeoff in various ways. The methods we consider are policy gradient with ϵ-greedy exploration, policy gradient with Boltzmann exploration, LinUCB and Thompson Sampling. Policy gradient methods optimize the weights of a policy via stochastic gradient descent, by solving the following optimization problem:

$$\min_{\pi_w} -\sum_{t=1}^{T} \log\left(\pi_w(a_t \mid \boldsymbol{x}_t)\right) \cdot r_t. \tag{4.1}$$

The ϵ-greedy heuristic selects actions by choosing with probability ϵ an action uniformly at random and with probability $(1-\epsilon)$ the best possible action. Boltzmann exploration [26] chooses actions by drawing them with a probability proportional to the policy: $a_t \sim \pi_w(\cdot \mid \boldsymbol{x}_t)$. LinUCB [100] and Thompson Sampling [7] are different from policy gradient methods because they make a linearity assumption. These methods construct a set of weights $w_a \in \mathbb{R}^m$ for every possible action, such that $\pi_w(a_t \mid \boldsymbol{x}_t) = w_{a_t}^T \boldsymbol{x}_t$. LinUCB selects actions by choosing the one with the highest confidence bound. Thompson Sampling samples new weights $\hat{w}_a$ from a posterior distribution and then chooses the best action given the sampled weights: $a_t = \arg\max_a \pi_{\hat{w}_a}(a \mid \boldsymbol{x}_t)$.

In IR there has been considerable attention for exploiting log data [59]. It is one of the most ubiquitous forms of data available, as it can be recorded from a variety of systems at little cost [155]. The interaction logs of such systems typically contain a record of the input to the system, the prediction made by the system, and the feedback. The feedback provides only partial information limited to the particular prediction shown by the system. Counterfactual learning algorithms tell us how data collected from interaction logs of one system can be used to optimize a new system [57, 155]. Interaction logs used in counterfactual learning are usually biased towards the policy that collected the data. To remove this bias, counterfactual learning methods resort to inverse propensity scoring. To use inverse propensity scoring, the logging policy must be stochastic and the corresponding propensity scores (probability of choosing the logged action) are recorded. Using these propensity scores, it is possible to reweigh data samples to remove the bias. An advantage of counterfactual learning methods is that they do not require interactive experimental control. Thus, there is no risk of hurting the user experience with these methods. In other words, counterfactual learning methods are safe by definition [72].

Unlike *online* methods, SEA's performance during the early stages of learning does not suffer and always stays at least as good as a baseline, making the method safe to use. Compared to *counterfactual* methods, SEA is capable of exploration, which makes it effective at finding areas of the action space that may have high reward.

4.2.2 Safety in contextual bandits

There has been a growing interest in concepts related to safety for contextual bandit problems. This is due to the fact that contextual bandit formulations are applied to automated decision making systems, where actions taken by the system can have a significant impact on the real world. There are two main groups of work: *risk-aware* methods and *conservative* methods.

Risk-aware methods [46, 65, 153] are online learning algorithms that model the risk associated with executing certain actions as a cost which is to be constrained and minimized. Galichet et al. [46] introduce the concept of risk-awareness for the multi-armed bandit framework. Sun et al. [153] extend the idea of risk-awareness to the adversial contextual bandit setting. Garcia and Fernández [47] explore risk-awareness for the reinforcement learning paradigm. Risk-aware methods use a separate type of feedback signal, in addition to the standard reward feedback, which is called *risk*. Risk-aware methods aim to keep the cumulative risk below a specific threshold. These types of methods are typically applied in fields like robotics where certain actions can be dangerous or cause damage. A drawback is that the risk has to be explicitly quantified by the environment. Designing a good risk feedback signal for IR systems is subject to modeling biases and in some cases impossible, limiting the application of such methods.

Conservative methods [180] measure safety as a policy's performance relative to a baseline policy. A method is safe if its performance is always within some margin of the baseline policy. The idea was first introduced for the multi-armed bandit case [180] and later extended to linear contextual bandits in the form of the CLUCB algorithm [89]. CLUCB is a safe conservative online learning method, which works by constructing confidence sets around the parameters of the policy. Unfortunately, the method has to solve a constrained optimization problem every time an action has to be selected, which has a significant computational overhead. In contrast, SEA addresses this problem because it only needs to do two computationally efficient operations: update a lower confidence bound estimate and perform a gradient update step. This makes SEA applicable to larger and more complex datasets.

Finally, Li et al. [99] introduce a complementary approach to SEA, called BubbleRank, an algorithm that gradually improves upon an initial ranked list by exchanging higher-ranked less attractive items for lower- ranked more attractive items. Li et al. define a safety constraint that is based on incorrectly-ordered item pairs in the ranked list, and prove that BubbleRank never violates this constraint with a high probability. We do not compare to BubbleRank in our experiments because it assumes a user model [30], an assumption that is orthogonal to our experimental setup [72]. Likewise, Thomas et al. [164] propose a method for policy improvement similar to SEA. However, their work focuses on the reinforcement learning domain whereas our work is tailored to the contextual bandit and LTR settings. Furthermore, we propose an efficient method for computing the high-confidence bounds (Section 4.4.3).

4.3 Background

4.3.1 Contextual bandits

In online LTR we try to optimize the parameters of a ranking model such that it places relevant items at the top of the ranked list. We can view the ranking model as a decision-making policy. When a new query arrives, the policy takes an action: it displays one possible ranked list to the user. The user then decides to click on documents shown in the ranked list, thus providing a reward signal. More formally, we consider the following contextual bandit framework. At each round t:

1. The environment announces an m-dimensional *context vector* $\boldsymbol{x}_t \in \mathcal{X}$. In IR terms, this would be a user (environment) issuing a query (context vector).
2. A policy π samples an action $a_t \in \mathcal{A}$, one of n possible actions, conditioned on $\boldsymbol{x}_t$: $a_t \sim \pi(\cdot \mid \boldsymbol{x}_t)$. In IR terms, this would be a ranking model (policy) producing a ranked list (action).
3. The environment announces only the reward r_{t,a_t} for the chosen action a_t, and not for other possible actions that the policy could have taken. We assume that $r \in [0, 1]$. In IR terms, this reward signal could be clicks on documents.

This learning setup is inherently different from supervised learning, where rewards for all possible actions are known. We only observe rewards for actions that the policy has taken. The setting is referred to as *partial-label problem* [86], *associative bandit problem* [152] or *associative reinforcement learning* [84]. In the context of LTR, this means that we do not know the optimal ranked list but can only observe a reward signal for rankings that our policy chooses to display.

There are two major classes of algorithms for learning a contextual bandit policy. The first are online algorithms that optimize the policy while it is being executed [7, 84, 100, 176]. Algorithms of this class explore the space of possible actions but may harm the user experience, because suboptimal actions could be executed. The second are counterfactual algorithms, which optimize a policy based on data that was collected using an existing logging policy [151, 156]. Algorithms of this class are safe as the newly learned policy is not executed and, thus, there is no risk of hurting the user experience. However, counterfactual algorithms are incapable of exploring the action space, which may harm the performance of the learned model. In this work we build on the second class of algorithms; see below.

4.3.2 Counterfactual learning from logged bandit feedback

Counterfactual learning algorithms collect bandit feedback using an existing logging policy, which we call a *baseline policy* π_b (also referred to as π_0 in the literature [83, 156, 158]). This feedback is collected in the following form, where at time t we have:

$$\mathcal{D}_t = \{(\boldsymbol{x}_i, a_i, r_{i,a_i}, p_i)\}_{i=1}^{t}\,, \tag{4.2}$$

where $\boldsymbol{x}_i$ is the observed context vector, a_i is the action taken by the baseline policy, $r_{i,a_i} \in [0, 1]$ is the reward given by the environment, and p_i is the propensity score

for action a_i. The propensity score is the probability of the baseline policy taking the logged action, that is, $p_i = \pi_b(a_i \mid \boldsymbol{x}_i)$ [156].

To learn a new policy π_w from the collected bandit feedback, the following maximization problem has to be solved [83]:

$$\widehat{\pi}_w = \max_{\pi_w} \frac{1}{t} \sum_{i=1}^{t} \frac{r_{i,a_i}}{p_i} \pi_w(a_i \mid \boldsymbol{x}_i). \tag{4.3}$$

This optimization problem is solved via SGD: if a new tuple $(\boldsymbol{x}_i,\ a_i,\ r_{i,a_i},\ p_i)$ is observed, we weigh the derivative of $\pi_w(a_i \mid \boldsymbol{x}_i)$ by $\frac{r_{i,a_i}}{p_i}$ and update the weights of π_w using SGD. We refer to this counterfactual learning method as IPS [83].

4.3.3 High-confidence off-policy evaluation

In a counterfactual learning setting, a newly learned policy π_w is never executed, so its performance cannot be measured directly. Instead, we *estimate* its performance using the collected bandit feedback $\mathcal{D}_t$ at time t. To do so, we assume that $\pi_b(a_i \mid \boldsymbol{x}_i) \neq 0$ whenever $\pi_w(a_i \mid \boldsymbol{x}_i) \neq 0$. The estimated reward of a policy π can then be written as [101]:[1]

$$\hat{R}(\pi, \mathcal{D}_t) = \frac{1}{t} \sum_{i=1}^{t} \hat{R}_i = \frac{1}{t} \sum_{i=1}^{t} \frac{r_{i,a_i}}{p_i} \pi(a_i \mid \boldsymbol{x}_i). \tag{4.4}$$

This estimate suffers from high variance, especially when the propensity scores are small. To resolve this issue, Thomas et al. [163] propose several high-confidence off-policy estimators. These estimators first calculate a confidence interval around the policy's performance and then use the lower bound on this performance for off-policy evaluation. The high-confidence off-policy estimator that we use is based on the Maurer & Pontill empirical Bernstein inequality. The confidence bound can be written as [163]:

$$CB(\pi, \mathcal{D}_t) = \frac{7b \ln\left(\frac{2}{\delta}\right)}{3(t-1)} + \frac{1}{t} \sqrt{\frac{\ln\left(\frac{2}{\delta}\right)}{t-1} \sum_{i,j=1}^{t} (\hat{R}_i - \hat{R}_j)^2}, \tag{4.5}$$

where $(1-\delta) \in [0, 1]$ is the confidence level and b is an upper bound on $\hat{R}$. When both r and p are bounded, b can be calculated exactly: $b = \frac{\max r}{\min p}$. The lower confidence bound on the performance of a policy at time t, $LCB(\pi, \mathcal{D}_t)$, can be computed as:

$$LCB(\pi, \mathcal{D}_t) = \frac{1}{t} \sum_{i=1}^{t} \hat{R}_i - CB(\pi, \mathcal{D}_t). \tag{4.6}$$

Similarly, we compute the *upper* confidence bound on the performance of a policy as follows:

$$UCB(\pi, \mathcal{D}_t) = \frac{1}{t} \sum_{i=1}^{t} \hat{R}_i + CB(\pi, \mathcal{D}_t). \tag{4.7}$$

[1] We use $\hat{R}_i$ here to denote the estimated reward of a policy on the ith data sample. This is different from the R_i used in Chapter 2 which represents a document at rank i in the ranked list.

The estimator described in this section provides an effective way to compute a lower and upper bound on the performance of a policy without executing it.

In this chapter we build on the estimators designed by Thomas et al. [163], but provide the following contributions: First, we develop SEA, an algorithm for automatic safe exploration by deploying new models (Section 4.4.2); second, we provide an efficient way to compute the high-confidence bounds (Section 4.4.3); and, third, we formally prove that SEA is indeed safe (Section 4.4.4).

4.4 Safe Exploration Algorithm (SEA)

We define the notion of *safety* as part of the learning process (Section 4.4.1), walk through the Safe Exploration Algorithm (SEA) (Section 4.4.2), address the problem of efficient off-policy evaluation (Section 4.4.3), formally prove the safety of SEA (Section 4.4.4), and then analyse SEA (Section 4.4.5).

4.4.1 Safety

We use the definition of *safety* from so-called conservative methods [89, 180]. These methods make a key assumption: there exists a baseline policy π_b whose actions can always be executed without risk. This assumption is very reasonable in practice. In most industrial settings there exists a production system that we can consider to be the baseline policy. A learning algorithm is considered *safe* if its performance at any round t is at least as good as the baseline policy. Thus, safety is a concept that is always measured relative to a baseline.

More formally, let us first consider the notion of regret at round t during training. We define the *regret* at time t as the cumulative difference in reward obtained by executing actions from our policy π compared to a perfect policy π^*, one that always chooses the action with maximum reward. For notational simplicity, we denote the action that a policy chooses in response to a context vector $\boldsymbol{x}$ as $\pi(\boldsymbol{x})$:

$$\mathit{Regret}_t(\pi) = \sum_{i=1}^{t} (r_{i,\pi^*(\boldsymbol{x}_i)} - r_{i,\pi(\boldsymbol{x}_i)}). \tag{4.8}$$

A policy π is considered *safe* if its regret is always at most as large as that of the safe baseline policy π_b, i.e., for every $t = 1, \ldots, T$:

$$\mathit{Regret}_t(\pi) \leq \mathit{Regret}_t(\pi_b) \tag{4.9}$$

or

$$\sum_{i=1}^{t} (r_{i,\pi^*(\boldsymbol{x}_i)} - r_{i,\pi(\boldsymbol{x}_i)}) \leq \sum_{i=1}^{t} (r_{i,\pi^*(\boldsymbol{x}_i)} - r_{i,\pi_b(\boldsymbol{x}_i)}) \tag{4.10}$$

so

$$\frac{1}{t} \sum_{i=1}^{t} r_{i,\pi(\boldsymbol{x}_i)} \geq \frac{1}{t} \sum_{i=1}^{t} r_{i,\pi_b(\boldsymbol{x}_i)}. \tag{4.11}$$

In other words, at every time t, we want the average reward of our policy π to be at least as large as the average reward of the baseline policy π_b. In practice, we cannot observe the average reward of a policy without executing it. This is problematic because we cannot know if a policy is safe until we execute it. Fortunately, the off-policy estimators described in Section 4.3.3 provide a way to *estimate* the performance of a policy without executing it.

4.4.2 A walkthrough of SEA

The Safe Exploration Algorithm (SEA) learns a new policy π_w *offline* from the output of a baseline policy π_b using counterfactual learning techniques described in Section 4.3.2. At each iteration t, SEA estimates the performance of the newly learned policy π_{w_t} and the currently deployed policy π_d using the high-confidence off-policy evaluators described in Section 4.3.3. The new policy is only deployed online when its estimated performance is above that of the existing deployed policy. This allows the newly learned policy to take over and start exploring. This only happens once SEA is confident enough that the new policy's performance will be satisfactory and safe to execute.

Algorithm 6 Safe Exploration Algorithm (SEA)

1: π_b *// Baseline policy (current production system)*
2: $\pi_{w_0} \leftarrow \pi_b$ *// Policy to be learned (initialized with baseline weights)*
3: $\pi_d \leftarrow \pi_b$ *// The deployed policy that is executing actions*
4: $\mathcal{D}_0 \leftarrow \{\}$
5: **for** $t = 1, \ldots, T$ **do**
6: $\quad \boldsymbol{x}_t \leftarrow$ contextual feature vector at time t
7: $\quad a_t \sim \pi_d(\cdot \mid \boldsymbol{x}_t)$
8: $\quad p_t \leftarrow \pi_d(a_t \mid \boldsymbol{x}_t)$
9: $\quad$ Play a_t and observe reward r_{t,a_t}
10: $\quad \mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, a_t, r_{t,a_t}, p_t)\}$
11: $\quad w_t \leftarrow w_{t-1} + \eta \frac{r_{t,a_t}}{p_t} \nabla_w \pi_{w_{t-1}}(a_t \mid \boldsymbol{x}_t)$ *// Update weights via gradient ascent*
12: $\quad$ Compute $LCB(\pi_{w_t}, \mathcal{D}_t)$ using Equation 4.6
13: $\quad$ Compute $UCB(\pi_d, \mathcal{D}_t)$ using Equation 4.7
14: $\quad$ **if** $LCB(\pi_{w_t}, \mathcal{D}_t) \geq UCB(\pi_d, \mathcal{D}_t)$ **then**
15: $\quad\quad \pi_d \leftarrow \pi_{w_t}$ *// Deploy new policy only when it is safe to do so*
16: $\quad$ **end if**
17: **end for**

The pseudocode for the Safe Exploration Algorithm (SEA) is provided in Algorithm 6. SEA starts from a baseline policy π_b (Line 1) and a new policy that we wish to optimize, π_{w_0} (Line 2). At the start, SEA deploys the policy π_d, which is initialized to the baseline policy (Line 3). At every iteration t, a context vector $\boldsymbol{x}_t$ is observed (Line 6). SEA draws an action from the deployed policy and computes its propensity score (Lines 7–8). SEA executes the chosen action a_t and observes the reward r_t (Line 9). The policy π_w is then updated via SGD (Line 11). We then update the confidence bounds on the estimated performance for the new policy π_{w_t} and the deployed

policy π_d (Lines 12–13). When the estimated lower bound performance of π_{w_t} is better than the estimated upper bound performance of π_d (Line 14), we deploy π_{w_t}, such that future actions are executed by the newly learned policy instead of the previous deployed policy π_d.

4.4.3 Efficient policy evaluation

To implement SEA we use the off-policy estimator described in Section 4.3.3. Such an implementation will be computationally expensive, because at each round T we have to compute a sum over all t data points collected so far. E.g., in Equation 4.6, we need to compute the mean

$$\frac{1}{t}\sum_{i=1}^{t}\hat{R}_i$$

and the variance

$$\frac{1}{t}\sum_{i,j=1}^{t}\left(\hat{R}_i-\hat{R}_j\right)^2,$$

at every round t. Therefore, the complexity of applying Equation 4.6 or 4.7 would be $\mathcal{O}(T)$ and the complexity of Algorithm 6 would be $\mathcal{O}\left(T^2\right)$.

Recall that $\mathcal{D}_t = \{(\mathbf{x}_i, a_i, r_{i,a_i}, p_i)\}_{i=1}^t$ is the collected log data and that there are $|\mathcal{A}|$ possible actions and $|\mathcal{X}|$ possible contexts. We can then compute the mean $\frac{1}{t}\sum_{i=1}^t \hat{R}_i$ as follows (similar results hold for computing the variance term):

$$\frac{1}{t}\sum_{i=1}^{t}\frac{r_{i,a_i}}{p_i}\pi(a_i \mid \boldsymbol{x}_i) = \frac{1}{t}\sum_{a\in\mathcal{A}}\sum_{\boldsymbol{x}\in\mathcal{X}}\sum_{i=1}^{t}\mathbf{1}\left[a_i = a \wedge \boldsymbol{x}_i = \boldsymbol{x}\right]\frac{r_{i,a_i}}{p_i}\pi(a \mid \boldsymbol{x}) \tag{4.12}$$

$$= \frac{1}{t}\sum_{a\in\mathcal{A}}\sum_{\boldsymbol{x}\in\mathcal{X}}\pi(a \mid \boldsymbol{x})\sum_{i=1}^{t}\mathbf{1}\left[a_i = a \wedge \boldsymbol{x}_i = \boldsymbol{x}\right]\frac{r_{i,a_i}}{p_i}, \tag{4.13}$$

where $\mathbf{1}[\cdot]$ is the indicator function.

The key insight here is that the inner sum, $\sum_{t=1}^{T}\mathbf{1}\left[a_t = a \wedge \boldsymbol{x}_t = \boldsymbol{x}\right]\frac{r_{t,a_t}}{p_t}$ can be efficiently computed online during data collection and is independent of the policy π_w that is being evaluated. Specifically, we can create a zero-initialized matrix $W \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{X}|}$ where, each time a new data point $(\boldsymbol{x}_i, a_i, r_{i,a_i}, p_i)$ is logged, we update an entry as follows:

$$W_{a_i,\boldsymbol{x}_i} \leftarrow W_{a_i,\boldsymbol{x}_i} + \frac{r_{i,a_i}}{p_i}.$$

This results in the following method for computing the mean:

$$\frac{1}{t}\sum_{a\in\mathcal{A}}\sum_{x\in\mathcal{X}}\pi(a \mid x)W_{a,x} = \frac{1}{t}\sum_{(a,x):W_{a,x}\neq 0}\pi(a \mid x)W_{a,x}, \tag{4.14}$$

which can be orders of magnitude faster to compute when $T \gg |\mathcal{A}| \cdot |\mathcal{X}|$. This is not unreasonable in practice, as the size of interaction logs is usually much larger than the number of users and items. In addition, we only need to compute the above sum for a and x for which $W_{a,x} \neq 0$. This enables the use of sparse data structures that can speed up computation even further.

4.4.4 Proof of safety

SEA is an *online* learning method that we claim to be *safe*. To show that SEA is actually safe, we bound the probability that a suboptimal policy, one whose expected reward is lower than the expected reward of the deployed policy, will be deployed during the duration of the algorithm.

Theorem 4.4.1. At any time t, with probability at least $1 - 2\delta$, SEA will not deploy a suboptimal policy.

Proof. For notational simplicity we write the expected reward of a policy π as:

$$R_t(\pi) = \mathbb{E}_{a\sim\pi}\left[\frac{1}{t}\sum_{i=1}^{t} r_{i,a}\right]. \tag{4.15}$$

First, from Algorithm 6 we know that we only deploy π_{w_t} if $LCB(\pi_{w_t}) > UCB(\pi_d)$. Suppose that our confidence bounds do not fail. In this case, it is impossible to deploy a suboptimal policy, since

$$R_t(\pi_{w_t}) \geq LCB(\pi_{w_t}) > UCB(\pi_d) \geq R_t(\pi_d). \tag{4.16}$$

Consequently, a suboptimal policy can only be deployed when either the confidence bound estimate on our newly learned policy π_{w_t} or our deployed policy π_d fails.

The high-confidence off-policy estimators (see Equations 4.6 and 4.7) provide a lower and upper bound on the estimated performance. According to Theorem 1 in [163], these confidence bounds hold with probability at least $1 - \delta$. Conversely, this means that the confidence bounds may fail with probability at most δ:

$$P\left(R_t(\pi) \notin [LCB(\pi), UCB(\pi)]\right) \leq \delta, \tag{4.17}$$

and, as a result, the probability that either the confidence bound on π_{w_t} or π_d will fail is at most 2δ:

$$P\left(R_t(\pi_{w_t}) \notin [LCB(\pi_{w_t}), UCB(\pi_{w_t})] \vee R_t(\pi_d) \notin [LCB(\pi_d), UCB(\pi_d)]\right) \leq 2\delta. \tag{4.18}$$

Therefore, with probability at least $1 - 2\delta$, we will not deploy a suboptimal policy. □

In our experiments (see Section 4.5) we set $\delta = 0.05$, which means that the algorithm is safe with probability at least $1 - 2\delta = 0.90$ at any time t. We observe that this lower bound is fairly loose because we do not observe a single suboptimal deployment across many repetitions and possible deployment moments.

4.4.5 Analysis

Conservative Linear Contextual Bandits (CLUCB) [89] is, to our knowledge, the only other online learning method for contextual bandits that is also safe. Unfortunately, CLUCB comes with two limitations:

1. CLUCB does not scale beyond toy problems because it constructs confidence sets around parameters, which requires solving a constraint optimization problem every time an action has to be chosen which makes it infeasible for realistic IR problems; and

2. CLUCB can only be applied to linear models.

Our method addresses both limitations. First, SEA scales to large and complex datasets, because it merely needs to compute the lower confidence bound and perform a gradient update step, both of which can be done highly efficiently. Second, SEA can easily be adapted to non-linear models such as gradient-boosted decision trees and neural networks as it is based on gradient descent.

SEA, being an online learning method, is capable of exploration. In contrast, *counterfactual* methods do not explore, they merely observe what a baseline policy is doing. If this baseline policy does not explore well, counterfactual learning techniques, whilst still able to learn, are less effective [156]. And even if the baseline policy is highly explorative, what truly matters is how well this policy explores the regions with favorable losses [124]. SEA solves this problem as it starts exploring actions using the newly learned policy as soon as it is safe to do so. Since the policy learned by SEA has a higher estimated performance than the baseline policy, the actions that it eventually takes are likely to be actions with high reward.

We note that the safety that SEA guarantees does come at a cost: SEA cannot guarantee that it will explore new actions beyond the initial deployed policy. In contrast, purely *online* methods can explore without any restrictions and as a result they may end up learning a better policy than SEA. Put differently, SEA provides a trade-off between safety and exploration: with a lower δ, SEA will be more safe, but gives up some amount of exploration. Vice versa, a higher δ allows SEA to explore more aggressively while giving up some level of safety.

4.5 Experimental Setup

To answer our research questions, we consider two tasks: *text classification* and *document ranking*. We consider these two complementary tasks as they differ in the size of the action space, which is small in the case of text classification (the number of classes) but large in the case of document ranking (the number of possible ranked lists). In both cases we turn a supervised learning problem into a bandit problem.

4.5.1 Text classification task

For *text classification* we use a dataset $\mathcal{D} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^{T}$, where $\boldsymbol{x}_t \in \mathbb{R}^m$ is a feature representation of the object we wish to classify and $y_t \in \{0, \ldots, n\}$ is the correct label for that object. E.g., in text classification $\boldsymbol{x}_t$ is a bag-of-words representation of a document we wish to classify and y_t the correct label for that document. We are interested in the contextual bandit formulation for multi-class classification, i.e., where the correct label y_t is not known, but we can observe a reward signal for a chosen label a_t.

Table 4.1: Datasets used for the text classification task.

Dataset	Classes	Features	Train size	Test size
USPS [64]	10	256	7,291	2,007
20 Newsgroups [94]	20	62,060	15,935	3,993
RCV1 [98]	53	47,236	15,564	518,571

We follow the methodology of Beygelzimer and Langford [18] to transform a supervised learning problem into a contextual bandit problem:

1. The environment presents the policy with the feature vector $\boldsymbol{x}_t$ (e.g., the bag-of-words representation of a text document).
2. The policy chooses a_t, one of n possible labels, as its prediction.
3. The environment returns a reward of 1 if the chosen label was correct and a reward of 0 if it was incorrect.

Methods that use counterfactual learning (IPS and SEA), require a stochastic baseline policy and the corresponding propensity scores, i.e., the probability that the baseline policy chose the selected label a_t. To this end we define our data-collection policy using the ϵ-greedy approach. An ϵ-greedy approach offers several benefits over alternative exploration strategies:

1. The amount of exploration can be manually tuned with the ϵ parameter, allowing either very conservative or very aggressive exploration.
2. It is a priori known, via the ϵ parameter, how much performance we are giving up to perform exploration.
3. The ϵ-greedy strategy generates stable and bounded propensity scores; this counteracts the high variance problem that is common in counterfactual learning and evaluation.

We consider two types of reward signal. First, the perfect scenario where a reward of 1 is given if the chosen label a_t is correct and 0 otherwise. Second, a near-random scenario where rewards are sampled from a Bernoulli distribution. More specifically, when a policy chooses the correct label a_t, the policy is given a reward of 1 with probability 0.6. Conversely, if the chosen label a_t is incorrect, the learner is given a reward of 1 with probability 0.4. We choose these contrasting scenarios as they highlight the need for safety. It is much easier for a learner to make mistakes in the near-random scenario due to the high levels of noise and we hypothesize that safety plays a more important role in that case. The text classification datasets that we use are specified in Table 4.1.

4.5.2 Document ranking task

For the *document ranking* task we use the counterfactual LTR framework as described in [82]. Table 4.2 details the datasets used for the document ranking task. In this

Table 4.2: Datasets used for the document ranking task.

Dataset	Queries	Features	Avg. docs per query
MSLR-WEB10K [130]	10,000	136	124
Yahoo Webscope [28]	36,251	700	23
Istella-s [106]	33,118	220	103

setup, a production ranker displays a ranked list to the user. It is assumed that the user does not examine all documents in the presented ranking, but is instead more likely to observe top-ranked documents than lower ranked ones, a phenomenon called "position bias" [79]. After a user observes a document they can either judge it as relevant, by clicking on it, or judge it as non-relevant, by not clicking on it.

We simulate user behavior as follows:

1. The policy being learned presents a ranked list to the simulated user.
2. The simulated user samples a set of observed documents from the ranked list, where the probability of observing the document at rank i is

$$p_i = \left(\frac{1}{i}\right)^{\gamma}, \tag{4.19}$$

   where γ is a parameter that controls the severity of click bias. This setup is identical to the one described in [82].
3. For each observed document, we generate clicks depending on the relevance of the document. We use varying levels of click noise ("perfect", "position-biased" and "near-random"), the specific click probabilities are listed in Table 4.3. This setup is in line with the methodologies described in [56, 60, 117, 122].

In our setup, clicks are simulated only on the top 10 documents. This more realistically simulates the behavior of web search users who are unlikely to visit the second (or later) result page [29]. For the counterfactual models we assume the correct propensity scores are known during learning. In other words, the propensity model used to simulate position bias is also used to compute the propensity scores during learning.

We update our model via stochastic gradient descent at the document level by updating the weights using a weighted gradient and stochastic gradient descent, as described in Section 4.3.2.

4.5.3 Methods used for comparison

Our experiments are aimed at assessing the performance of SEA for solving contextual bandit problems. We compare SEA against counterfactual and online methods. The former comparison is motivated by the fact that counterfactual methods are safe by definition so a comparison of SEA against such methods will inform us about the potential performance gains by using SEA. The latter is motivated by the fact that SEA is an online method so a comparison of SEA against such methods will inform us about

Table 4.3: Click noise settings for the document ranking task. Each entry is the probability of clicking a document given its relevance label.

	$P(click \mid relevance)$				
	0	1	2	3	4
Perfect	0.00	0.20	0.40	0.80	1.00
Position-biased	0.10	0.10	0.10	1.00	1.00
Near random	0.40	0.45	0.50	0.55	0.60

the safety of using SEA. In all experiments, the models to be trained are warm-started with the weights of the deployed policy.

Counterfactual methods for contextual bandit problems optimize a policy by observing actions that are taken by a separate logging policy π_b. This makes them safe, because learning happens only on data that has been collected in the past by a logging policy. However, safety also makes them incapable of exploration. We use the state-of-the-art counterfactual method λ-translated Inverse Propensity Scoring (λ-IPS) [83]. We hypothesize that the average reward of a model trained by SEA is higher than of a model trained by λ-IPS because SEA is capable of exploring actions from the newly learned policy.

Online methods optimize a policy π_w by having it interact with the environment. Such methods are effective at exploring the action space but have no notion of safety. The online methods we use for the classification task are:

1. policy gradient with an ϵ-greedy strategy,
2. policy gradient with a Boltzmann exploration strategy,
3. LinUCB [100], and
4. Thompson Sampling [7].

For the ranking task, we use the following online methods:

1. SVMRank Online [76], and
2. Dueling Bandit Gradient Descent [184] with Team-Draft Interleaving [136].

We hypothesize that the user experience of online methods suffers in the early stages of learning (i.e., performance will be below the baseline policy π_b) because these approaches do not provide formal guarantees of safety while exploring new actions.

Safe online methods are meant to be safe in the sense that during training, the algorithms never perform worse than a baseline policy π_b. Our contribution, SEA, falls in this class of methods. We also consider CLUCB (Conservative Linear UCB) [89]. Its design makes it impractical for problems with high dimensionality or large action spaces as it requires performing an $m \times m$ matrix inversion and solving a constrained optimization problem whenever an action has to be selected; its high computational complexity prevents us from using it with our datasets.

Finally, we also include an empirically safe version of SEA, which does not use the high-confidence bounds, which we denote as Boundless Safe Exploration Algorithm (BSEA). This method compares the mean estimate of the reward of the learned policy π_w and the deployed policy π_d, instead of the lower and upper confidence bounds. Specifically, we use Algorithm 6 where we set

$$UCB(\pi, \mathcal{D}_t) = LCB(\pi, \mathcal{D}_t) = \frac{1}{t} \sum_{i=1}^{t} \frac{r_{i,a_i}}{p_i} \pi(a_i \mid \boldsymbol{x}_i). \tag{4.20}$$

We expect this method to deploy its learned model earlier and more frequently than SEA because it does not have to overcome potentially large confidence bounds. Hence, we expect this policy to do better than SEA, but it may exhibit unsafe behavior as it no longer provides the same formal safety guarantees as SEA.

4.5.4 Choice of baseline policy

Both λ-IPS and SEA depend on a baseline policy π_b. We require that this baseline policy is sub-optimal, so that learning can occur. This requirement is reasonable since we cannot hope to improve an already optimal policy. We introduce suboptimality in π_b by subsampling the training set on which we train the policy (1% sample for the classification task and 0.1% sample for the ranking task). This is motivated by a scenario that commonly occurs in real search engines or classification systems: Manual labels are expensive to obtain and are usually available on a scale of several orders of magnitude smaller than logged bandit feedback. This strategy for introducing suboptimality results in a baseline policy whose performance is much better than random, but still not optimal.

4.5.5 Metrics and statistical significance

We evaluate SEA and the competing approaches in two ways. One is in terms of *cumulative reward* during training, which is the sum of all rewards received as a function of the number of rounds; this type of metric allows us to quantify the degree to which the user experience is affected. A higher cumulative reward during training indicates a better user experience. The second is in terms of *average reward*, which is the reward averaged per action on held-out test data; this type of metric allows us to quantify how well a trained policy generalizes to unseen test data. The document ranking task uses simulated clicks as a reward signal (see Section 4.5.2) during training, which is not a very insightful metric for evaluating the true performance of a policy. Instead, to *evaluate* the learned policies, we use nDCG@10 [75]. We measure statistical significance of observed differences using a t-test ($p < 0.01$).

4.6 Results

4.6.1 Safety

We first address RQ3.1:

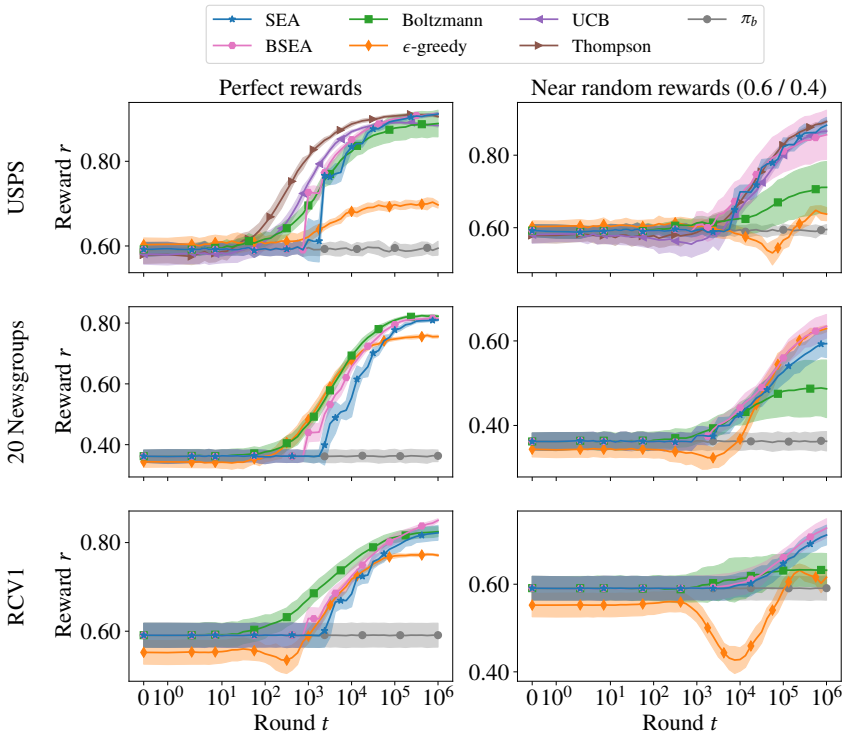

Figure 4.2: The performance of SEA (blue line) compared to *online* algorithms for the text classification task. The top row indicates the USPS dataset, the middle row the 20 Newsgroups dataset and the bottom row the RCV1 dataset. The performance is measured on a held-out test set after each round t. Shaded areas indicate standard deviation. Best viewed in color.

> Is SEA safe? I.e., does it always perform at least as good as a baseline policy?

To answer RQ3.1, we plot the performance of SEA against that of *online* algorithms while they are training, using average reward on a held-out test dataset as our metric.

Let us first consider the text classification task; see Figure 4.2. The baseline policies have an average accuracy of around 0.6, except in the 20 Newsgroups dataset where this is 0.4. The tested online algorithms include ϵ-greedy, Boltzmann exploration, Thompson sampling and LinUCB. For the 20 Newsgroups and RCV1 datasets we do not run LinUCB and Thompson sampling because they require inverting a $47{,}236 \times 47{,}236$ matrix (for 20 Newsgroups) and $62{,}060 \times 62{,}060$ matrix (for RCV1) on every update which is too computationally expensive. The performance of SEA is at least as good as the baseline policy. Similarly, it seems that warm-started online algorithms are also safe in this scenario, as they too perform always at least as good as the baseline policy. The best algorithm in terms of final performance is different for each of the

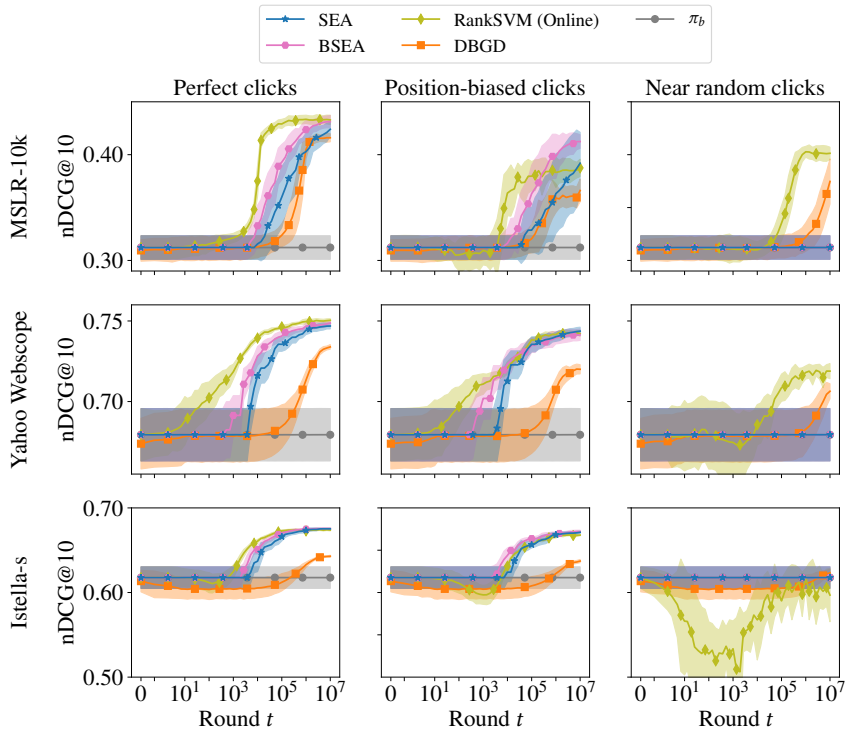

Figure 4.3: The performance of SEA (blue line) compared to an *online* approach (orange line) for the document ranking task on MSLR-10k (top row), Yahoo Webscope (middle row) and Istella-s (bottom row) with varying levels of click noise. The performance of the trained ranker is measured on a held-out test set after each round t. Shaded areas indicate standard deviation. Best viewed in color.

datasets. Overall we find that when it is computationally feasible to apply UCB or Thompson sampling, they outperform all other methods. Furthermore we find that Boltzmann exploration works very well with a perfect reward signal. However, in the case of near random rewards, both Boltzmann exploration and ϵ-greedy are not always capable of learning a good policy. Finally, we observe that BSEA is on par with SEA, and in some cases significantly outperforms it, while being empirically safe. BSEA is capable of deploying its learned model faster and more frequently than SEA as it does not have to overcome potentially large confidence bounds. Although this means that BSEA is not guaranteed to be safe, we find that it is empirically safe across all experimental conditions.

Next, we consider the document ranking setting; see Figure 4.3. We use nDCG@10 on held-out test data as the evaluation metric. We observe that SEA is always at least as good as the baseline policy whereas online learning methods may suffer from unsafe performance in the early stages of learning. We see that in the case of near-random clicks on the Istella-s dataset, SEA accurately identifies that it is not safe to deploy

whereas the online method suffers from unsafe performance and actually go below the baseline level. However, on the other datasets with near-random clicks, SEA never deploys a new model and consequently does not explore different actions. As a result, it is unable to improve upon the production policy. *Online* approaches outperform SEA here because they explore more aggressively and are able to learn even in the face of large amounts of noise. These results are in line with previous work [72], which has shown that counterfactual methods have difficulty learning in scenarios with large amounts of noise. In cases with position-biased or perfect clicks, the ranker trained by SEA performs on par with the online method. This means that in realistic scenarios we do not sacrifice any performance by using SEA. Finally, we note that BSEA performs empirically safe on the document ranking task across all experimental settings, which is in line with the results on the text classification task.

This answers RQ3.1. Our experiments indicate that SEA is indeed safe, its performance does not fall below that of the baseline policy.

4.6.2 Improved user experience

Next, we answer RQ3.2:

> Does SEA provide a better user experience during training than counterfactual or online learning methods? I.e., does it accumulate a higher reward during training?

To answer RQ3.2, we measure the cumulative reward obtained by the learning algorithms. During training, the counterfactual method, λ-IPS, only observes actions taken by the baseline policy, hence its cumulative reward is always equal to the cumulative reward of the baseline and is omitted from the result tables to save space.

Table 4.4 lists the results for the text classification task. The cumulative reward achieved by SEA is at least as good as the baseline, and eventually better. We find that only ϵ-greedy on the RCV1 dataset under near random rewards performs significantly worse than the baseline. In all other settings we find that all the warm-started online methods similarly provide a user experience that is at least as good as the baseline, and eventually better. This means that for the text classification task, all methods, with the exception of ϵ-greedy, perform safely and never harm the user experience. Finally, we see that BSEA is able to improve the user experience faster and more quickly than SEA in the perfect rewards setting, and performs on par in the near random rewards setting. Similar to the results we have observed in Section 4.6.1, we hypothesize that BSEA does not have to overcome potentially large confidence bounds and as a result is able to more quickly and more frequently deploy its learned model. As a result, BSEA is capable of improving the user experience over SEA.

Next we turn to the case of ranking. See Table 4.5 for the cumulative reward results for document ranking. It is clear that SEA performs at least as good as the baseline policy and eventually outperforms it. For the online learning method, this is not the case. Specifically for the Istella-s dataset with near random clicks, we observe significant performance degradations in the early stages of learning which harms the user experience. However, on the other datasets with near random clicks we find that the online methods are capable of safely exploring the action space, even with high noise, and as a

Table 4.4: Cumulative reward, relative to the baseline policy, while training for text classification. LinUCB and Thompson Sampling cannot be run on the 20 Newsgroups and RCV1 datasets due to their high computational complexity. Statistically significant differences with SEA are indicated using $^{\blacktriangle}$ ($p < 0.01$) for gains and $^{\blacktriangledown}$ ($p < 0.01$) for losses.

		Round t	10^2	10^3	10^4	10^5	10^6
Perfect rewards	USPS	UCB	1.73	105.53$^{\blacktriangle}$	25.7×10^2$^{\blacktriangle}$	32.5×10^3$^{\blacktriangle}$	35.0×10^4$^{\blacktriangle}$
		Thompson	4.47	172.40$^{\blacktriangle}$	28.3×10^2$^{\blacktriangle}$	33.3×10^3$^{\blacktriangle}$	35.3×10^4$^{\blacktriangle}$
		ϵ-greedy	0.80	31.87	69.9×10^2$^{\blacktriangledown}$	10.6×10^3$^{\blacktriangledown}$	11.7×10^4$^{\blacktriangledown}$
		Boltzmann	0.93	69.27$^{\blacktriangle}$	19.5×10^2	28.0×10^3	31.3×10^4
		BSEA	0.00	0.00	20.9×10^2$^{\blacktriangle}$	30.5×10^3$^{\blacktriangle}$	34.2×10^4$^{\blacktriangle}$
		SEA	0.00	0.00	17.2×10^2	29.3×10^3	33.8×10^4
	20-News	ϵ-greedy	-2.00	60.07$^{\blacktriangle}$	25.7×10^2$^{\blacktriangle}$	42.9×10^3$^{\blacktriangle}$	50.0×10^4$^{\blacktriangledown}$
		Boltzmann	0.33	56.47$^{\blacktriangle}$	23.9×10^2$^{\blacktriangle}$	45.2×10^3$^{\blacktriangle}$	56.4×10^4$^{\blacktriangle}$
		BSEA	0.00	0.00	19.4×10^2$^{\blacktriangle}$	43.2×10^3$^{\blacktriangle}$	56.4×10^4$^{\blacktriangle}$
		SEA	0.00	0.00	10.8×10^2	37.3×10^3	54.0×10^4
	RCV1	ϵ-greedy	-3.53	-28.33	9.1×10^2$^{\blacktriangle}$	21.4×10^3$^{\blacktriangle}$	28.3×10^4
		Boltzmann	0.80	51.40	13.6×10^2$^{\blacktriangle}$	22.1×10^3$^{\blacktriangle}$	26.9×10^4
		BSEA	0.00	0.00	9.5×10^2	22.1×10^3$^{\blacktriangle}$	32.2×10^4$^{\blacktriangle}$
		SEA	0.00	0.00	5.7×10^2	18.9×10^3	28.2×10^4
Near random rewards (0.6 / 0.4)	USPS	UCB	-1.73	3.93	0.9×10^2	3.6×10^3	6.0×10^4
		Thompson	-1.73	-3.47	1.2×10^2	4.0×10^3	6.1×10^4
		ϵ-greedy	-0.60	9.73	0.5×10^2	-0.5×10^3$^{\blacktriangledown}$	0.7×10^4$^{\blacktriangledown}$
		Boltzmann	0.33	3.20	0.5×10^2	1.3×10^3$^{\blacktriangledown}$	2.3×10^4$^{\blacktriangledown}$
		BSEA	0.00	0.00	0.9×10^2	3.7×10^3	5.2×10^4
		SEA	0.00	0.00	0.5×10^2	3.6×10^3	5.5×10^4
	20-News	ϵ-greedy	-1.87	-11.33	-0.2×10^2	2.4×10^3	5.7×10^4
		Boltzmann	-2.40	-2.67	1.0×10^2	2.2×10^3	3.3×10^4
		BSEA	-0.33	1.60	1.1×10^2	3.0×10^3	5.8×10^4
		SEA	-0.33	1.60	0.8×10^2	2.5×10^3	4.8×10^4
	RCV1	ϵ-greedy	-0.47	-9.40	-2.5×10^2$^{\blacktriangledown}$	-1.4×10^3$^{\blacktriangledown}$	0.5×10^4$^{\blacktriangledown}$
		Boltzmann	-0.33	0.80	0.3×10^2	0.8×10^3	0.9×10^4$^{\blacktriangledown}$
		BSEA	0.00	0.47	0.0×10^2	0.9×10^3	2.5×10^4
		SEA	0.00	0.00	0.1×10^2	0.7×10^3	2.2×10^4

Table 4.5: Cumulative nDCG@10, relative to the baseline policy, while training for document ranking under varying levels of click noise. Statistical significance is denoted in the same way as in Table 4.4.

		Round t	10^3	10^4	10^5	10^6	10^7
Perfect clicks	MSLR10k	RankSVM	6.48	259.78▲	10.1×10^3▲	11.6×10^4▲	11.8×10^5▲
		DBGD	-0.62	-1.22	0.5×10^3	4.8×10^4▲	9.4×10^5▲
		BSEA	0.00	49.80	5.5×10^3▲	9.6×10^4▲	11.3×10^5▲
		SEA	0.00	6.52	2.7×10^3	7.2×10^4▲	10.0×10^5▲
	Webscope	RankSVM	31.75▲	514.23▲	6.5×10^3▲	7.1×10^4▲	7.3×10^5▲
		DBGD	-1.22	-4.49	0.2×10^3	1.9×10^4	4.8×10^5▲
		BSEA	2.98	325.39▲	5.5×10^3▲	6.5×10^4▲	6.9×10^5▲
		SEA	-0.05	151.24	4.7×10^3▲	6.0×10^4▲	6.7×10^5▲
	Istella-s	RankSVM	0.96	325.50▲	5.4×10^3▲	6.1×10^4▲	6.3×10^5▲
		DBGD	-13.16	-130.50	-0.9×10^3	0.6×10^4	2.2×10^5▲
		BSEA	0.00	206.87	4.7×10^3▲	6.0×10^4▲	6.3×10^5▲
		SEA	0.00	76.25	4.0×10^3▲	5.6×10^4▲	6.2×10^5▲
Position-biased clicks	MSLR10k	RankSVM	-5.17	184.43	5.8×10^3▲	6.9×10^4▲	7.3×10^5▲
		DBGD	-0.81	-3.12	0.4×10^3	3.1×10^4▲	4.5×10^5▲
		BSEA	0.00	3.85	2.8×10^3	7.0×10^4▲	9.3×10^5▲
		SEA	0.00	-0.01	0.9×10^3	3.2×10^4	6.3×10^5▲
	Webscope	RankSVM	26.88	349.04▲	5.0×10^3▲	6.1×10^4▲	6.4×10^5▲
		DBGD	-1.15	-6.12	0.1×10^3	1.5×10^4	3.5×10^5▲
		BSEA	8.86	324.38▲	4.9×10^3▲	5.8×10^4▲	6.1×10^5▲
		SEA	0.01	144.06	4.5×10^3▲	5.7×10^4▲	6.3×10^5▲
	Istella-s	RankSVM	-13.49	-14.85	3.3×10^3▲	5.0×10^4▲	5.6×10^5▲
		DBGD	-13.14	-133.17	-1.1×10^3	-0.2×10^4	1.5×10^5
		BSEA	0.00	128.66	4.2×10^3▲	5.5×10^4▲	6.0×10^5▲
		SEA	0.00	0.00	3.2×10^3▲	5.0×10^4▲	5.8×10^5▲
Near-random clicks	MSLR10k	RankSVM	-0.71	-24.48	0.9×10^3	6.6×10^4▲	8.5×10^5▲
		DBGD	-0.78	-7.48	-6.20	0.4×10^4	3.4×10^5▲
		BSEA	-0.00	-0.01	-0.06	-0.09	0.84
		SEA	-0.00	-0.01	-0.06	-0.09	0.84
	Webscope	RankSVM	-2.44	1.81	1.8×10^3	3.5×10^4▲	4.0×10^5▲
		DBGD	-1.24	-7.46	-50.84	0.1×10^4	1.7×10^5
		BSEA	0.03	0.03	-0.12	0.83	1.18
		SEA	0.03	0.03	-0.12	0.83	1.18
	Istella-s	RankSVM	-83.71▼	-678.03▼	-3.2×10^3▼	-1.8×10^4	-1.4×10^5
		DBGD	-13.07	-135.09	-1.3×10^3	-1.0×10^4	3837.78
		BSEA	0.00	0.00	0.00	0.00	0.00
		SEA	0.00	0.00	0.00	0.00	0.00

Table 4.6: Distribution of relevance grades on documents for the ranking datasets. Note that Istella-s is very sparse, nearly 90% of the documents are judged as non-relevant.

Relevance grade	0	1	2	3	4
Webscope	0.26	0.36	0.28	0.08	0.02
MSLR10k	0.52	0.32	0.13	0.02	0.01
Istella-s	0.89	0.02	0.04	0.03	0.02

result improve the overall user experience more than SEA. From Table 4.6 we see that Istella-s contains very sparse relevance feedback compared to the MSLR10k and Webscope datasets. In the near-random click scenario this means that there are significantly more clicks on non-relevant documents, making Istella-s with near-random clicks the most challenging learning scenario. We once again see that BSEA is a very strong baseline across all datasets, improving the user experience over SEA in the perfect and position-biased clicks settings while performing on par with SEA in the near-random click setting.

This answers RQ3.2. SEA provides a better user experience than online methods, particularly in the document ranking setting with high levels of click noise where the user experience may be negatively affected by online algorithms. In the later stages of learning SEA provides a user experience that is significantly better than the baseline and comparable to online approaches. We find that BSEA is able to improve the user experience even further by being able to deploy its learned model earlier and more frequently.

4.6.3 The benefit of exploring

Finally, we turn to RQ3.3:

> Does SEA, which explores the action space, learn a more effective policy compared to purely counterfactual learning methods, which do not perform exploration?

To answer RQ3.3, we compare SEA to the state-of-the-art counterfactual learning algorithm λ-IPS on unseen test data. The results for the classification task are displayed in Table 4.7. Because SEA is capable of exploring, it finds highly favorable regions of the action space. λ-IPS is, by design, incapable of exploration and cannot deviate far from the baseline policy in terms of performance. As a result, in the case of text classification, the final model learned by SEA outperforms the final model learned by λ-IPS on the 20 Newsgroups datasets. On the USPS and RCV1 dataset, there is no noticeable performance difference between λ-IPS and SEA. Note that the baseline policy for USPS and RCV1 has a performance of around 0.6, whereas the baseline policy for the 20 Newsgroups dataset only has a performance of around 0.4. We postulate that this difference in baseline performance may cause λ-IPS to learn better and therefore perform equally to SEA on the USPS and RCV1 datasets. Lastly, we find that BSEA performs on par with SEA in all settings, while producing a significantly better model on the RCV1 dataset with perfect rewards. Similar to our observations in

Table 4.7: Average reward on held-out test data for the learned model for the document classification task after 1,000,000 rounds. Statistical significance is denoted the same way as in Table 4.4.

	Type of rewards:	*Perfect*	*Near random (0.6 / 0.4)*
USPS	ϵ-greedy	0.90$^{\blacktriangledown}$	0.82$^{\blacktriangledown}$
	Boltzmann	0.90	0.72$^{\blacktriangledown}$
	UCB	0.91$^{\blacktriangledown}$	0.91
	Thompson	0.91$^{\blacktriangledown}$	0.91
	IPS	0.92	0.89
	BSEA	0.92	0.86
	SEA	0.92	0.90
20-News	ϵ-greedy	0.83	0.69
	Boltzmann	0.88$^{\blacktriangle}$	0.53$^{\blacktriangledown}$
	IPS	0.83$^{\blacktriangledown}$	0.53$^{\blacktriangledown}$
	BSEA	0.86	0.74
	SEA	0.85	0.70
RCV1	ϵ-greedy	0.85	0.68
	Boltzmann	0.83	0.63$^{\blacktriangledown}$
	IPS	0.81	0.68
	BSEA	0.87$^{\blacktriangle}$	0.74
	SEA	0.84	0.73

Sections 4.6.1 and 4.6.2, we find that BSEA likely performs so well because it does not have to overcome confidence bounds and can deploy its learned model faster and more frequently than SEA, allowing it to learn a more effective model.

Finally, we consider the document ranking setting; see Table 4.8. SEA is able to learn a more effective ranker than λ-IPS on all datasets with perfect clicks and on Istella-s with position-biased clicks. We hypothesize that this is because SEA, being capable of exploration, eventually shows documents to the user that the baseline policy would rarely, if ever, show. As a result, SEA is able to obtain clicks on these documents, allowing it to learn more effectively. This is in line with our expectations because previous work has shown that even a tiny amount of exploration can result in substantial improvements in LTR [57]. Finally, we find that BSEA does not produce a better ranker than SEA for the document ranking task. Across all settings, the models produced by BSEA and SEA are comparable.

This answers RQ3.3. Exploration does indeed help the performance of the policy learned by SEA and it outperforms λ-IPS both for scenarios with a small action space (e.g., text classification) and for scenarios with a large action space (e.g., document ranking).

Table 4.8: Average reward on held-out test data for the learned model for the document ranking task after 10,000,000 rounds. Statistical significance is denoted the same way as in Table 4.4.

	Type of clicks:	*Perfect*	*Position-biased*	*Near random*
MSLR10k	RankSVM (Online)	0.43	0.39	0.40▲
	DBGD	0.43	0.38	0.39▲
	IPS	0.35▼	0.35▼	0.32
	BSEA	0.43	0.41	0.32
	SEA	0.43	0.40	0.32
Webscope	RankSVM (Online)	0.75	0.74	0.72▲
	DBGD	0.74▼	0.72▼	0.71
	IPS	0.73▼	0.74▼	0.69
	BSEA	0.75	0.74	0.69
	SEA	0.75	0.74	0.69
Istella-s	RankSVM (Online)	0.67	0.67▼	0.60
	DBGD	0.66▼	0.65▼	0.64
	IPS	0.67▼	0.66▼	0.64
	BSEA	0.68	0.67	0.64
	SEA	0.68	0.67	0.64

4.7 Conclusion

We have proposed SEA, a *safe online* learning algorithm for contextual bandit problems. SEA learns a new policy from the behavior of an existing baseline policy and then starts to execute actions from the new policy once its estimated performance is at least as good as that of the baseline. This brings us the best of both worlds, achieving the performance of *online* learning with the safety of *counterfactual* learning.

We perform extensive experimentation on two IR tasks, *text classification* and *document ranking*. In both tasks SEA is safe. It never performs worse than a baseline policy, whereas online methods are unsafe and suffer from suboptimal performance in the early stages of learning. We observe that the user experience with SEA is improved in the early stages of training, but may be suboptimal in later stages when compared to methods that converge much faster, such as LinUCB (although such methods are not generalizable to all datasets). The final performance of a model learned with SEA is comparable to other online algorithms and beats that of counterfactual methods, which are incapable of exploration. Finally, we find that BSEA, a boundless version of SEA, is empirically just as safe as SEA while being able to explore faster and, as a result, outperform SEA in many experimental conditions. These results confirm that SEA does indeed trade off advantages and disadvantages of *counterfactual* and *online* learning, in some scenarios outperforming online methods in the early stages of learning and having higher final performance than counterfactual methods. However, compared to purely *online* approaches, SEA may not be as effective in learning a good policy due to

the fact that it is more conservative when exploring. The conservative nature of SEA implies that safety is not free, there is a possible performance cost involved for cases where SEA is unable to effectively explore.

An interesting direction for future work is an extension of SEA to non-linear models. SEA builds on gradient descent and it is trivial to extend the method to use gradient-boosted decision trees or neural networks. This line of work is especially applicable for the document ranking task where it is known that non-linear models can outperform linear models by a wide margin [178]. Furthermore, another possibility for future work is to perform a study on safety for a broad range of online and counterfactual methods, similar to the comparison performed in Chapter 2. Specifically, it would be interesting to compare against recent work on safe online learning to re-rank [99].

All chapters so far have assumed stationary user preferences that do not change over time and we have only looked at clicks as a user interaction signal. In the following two chapters we look more closely at these topics. In Chapter 5 we introduce counterfactual estimators that work well in *non-stationary* environments. Then, in Chapter 6 we look at *activity data* in the context of cloud file storage platforms.

5

Off-Policy Evaluation for Non-Stationary Recommendation Environments

5.1 Introduction

Modern Information Retrieval (IR) systems leverage user interactions such as clicks to optimize which items such as articles, music or movies to show to users [59, 76, 104]. A challenge in utilizing interaction feedback is that it is a "partial label" problem: We only observe feedback for items that were shown to a user, but not for other items that could have been shown. The contextual bandit framework [100] provides a natural way to solve problems with this interactive nature. In the contextual bandit setup, an interactive system (e.g., a recommender system), often called a *policy*, observes a context (e.g., a user visiting a website), performs an action (e.g., by showing a recommendation to the user) and finally observes a reward for the performed action (e.g., a click or no click) [100].

To evaluate a policy, it is best to deploy it online, e.g., in the form of an A/B test. However, this is expensive in terms of engineering and logistic overhead [70, 182] and may harm the user experience [143]. *Off-policy* evaluation is an alternative strategy that avoids the problems of deploying and measuring a policy's performance online [101]. In off-policy evaluation, we use historical interaction data, often referred to as *bandit feedback*, collected by an existing logging policy to estimate the performance of a new policy. In existing work, off-policy evaluation has been well studied in the context of a stationary world, one where interactions happen independent of time [15, 40, 101, 129, 158, 163, 175].

However, IR environments are usually *non-stationary* with user preferences changing over time [93, 111, 114, 133, 179]. Existing off-policy evaluation techniques fail to work in such environments. In this chapter, we address the problem of off-policy evaluation in non-stationary environments. We propose several off-policy estimators that operate well when the environment is non-stationary. Our estimators are based on applying two types of moving averages to the collected bandit feedback: (1) a sliding window average, and, (2) an exponential decay average. These proposed estimators rely more on recent bandit feedback and, thus, accurately capture changes in

This chapter was published as [71].

non-stationary environments.

We provide a rigorous analysis of the proposed estimators' bias in the non-stationary setting and show that the bias does not grow over time. In contrast, we show that the standard Inverse Propensity Scoring (IPS) estimator suffers from a large bias that grows over time when applied to non-stationary environments. Finally, we use the results from our analysis to create adaptive variants of the sliding window and exponential decay estimators that change their parameters in real-time to improve estimation performance.

We perform extensive empirical evaluation of the proposed off-policy estimators on two recommendation datasets to showcase how they behave under varying levels of non-stationarity. Our main finding is that the proposed estimators significantly outperform the regular IPS estimator and provide a much more accurate estimation of a policy's true performance, while the regular IPS fails to capture the changes in non-stationary environments. Moreover, we demonstrate that these results hold for both smooth and abrupt changes in the environment. Our findings open up the way for off-policy evaluation to be applied to real-world settings where non-stationarity is prevalent.

The remainder of this chapter is structured as follows: In Section 5.2 we provide background information about off-policy evaluation and non-stationarity. Next, Section 5.3 describes our estimators for solving the non-stationary off-policy evaluation problem. The experimental setup and results are described in Section 5.4 and Section 5.5, respectively. Finally, we conclude in Section 5.6.

5.2 Background

5.2.1 Off-policy evaluation

Off-policy evaluation is an important technique for assessing the behavior of a decision making policy, e.g., a new recommendation strategy, ad-placement technique or some other new feature, without deploying the policy in a classical A/B test [129]. In settings where the deployment of a new policy is costly, either in terms of logistic and engineering overhead or in terms of potential harm to the user experience, off-policy evaluation is a safe and efficient alternative to A/B testing [162]. The main idea in off-policy evaluation is to collect data by having an already deployed policy taking actions and logging the corresponding user interactions. The typical approach in off-policy evaluation is to then re-weigh the logged data according to what the new policy *would have done* to obtain an unbiased estimate of the expected return of this new policy. Although a randomized logging policy is typically required for unbiased off-policy evaluation, the amount of randomness can usually be controlled through some parameter, trading off exploration and exploitation [40].

Existing work in off-policy evaluation has focused on creating unbiased estimators [62, 101, 158] and reducing their variance [15, 40, 175]. However, these off-policy estimators usually do not take into account the *temporal* component and assume that the world and rewards are stationary. In contrast to existing work, we postulate that the world is *non-stationary* and create off-policy estimators that take this into account.

We should note that Dudík et al. [41] have studied policy evaluation in a different non-stationary setting, namely one where a policy's behavior depends on a history of contexts and actions. However, unlike our work, Dudík et al. still assumes a stationary world and rewards.

In the reinforcement learning domain, the application of time series prediction methods to predict future off-policy performance in non-stationary environments has been studied by Thomas et al. [165]. Our work is different in important ways: (1) Our work focuses on the contextual bandit scenario, whereas theirs is in the reinforcement learning domain. (2) We are the first to develop a theory for non-stationary off-policy evaluation. (3) Their work is designed for small-scale problems, with up to a few thousand iterations as the complexity of their method is quadratic in the number of iterations, whereas our method scales linearly with the number of iterations, enabling experimentation that is two orders of magnitude larger. (4) We target the recommendation setting, whereas Thomas et al. [165] focus on proprietary datasets from digital ad marketing, limiting reproducibility. The only publicly available dataset used in their work is a synthetic scenario called mountain car [154]. Our results are produced on more realistic publicly available recommendation datasets from LastFM [96] and Delicious [22, 35].

Finally, Garivier and Moulines [48] studied the use of sliding-window and exponential decay techniques for optimizing contextual bandits in abruptly changing environments. Our work differs in two ways: (1) we study off-policy evaluation and not contextual bandit learning, and (2) we focus on the smooth non-stationary setting instead of the abrupt non-stationary setting, as we will explain in the next section.

5.2.2 Non-stationary environments

Non-stationarity environments have been studied in the context of learning multi-armed bandits [17, 48, 103, 177] and contextual bandits [179]. Two settings naturally arise when dealing with a non-stationary world:

1. Abrupt non-stationarity [48, 179], sometimes called piecewise-stationary [103].

2. Smooth non-stationarity [177].

The first setting, abrupt non-stationarity, assumes a stationary world where changes happen abruptly at certain points in time. This is a natural setting in, for example, news recommendation, where a sudden event causes a shift in users' interests [97].

The second setting, smooth non-stationarity, assumes that the world changes constantly but that it changes only a little bit at a time. This is the natural condition of human attitudes (including likes and dislikes). Social psychologists have found that preferences are neither enduring nor stable [146, 166]. In cognitive psychology, numerous experiments have provided evidence of gradual taste changes, for instance in response to changing constraints and abilities [14] or in relation to perceived risk levels [85].

Specifically, in settings such as e-commerce [108], music recommendation [111, 132] and news recommendation [37], the behavior of users is often non-stationary in a smooth manner. Pereira et al. [128] have studied non-stationarity in user preferences

on social media and have found strong correlations between the temporal dynamics of users' preferences and changes in their social network graph. Taking smooth non-stationarity into account may benefit overall search and recommendation performance, e.g., in music recommendation; Quadrana et al. [132] have found that encoding the evolution of users' listening preferences via recurrent neural networks, can lead to substantial improvements in recommendation quality.

In our work, we specifically design off-policy estimators that deal with the *non-abrupt* case, that is, estimators that work well when the environment exhibits *smooth non-stationarity*.

5.3 Non-stationary Off-policy Evaluation

In this section we first formulate the problem of off-policy evaluation in non-stationary environments. In this setting, we prove that the upper bound on the bias of the regular IPS estimator grows with time. Then we propose two alternative estimators, a sliding window approach and an exponential decay approach, and show that their bias can be bounded. Finally, we use our theoretical findings to propose a method that can adaptively set the window-size or the decay rate of our proposed estimators, based on the principle of minimizing the Mean Squared Error (MSE).

5.3.1 Problem definition

We consider the following two policies: (i) π_0 is a stochastic logging policy that collects data, and (ii) π_w is a new policy that we want to evaluate. We observe an infinite stream of log data, generated by the logging policy π_0. At each time $t = 1, \ldots, \infty$, the following occurs:

1. The environment generates a context vector x_t and rewards r_t for all possible actions at time t:
$$(x_t, r_t) \overset{\text{i.i.d.}}{\sim} \mathcal{D}_t. \tag{5.1}$$
The context could, for example, be a user who interacts with a recommender system, while actions could be possible recommendations for that user. The true interest of the user, e.g., what recommendation they are actually interested in, is captured by rewards. We build on previous work [101], which assumes that contexts and rewards are sampled i.i.d. from an unknown distribution $\mathcal{D}_t$. However, unlike previous work, we generalize to a non-stationary world that may change over time. More formally, we allow the distribution $\mathcal{D}_t$ to change with t:
$$\mathcal{D}_1 \neq \mathcal{D}_2 \neq \ldots \neq \mathcal{D}_t. \tag{5.2}$$

2. After observing the context vector x_t, the logging policy π_0 samples an action (e.g., given a user, π_0 chooses a recommendation for that user):
$$a_t \sim \pi_0(\cdot \mid x_t) \tag{5.3}$$
and records the corresponding propensity score:
$$p_t \leftarrow \pi_0(a_t \mid x_t). \tag{5.4}$$

3. The reward $r_t(a_t)$ for the chosen action is revealed, but not the rewards for other possible actions that could have been chosen. Without loss of generality we assume $r_t(a_t) \in [0, 1]$. In practice, the reward would be a click or no-click on the recommendation that was shown.

Our goal is to estimate the *value* of the new policy π_w at time t, denoted as $\mathrm{V}_t^*(\pi_w)$, based on the data collected by the logging policy π_0. We write this value as the expected reward of the policy π_w:

$$\mathrm{V}_t^*(\pi_w) = \mathbb{E}_{(x_t, r_t)\sim\mathcal{D}_t, a_t\sim\pi_w(\cdot|x_t)}\left[r_t(a_t)\right] = \mathbb{E}_{\pi_w}\left[r_t(a_t)\right]. \tag{5.5}$$

Finding an estimator for the above quantity would be near impossible if no further assumptions are made about the reward function r_t. For example, if a user's preferences completely changed every time they enter a recommendation website, it would be impossible to perform any type of estimation or evaluation. To make this problem approachable, we assume that the change of a policy's value between any two consecutive points in time is bounded. More formally:

Assumption 5.3.1. The value of a policy is a Lipschitz function of time:

$$|\mathrm{V}_{t_1}^*(\pi_w) - \mathrm{V}_{t_2}^*(\pi_w)| \leq |t_1 - t_2|k, \tag{5.6}$$

where k is the Lipschitz-constant.

This assumption ensures that the expected reward of a policy cannot abruptly jump between time t_1 and time t_2. This is supported by practical observations that for real-world recommendation systems user behavior changes slowly over time [111, 114].

5.3.2 Regular IPS

A widely used policy evaluation technique is Inverse Propensity Scoring (IPS) [62], defined as:

$$\mathrm{V}_t^{\mathrm{IPS}}(\pi_w) = \frac{1}{t}\sum_{i=1}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}. \tag{5.7}$$

Under the assumption of a stationary world, this is an unbiased estimate of $\mathrm{V}_t^*(\pi_w)$ [62]:

Lemma 5.3.2. $\mathrm{V}^{\mathrm{IPS}}(\pi_w)$ is an unbiased estimate of $\mathrm{V}_t^*(\pi_w)$, under the assumption of a stationary world ($\mathcal{D}_1 = \mathcal{D}_2 = \ldots = \mathcal{D}_t$).[1]

Proof. First, we show that for any point in time $i \in \{1, \ldots, t\}$, the IPS estimate of a single observation is unbiased. To do this, we take the expectation of the IPS estimate under the logging policy, and show that it is equal to the reward of the policy under

[1] We assume $\pi_0(a \mid x) \neq 0$ when $\pi_w(a \mid x) \neq 0$ throughout this chapter, a necessary assumption to guarantee unbiasedness of IPS estimators.

evaluation:

$$\begin{aligned}\mathbb{E}_{\pi_0}\left[r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] &= \sum_{a'} r_i(a'_i)\frac{\pi_w(a'_i \mid x_i)}{\pi_0(a'_i \mid x_i)}\pi_0(a'_i \mid x_i) \\ &= \sum_{a'} r_i(a'_i)\pi_w(a'_i \mid x_i) \\ &= \mathbb{E}_{\pi_w}\left[r_i(a_i)\right].\end{aligned}$$

Using this fact, it is easy to show that $\mathrm{V}_t^{\mathrm{IPS}}(\pi_w)$ is indeed an unbiased estimate of $\mathrm{V}_t^*(\pi_w)$:

$$\begin{aligned}\mathbb{E}_{\pi_0}\left[\mathrm{V}_t^{\mathrm{IPS}}(\pi_w)\right] &= \mathbb{E}_{\pi_0}\left[\frac{1}{t}\sum_{i=1}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\pi_w}\left[r_i(a_i)\right] \\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\pi_w}\left[r_t(a_t)\right] \\ &= \mathbb{E}_{\pi_w}\left[r_t(a_t)\right] \\ &= \mathrm{V}_t^*(\pi_w). \quad \square\end{aligned}$$

When generalizing to a non-stationary world (under Assumption 5.3.1), it can be shown that the standard IPS estimate is biased.

Lemma 5.3.3. Under Assumption 5.3.1, $\mathrm{V}_t^{\mathrm{IPS}}(\pi_w)$ is a biased estimate of $\mathrm{V}_t^*(\pi_w)$ and the upper bound on the bias grows with t.

Proof. We have:

$$\begin{aligned}\mathbb{E}_{\pi_0}\left[\mathrm{V}_t^{\mathrm{IPS}}(\pi_w)\right] &= \mathbb{E}_{\pi_0}\left[\frac{1}{t}\sum_{i=1}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\pi_w}\left[r_i(a_i)\right] \\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\pi_w}\left[r_i(a_i) - r_t(a_t) + r_t(a_t)\right] \\ &= \frac{1}{t}\sum_{i=1}^{t}\left(\mathbb{E}_{\pi_w}\left[r_t(a_t)\right] + \mathbb{E}_{\pi_w}\left[r_i(a_i) - r_t(a_t)\right]\right) \\ &= \mathbb{E}_{\pi_w}\left[r_t(a_t)\right] + \frac{1}{t}\sum_{i=1}^{t}\left(\mathbb{E}_{\pi_w}\left[r_i(a_i)\right] - \mathbb{E}_{\pi_w}\left[r_t(a_t)\right]\right) \\ &= \mathrm{V}_t^*(\pi_w) + \frac{1}{t}\sum_{i=1}^{t}\left(\mathrm{V}_i^*(\pi_w) - \mathrm{V}_t^*(\pi_w)\right)\end{aligned}$$

$$\leq \mathrm{V}_t^*(\pi_w) + \frac{1}{t}\sum_{i=1}^{t} |t-i|k$$
$$= \mathrm{V}_t^*(\pi_w) + \underbrace{\frac{k(t-1)}{2}}_{\text{bias}} \quad \square$$

The upper bound on the bias term, i.e., $\frac{k(t-1)}{2}$, grows with t, which is unfortunate because it means that the more data we observe, the larger our bias potentially becomes. We propose two estimators that deal with this problem by avoiding a bias term that grows with t: the sliding window IPS and the exponential decay IPS, which we describe next.

5.3.3 Sliding window IPS

The first IPS estimator we propose is the *sliding window IPS estimator*, $\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)$. This estimator only takes into account the τ most recent observations and ignores older ones:

$$\mathrm{V}_t^{\tau\text{IPS}}(\pi_w) = \frac{1}{\tau}\sum_{i=t-\tau}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}. \tag{5.8}$$

This estimator has a bias that does not grow with t, but is instead controlled by the window size τ:

Theorem 5.3.4. Under Assumption 5.3.1, $\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)$ is a biased estimate of $\mathrm{V}_t^*(\pi_w)$ and its bias is at most $\frac{k(\tau-1)}{2}$.

Proof. This proof largely follows the proof of Lemma 5.3.3, so we will be concise:

$$\mathbb{E}_{\pi_0}\left[\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)\right] = \mathbb{E}_{\pi_0}\left[\frac{1}{\tau}\sum_{i=t-\tau}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right]$$
$$\leq \mathrm{V}_t^*(\pi_w) + \frac{1}{\tau}\sum_{i=t-\tau}^{t} |t-i|k$$
$$= \mathrm{V}_t^*(\pi_w) + \frac{1}{\tau}\sum_{i=1}^{\tau} |\tau-i|k$$
$$\leq \mathrm{V}_t^*(\pi_w) + \underbrace{\frac{k(\tau-1)}{2}}_{\text{bias}}. \quad \square$$

The advantage of the sliding window estimator $\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)$ is that its bias term can be controlled by the window size τ. One may consider setting the window size τ to 1, which would effectively produce an unbiased estimate:

$$\frac{k(\tau-1)}{2} = \frac{k(1-1)}{2} = 0. \tag{5.9}$$

This is a particularly powerful statement because we would obtain an unbiased estimator even in the face of non-stationarity. Unfortunately, a drawback would be that having such a small window size will cause a large variance.

To formally derive the variance of the $\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)$ estimator, we assume that the following variance does not change over time:

$$\mathbb{V}\left[r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] = \mathbb{V}\left[r_{i+1}(a_{i+1})\frac{\pi_w(a_{i+1} \mid x_{i+1})}{p_{i+1}}\right]. \tag{5.10}$$

We make this assumption to simplify writing down the variance of our estimator. To further motivate this assumption, we note that the variance of IPS estimators scales quadratically with the inverse propensity scores [40]. As a result, the variance term of the IPS estimator is dominated by the usually large inverse propensity weights and not by the variance in the rewards. Since we do not change our logging policy over time, the distribution of propensity scores will also not change, and hence we expect the variance to remain constant over time. We can write down the variance of $\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)$ as follows:

$$\begin{aligned}\mathbb{V}\left[\mathrm{V}_t^{\tau\text{IPS}}(\pi_w)\right] &= \mathbb{V}\left[\frac{1}{\tau}\sum_{i=t-\tau}^{t} r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\ &= \frac{1}{\tau^2}\sum_{i=t-\tau}^{t}\mathbb{V}\left[r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] = \frac{1}{\tau}\mathbb{V}\left[r_t(a_t)\frac{\pi_w(a_t \mid x_t)}{p_t}\right].\end{aligned}$$

As we can see, the variance scales by $\frac{1}{\tau}$, which means larger values of τ reduce variance and conversely smaller values of τ increase variance.

Hence, setting the window size is a trade-off between how much bias and variance we are willing to tolerate. We will see a similar bias-variance trade-off in the next estimator, the exponential decay IPS.

5.3.4 Exponential decay IPS

The *exponential decay IPS estimator*, $\mathrm{V}_t^{\alpha\text{IPS}}(\pi_w)$, uses an exponential moving average to weigh recent observations more heavily than old observations:

$$\mathrm{V}_t^{\alpha\text{IPS}}(\pi_w) = \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}, \tag{5.11}$$

where $\alpha \in (0,1)$ is a hyper parameter controlling the rate of decay. A large value of α indicates a slow decay, meaning that old observations weigh more heavily. Conversely, a small value of α indicates a rapid decay, which means recent observations weigh more heavily.

The bias of this estimator does not grow with t and is controlled by the decay rate α:

Theorem 5.3.5. Under Assumption 5.3.1, $\mathrm{V}_t^{\alpha\text{IPS}}(\pi_w)$ is a biased estimate of $\mathrm{V}_t^*(\pi_w)$ and its bias is at most $\frac{k\alpha}{(1-\alpha)(1-\alpha^t)}$.

Proof. For notational simplicity, we define $\mathrm{V}_i^* = \mathrm{V}_i^*(\pi_w)$. Then:

$$\begin{aligned}
\mathbb{E}_{\pi_0}\left[\mathrm{V}_t^{\alpha\text{IPS}}(\pi_w)\right] &= \mathbb{E}_{\pi_0}\left[\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\
&= \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}\mathbb{E}_{\pi_w}\left[r_i(a_i)\right] \\
&= \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}(\mathrm{V}_i^* - \mathrm{V}_t^* + \mathrm{V}_t^*) \\
&= \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}\mathrm{V}_t^* + \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}(\mathrm{V}_i^* - \mathrm{V}_t^*) \\
&= \mathrm{V}_t^* + \underbrace{\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}(\mathrm{V}_i^* - \mathrm{V}_t^*)}_{\text{bias}}.
\end{aligned}$$

We can further simplify the bias term as follows:

$$\begin{aligned}
\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}(\mathrm{V}_i^* - \mathrm{V}_t^*) &\leq \frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}|t-i|k \\
&= k\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}|t-i|.
\end{aligned}$$

Note that $\sum_{i=1}^{t}\alpha^{t-i}|t-i|$ is a convergent series for $|\alpha| < 1$:

$$\sum_{i=1}^{t}\alpha^{t-i}|t-i| \leq \frac{\alpha}{(1-\alpha)^2}.$$

Plugging this expression into the above equation completes the proof:

$$k\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}|t-i| \leq k\frac{1-\alpha}{1-\alpha^t}\frac{\alpha}{(1-\alpha)^2} = \frac{k\alpha}{(1-\alpha)(1-\alpha^t)}. \qquad \square$$

The bias term $\frac{k\alpha}{(1-\alpha)(1-\alpha^t)}$ exhibits behavior that we expect: If k is large, and thus the environment is highly non-stationary, the estimate will be more biased. Conversely, when $k = 0$, we recover the stationary case and have an unbiased estimator. Furthermore, when α approaches 1, the bias term grows because we weigh old observations more heavily. Finally, we note that $\lim_{t\to\infty}(1-\alpha^t) = 1$ and thus t vanishes from the bias term as t approaches infinity.

Let us now consider the variance of the exponential decay IPS estimator. Similarly to the sliding window IPS estimator, we assume that the variance does not change over

time. This gives us:

$$\begin{aligned}\mathbb{V}\left[\mathrm{V}_t^{\alpha\mathrm{IPS}}(\pi_w)\right] &= \mathbb{V}\left[\frac{1-\alpha}{1-\alpha^t}\sum_{i=1}^{t}\alpha^{t-i}r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\ &= \left(\frac{1-\alpha}{1-\alpha^t}\right)^2\sum_{i=1}^{t}\alpha^{2(t-i)}\mathbb{V}\left[r_i(a_i)\frac{\pi_w(a_i \mid x_i)}{p_i}\right] \\ &= \left(\frac{1-\alpha}{1-\alpha^t}\right)^2\left(\frac{1-\alpha^{2t}}{1-\alpha^2}\right)\mathbb{V}\left[r_t(a_t)\frac{\pi_w(a_t \mid x_t)}{p_t}\right].\end{aligned}$$

As expected, the variance scaling factor $\left(\frac{1-\alpha}{1-\alpha^t}\right)^2\left(\frac{1-\alpha^{2t}}{1-\alpha^2}\right)$ decreases as α goes to 1. Conversely, the variance increases as α goes to 0. Similarly to the bias term, we see that t vanishes from the variance as t approaches infinity:

$$\lim_{t\to\infty}\left(\frac{1-\alpha}{1-\alpha^t}\right)^2\left(\frac{1-\alpha^{2t}}{1-\alpha^2}\right) = \frac{1-\alpha}{1+\alpha}.$$

5.3.5 How to choose τ and α

Compared to regular IPS estimators, $\mathrm{V}_t^{\tau\mathrm{IPS}}$ and $\mathrm{V}_t^{\alpha\mathrm{IPS}}$ have additional parameters τ and α, respectively, that need to be set.

Let us first consider the scenario where an unbiased estimator is the goal. We can set $\tau = 1$ or $\alpha = 0$ to obtain an unbiased estimator. This is equivalent to computing an IPS estimate on only the current observation. It is obvious that such a strategy will suffer from high variance and is not very useful in practice.

Conversely, if we were to consider the scenario where an estimator with minimal variance is the goal, we could set $\tau = \infty$ or α arbitrarily close to 1, resulting in an estimator that would heavily weigh as many old observations as possible. This is also a poor strategy as it would result in potentially unbounded bias.

Setting τ or α comes down to finding a balance between bias and variance. A principled way to trade off these quantities is by minimizing the mean squared error of the estimator [169]:

$$MSE = \text{bias}^2 + \text{variance}.$$

If the Lipschitz constant k is known, we can compute a value of τ or α that minimizes the mean squared error at every time t as follows:

$$\begin{aligned}\tau_t^* &= \operatorname*{argmin}_{\tau\in\mathbb{N}}\lambda\left(\frac{k(\tau-1)}{2}\right)^2 + \mathbb{V}\left[\mathrm{V}_t^{\tau\mathrm{IPS}}(\pi_w)\right], \\ \alpha_t^* &= \operatorname*{argmin}_{\alpha\in[0,1)}\lambda\left(\frac{k\alpha}{(1-\alpha)(1-\alpha^t)}\right)^2 + \mathbb{V}\left[\mathrm{V}_t^{\alpha\mathrm{IPS}}(\pi_w)\right],\end{aligned}$$

where λ is a hyperparameter that trades off bias for variance. In practice we would tune λ to achieve a good trade-off.

Finding the optimal values τ_t^* and α_t^* requires knowledge about the Lipschitz constant k which is usually not known in practice. In the next section, we describe a heuristic that estimates k.

5.3.6 Estimating the Lipschitz constant k

The Lipschitz constant k tells us how fast the true value of a policy is moving (see Eq. (5.6)). Since the true value $\mathrm{V}_t^*(\pi_w)$ cannot be observed without deploying the policy π_w, we rely on the IPS estimated rewards. To estimate k, we track the difference between two moving averages: one at time t, denoted as V_t and one at time $t-s$, denoted as V_{t-s}, where $s > 0$ is a parameter representing a window size for estimating k.

Now, we can estimate k at every time t as follows:

$$\hat{k}_t = \frac{1}{s} \left(\mathrm{V}_t - \mathrm{V}_{t-s} \right), \tag{5.12}$$

where V_t is a moving average estimator at time t. For example, V_t could be the exponential decay estimator $\mathrm{V}_t^{\alpha\text{IPS}}(\pi_w)$.

Tracking the difference between two averages at different points in time has previously been used as a change-point detection mechanism for contextual bandits. For example, the windowed mean-shift algorithm uses a very similar method to detect when an abrupt change occurs [183]. Our heuristic is different in the fact that it does not detect an abrupt change, but instead is measuring how fast the true value of the policy is moving up and down.

5.4 Experimental Setup

In this section we describe our experimental setup. The goal of our experiments is to answer the following research question:

> **RQ4** How can counterfactual approaches be adapted to deal with non-stationary environments?

To answer the above research question we propose the following sub-questions:

RQ4.1 How well do the proposed estimators perform in a non-stationary environment?

RQ4.2 How well do the estimators function when Assumption 5.3.1 is violated? E.g., when the environment changes abruptly?

RQ4.3 Can the proposed estimators be applied to stationary environments?

RQ4.4 How do the estimators behave under different parameters?

To answer these questions we consider a simulated non-stationary contextual bandit setup as described in [179]. Note that although our setup is the same as in [179], we are solving a different problem: particularly, we perform off-policy evaluation whereas Wu et al. [179] perform online learning.

5.4.1 Experimental methodology

We evaluate our proposed off-policy estimators in the context of recommendation, where a policy recommends an item to a user. We use the non-stationary contextual bandit setup of Wu et al. [179], which, in turn, builds on the experimental setup of Cesa-Bianchi et al. [25]. In this experimental setup, we use two datasets made available as part of the HetRec2011 workshop [22, 35, 96] and convert them into a contextual bandit problem: LastFM and Delicious. For the LastFM dataset [96], we consider a random artist that the user has listened to as positive feedback and an artist that the user has not listened to as negative feedback. For the Delicious dataset [35], we consider a website that the user has bookmarked as positive feedback and websites that the user has not bookmarked as negative feedback. For each user we consider a random positive item and 24 random negative items as the set of candidate actions. Correspondingly, a reward of 1 is given if a policy chooses the positive item and 0 otherwise. Each item is described by a TF-IDF feature vector comprised of the item's tags, e.g., "metal", "electronic", "rock", etc. in the case of music recommendation (LastFM), and "social", "games", "tech", etc. in the case of bookmark recommendation (Delicious). This feature vector is reduced to 25 dimensions via PCA, as described in [25].

To introduce non-stationarity we follow the setup of Wu et al. [179]: We cluster users into 10 user groups (or super-users) via spectral clustering based on the social network graph structure. Users who are close in the social network graph are hypothesized to have similar preferences.

Then, a single hybrid user is created from the 10 super-users by stacking the preferences of the 10 user groups chronologically. This hybrid user is non-stationary because its preferences change when it moves from one super-user to the next. In [179], the hybrid user switches abruptly between the 10 super-users at certain points in time. We experiment with the existing abrupt case of [179] and introduce a setup where a mixture of the 10 super-users slowly changes over time as observed in real-world recommender systems [111, 114]. Below we describe both setups in detail.

The hybrid user can be represented by a mixture with 10 components which add up to 1. For example:

$$[0, \quad 1, \quad 0, \quad 0, \quad \ldots, \quad 0].$$

To simulate a *smooth* non-stationary setup, we introduce a transition period from time t_1 to t_2. In this transition period we change two components, linearly reducing one component while linearly increasing the other. For example, changing from the second to the third super-user would happen as follows:

$$\begin{array}{lllllll}
t_1: & [0, & 1, & 0, & 0, & \ldots, & 0] \\
t_1+1: & [0, & 0.9, & 0.1, & 0, & \ldots, & 0] \\
t_1+2: & [0, & 0.8, & 0.2, & 0, & \ldots, & 0] \\
\vdots & & & & & & \\
t_2: & [0, & 0, & 1, & 0, & \ldots, & 0].
\end{array}$$

This setup is in line with Assumption 5.3.1, which states that the environment changes only a little bit at a time and not abruptly.

In the *abrupt* setup, one component is set to 1 and the other components are set to 0. An abrupt change happens by changing which component is set to 1. This is equivalent to the setup in [179] where the hybrid user switches abruptly between the 10 super-users.

Using the non-stationary setups described in this section, we can now deploy a logging policy that collects bandit feedback and evaluate a set of candidate policies using the proposed off-policy estimators. The choice of a logging policy and candidate policies are described next.

5.4.2 Logging policy

As described in Section 5.3.1, the deployed logging policy π_0 logs the data on which we evaluate our candidate policies. The logging policy was trained via LinUCB [100], which is a state-of-the-art contextual bandit method, across all super-users and is expected to function well on average. We freeze the policy obtained with LinUCB and use it as our logging policy.

To ensure a logging policy that explores, we make the logging policy stochastic and give it full support (that is, every action has a non-zero probability). This is accomplished by using ϵ-greedy exploration (with $\epsilon > 0$) [154]; ϵ-greedy exploration selects an action uniformly at random with probability ϵ and the best action (according to the logging policy) with probability $(1 - \epsilon)$. The ϵ parameter allows us to trade off the exploration aggressiveness and the performance of the logging policy.

On the one hand, we want a logging policy that explores aggressively, so as to obtain as much information as possible in the logged feedback. On the other hand, we want a logging policy that performs well, as it is the only component that is exposed to users of the system and we would not want to hurt the user experience. We use $\epsilon = 0.2$ in our experiments, resulting in a policy that exploits 80% of the time and explores 20% of the time. Exploration is necessary for off-policy evaluation and $\epsilon = 0.2$ strikes a decent balance where the logging policy is expected to still function well. We definitely want to avoid $\epsilon = 0$ because it would result in a deterministic policy which is problematic for off-policy evaluation and we want to avoid $\epsilon = 1$ because it is unrealistic to expect a purely random policy to be deployed. In practice one would want to deploy a policy that mostly performs the best actions but performs a little bit of exploration, thus ϵ tends to be closer to 0 than 1.

5.4.3 Candidate policies

To perform off-policy evaluation we need a set of candidate policies. These are policies whose performance we wish to estimate. In our experiments, candidate policies are trained via LinUCB on each of the 10 super-users, thus, resulting in 10 candidate policies. Each of the 10 candidate policies is expected to work well when the hybrid user switches to the super-user the candidate policy was trained on, but is expected to underperform at any other point in time.

Table 5.1: The best parameters (in terms of minimizing MSE) for each estimator after a grid search.

Estimator	LastFM	Delicious
$V^{\tau IPS}$	$\tau = 10,000$	$\tau = 50,000$
$V^{\tau IPS}$ (adaptive)	$\tau = 10,000$	$\tau = 50,000$
	$\lambda = 0.00005$	$\lambda = 0.00001$
	$s = 50,000$	$s = 30,000$
$V^{\alpha IPS}$	$\alpha = 0.9999$	$\alpha = 0.99995$
$V^{\alpha IPS}$ (adaptive)	$\alpha = 0.99995$	$\alpha = 0.99995$
	$\lambda = 0.00005$	$\lambda = 0.00005$
	$s = 50,000$	$s = 100,000$

5.4.4 Ground-truth and metrics

To evaluate how well an estimator predicts a policy's performance we require a ground-truth, i.e., the true performance of a policy. The ground-truth can obtained by deploying the policy and measuring how well it actually performs [41]. According to our task definition, this cannot be done in practice as we have only one deployed logging policy, which does not change. However, in our experimental setup we have full control over the environment and so can simulate the deployment of any candidate policy and measure its true performance. This is done by, at any point in time t, running the candidate policy for 20,000 contextual bandit interactions (observing a context, playing an action and obtaining a reward) and then averaging the rewards.

To evaluate estimators, we measure the Mean Squared Error (MSE) between the estimated policy performance, given by the estimators, and the ground-truth, obtained as described above. The reported MSE values are averaged across the 10 candidate policies described in the previous section. Lower values of MSE correspond to better performance. To measure statistical significance, we run each experiment 20 times and compare the outcomes of the considered off-policy estimators using a paired two-tailed t-test.

5.4.5 Hyperparameters

Some of the estimators require setting a parameter. For example, $V^{\tau IPS}$ and $V^{\alpha IPS}$ require a window size τ and a decay rate α, respectively. The adaptive variants require us to set λ, which trades off variance and bias, and s, which is the Lipschitz estimation window. We found parameters that minimize MSE via a grid search. The final parameters are displayed in Table 5.1.[2]

[2]We are able to tune these parameters because we have access to the ground truth due to full experimental control. In practical deployment settings, A/B testing may be required to find good parameters.

Table 5.2: Mean Squared Error ($\times 10^{-3}$) on the LastFM dataset. Lower is better. We use $\triangledown$ and $\triangle$ to denote statistically significantly ($p < 0.01$) lower and higher MSE respectively compared to V^{IPS}. For the adaptive estimators we use $\blacktriangledown$ and $\blacktriangle$ to denote statistically significantly ($p < 0.01$) lower and higher MSE compared to their non-adaptive counterparts.

Estimator	Smooth	Abrupt	Stationary
V^{IPS}	6.029	7.787	1.183
$V^{\tau IPS}$	1.709 $^{\triangledown}$	3.041 $^{\triangledown}$	1.657 $^{\triangle}$
$V^{\tau IPS}$ (adaptive)	1.565 $^{\triangledown\blacktriangledown}$	2.881 $^{\triangledown\blacktriangledown}$	1.407 $^{\triangle\blacktriangledown}$
$V^{\alpha IPS}$	1.541 $^{\triangledown}$	2.981 $^{\triangledown}$	1.408 $^{\triangle}$
$V^{\alpha IPS}$ (adaptive)	1.546 $^{\triangledown}$	3.067 $^{\triangledown\blacktriangle}$	1.278 $^{\triangle\blacktriangledown}$

Table 5.3: Mean Squared Error ($\times 10^{-3}$) on the Delicious dataset. Lower is better. Statistical significance is denoted in the same way as in Table 5.2.

Estimator	Smooth	Abrupt	Stationary
V^{IPS}	0.312	0.469	0.022
$V^{\tau IPS}$	0.111 $^{\triangledown}$	0.268 $^{\triangledown}$	0.058 $^{\triangle}$
$V^{\tau IPS}$ (adaptive)	0.116 $^{\triangledown\blacktriangle}$	0.260 $^{\triangledown\blacktriangledown}$	0.045 $^{\triangle\blacktriangledown}$
$V^{\alpha IPS}$	0.099 $^{\triangledown}$	0.218 $^{\triangledown}$	0.070 $^{\triangle}$
$V^{\alpha IPS}$ (adaptive)	0.104 $^{\triangledown\blacktriangle}$	0.230 $^{\triangledown\blacktriangle}$	0.047 $^{\triangle\blacktriangledown}$

5.5 Results

In this section, we present the results of our empirical evaluation. We separate our results in four sections, each answering one of our research questions. The overall results of the proposed methods are presented in Tables 5.2 and 5.3. The figures for the sliding window estimator $V^{\tau IPS}$ and exponential decay estimator $V^{\alpha IPS}$ are very similar to each other, so we omit the figures for $V^{\tau IPS}$ and only include those for $V^{\alpha IPS}$.

5.5.1 Smooth non-stationarity

First, we look at RQ4.1:

> How well do the proposed estimators perform in a non-stationary environment?

The first column of Tables 5.2 and 5.3 shows that the proposed $V^{\tau IPS}$ and $V^{\alpha IPS}$ off-policy estimators have significantly lower MSE than the standard V^{IPS} estimator, being three times more effective in estimating the actual performance of a recommendation policy on both the LastFM and Delicious datasets. Note that the rewards on the LastFM dataset are higher than those on the Delicious dataset, which is in line with the results of [179] and can be attributed to the fact that it is easier to recommend correct artists

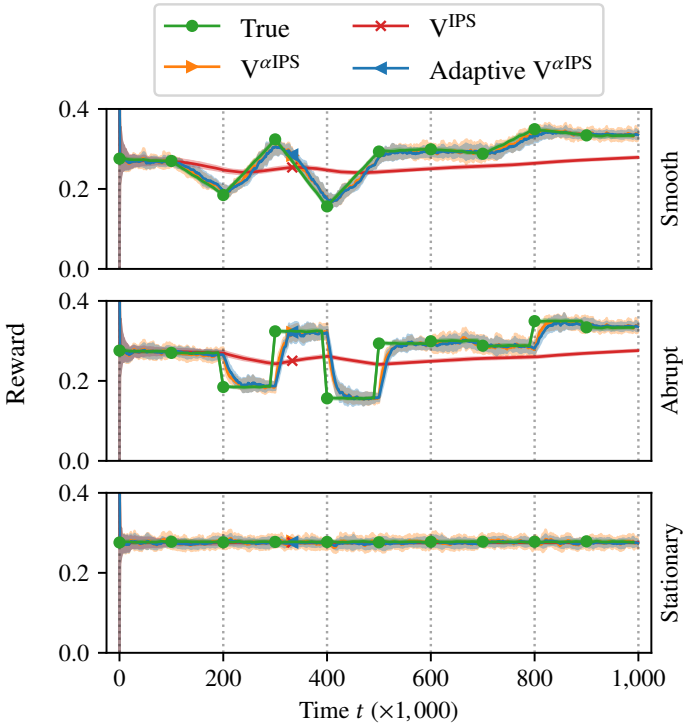

Figure 5.1: Exponential decay estimators in a smooth (top row), abrupt (middle row) and stationary (bottom row) setting on the LastFM dataset. The shaded areas indicate the standard deviation across 20 runs.

(and, thus, accumulate higher reward) than to recommend correct websites, because the number of artists is smaller than the number of websites.

To better understand the behavior of the proposed off-policy estimators over time, we plot the actual and estimated rewards of one of the 10 candidate policies in Figures 5.1 and 5.2 (the choice of a policy is not important, we use policy 6 in all figures). We only present figures for the exponential decay estimator $V^{\alpha IPS}$ here; the figures for the sliding window estimator $V^{\tau IPS}$ are similar.

The top plots in Figures 5.1 and 5.2 show that the $V^{\alpha IPS}$ estimator closely follows the actual performance of a recommendation policy on both the LastFM and Delicious datasets. The standard V^{IPS} estimator, instead, fails to approximate the policy's actual performance and accumulates a large amount of bias.

The adaptive variants of our proposed off-policy estimators perform similarly to their non-adaptive counterparts, outperforming or underperforming the latter in a few cases (see Tables 5.2 and 5.3). This means that in the smooth non-stationary setup we can use either type of estimator. Below we will show that in other setups adaptive estimators should be preferred over non-adaptive ones.

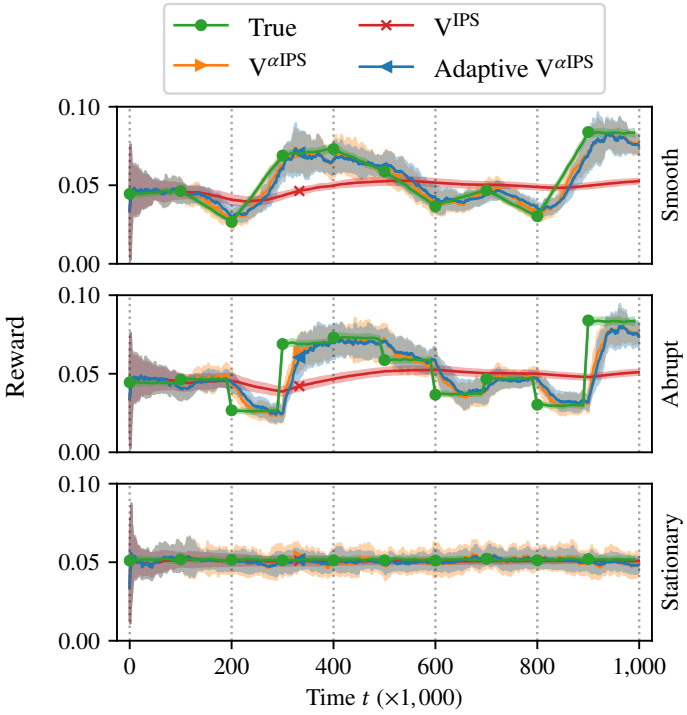

Figure 5.2: Exponential decay estimators in a smooth (top row), abrupt (middle row) and stationary (bottom row) setting on the Delicious dataset. The shaded areas indicate the standard deviation across 20 runs.

5.5.2 Abrupt non-stationarity

Our work builds on the assumption of a smooth non-stationary environment, one where the world changes slowly over time. We wish to investigate how well our estimators work when this assumption is violated, i.e., when the world behaves in an abrupt non-stationary way. This leads to RQ4.2:

> How well do the estimators function when Assumption 5.3.1 is violated? E.g., when the environment changes abruptly?

The second column of Tables 5.2 and 5.3 indicates that in the abrupt non-stationary setup the MSE of $V^{\tau IPS}$ and $V^{\alpha IPS}$ is about two times lower than the MSE of V^{IPS} (all differences are statistically significant). This shows that our proposed estimators approximate the actual performance of a policy well even when the theoretical upper bounds on the estimators' bias are no longer valid.

The second row of Figures 5.1 and 5.2 further confirms this by showing that the $V^{\alpha IPS}$ estimator closely follows the true reward of a policy even if the changes in the environment are abrupt. The standard V^{IPS} estimator still cannot follow the true reward in this setup.

5.5.3 Stationary environment

In this section we answer RQ4.3:

> Can the proposed estimators be applied to stationary environments?

In the stationary environment, the regular V^{IPS} estimator is guaranteed to perform the best: in this setup it is unbiased and has variance that goes to zero when t grows [40]. Our proposed estimators are also unbiased in the stationary environment, but their variance does not decrease over time. Thus, we expect the V^{IPS} estimator to outperform $V^{\tau IPS}$ and $V^{\alpha IPS}$ in the stationary setup.

The above intuitions are confirmed by the results in the last column of Tables 5.2 and 5.3. The V^{IPS} estimator indeed has the lowest MSE compared to all other estimators. Interestingly, $V^{\tau IPS}$ and $V^{\alpha IPS}$ are also able to approximate the true reward of a policy relatively well. Particularly, the MSE of $V^{\tau IPS}$ and $V^{\alpha IPS}$ on the LastFM dataset is at most 0.4 times higher than the MSE of V^{IPS} (recall, that V^{IPS} has 2–3 times higher MSE in the non-stationary setups). On the Delicious dataset the differences in MSE are larger, but the absolute MSE values are an order of magnitude smaller than in the non-stationary setups. The adaptive variants of our estimators are significantly better than the non-adaptive ones in the stationary environment, having much lower MSE: the adaptive variants are able to detect the stationary situation, adapt their parameters appropriately and reduce their overall variance.

Thus, we can conclude that although designed for non-stationary environments, the $V^{\tau IPS}$ and $V^{\alpha IPS}$ estimators, and especially their adaptive variants, can be applied in stationary environments. This is further confirmed by the bottom plots in Figures 5.1 and 5.2, where all estimators closely follow the true (stationary) reward.

5.5.4 Impact of parameters

Finally, we look at RQ4.4:

> How do the estimators behave under different parameters?

To answer this question, we have investigated different parameter settings for τ, α and λ. In Figure 5.3, we plot the true and estimated rewards for different values of α for the $V^{\alpha IPS}$ estimator on the LastFM dataset. The observations for τ and λ are very similar, so we omit these results to save space. From Figure 5.3, we see that setting α is a trade-off in bias and variance. This is in line with our theoretical results (Section 5.3), which state that as α approaches 1, we expect lower variance but higher bias, and vice versa for $\alpha \to 0$. The same results hold for the window size τ (a higher value causes lower variance and higher bias) and the λ parameter, which, by design, trades off variance and bias.

5.6 Conclusion

In this chapter we studied *non-stationary* off-policy evaluation. We showed that in non-stationary environments the traditional IPS off-policy estimator fails to approximate the

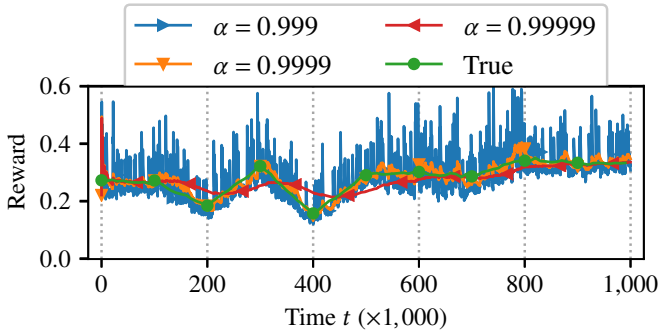

Figure 5.3: Impact of the α parameter on the exponential decay estimator $\mathrm{V}^{\alpha\mathrm{IPS}}$ in a smooth non-stationary setting on the LastFM dataset.

true performance of a recommendation policy and suffers from a large bias that grows over time. To address the problem of non-stationary off-policy evaluation, we proposed two estimators that closely follow the changes in the true performance of a policy: one using a sliding window average and one using an exponential decay average. Our analysis of the proposed estimators shows that their bias does not grow over time and can be bounded. The bias of our estimators can be controlled by the window size τ and the decay rate α. Using the results of our analysis, we proposed a principled way to adapt τ and α automatically according to the changing environment.

We evaluated the proposed estimators in non-stationary recommendation environments using the LastFM and Delicious data sets. The experimental results show that our estimators approximate the policy's actual performance well, having MSE that is 2–3 times lower than that of the standard IPS estimator. We showed that these results hold not only in smooth non-stationary environments, where we can derive upper bounds on the bias of our estimators, but also in the abrupt non-stationary setup, where the theory does not hold. Finally, our results suggest that the proposed off-policy estimators, although designed for non-stationary environments, can be applied in the stationary setup with adaptive variants of the proposed estimators being particularly effective. These findings open up the way for off-policy evaluation to be applied to practical non-stationary real-world scenarios.

An interesting direction for future work is to investigate the use of more advanced off-policy estimators such as Doubly Robust [40] or Switch [175] in non-stationary environments. We hypothesize that such estimators will also suffer from a large bias, while the moving average estimators will be able to solve this issue.

All chapters so far have considered clicks on recommended items or ranked lists. In Chapter 6 we will go beyond traditional search logs and investigate how *activity logs* can be used for learning semantic matching models for ranking.

6 Using Activity Logs for Learning to Rank

6.1 Introduction

Cloud storage platforms, such as Dropbox or Google Drive, are widely used as a means for storing, editing and sharing personal and organizational documents. They are characterized by storing documents in the cloud and facilitating easy assess across different devices. Unlike traditional web-based search, *cloud storage search* is challenging because it involves searching through private collections of documents. Due to the private nature of the documents, it is impossible to gather relevance judgments from professional annotators, instead necessitating the use of click logs [172]. As such, existing work has extensively explored the use of click logs [76, 135, 172, 173]. Most of this existing work focuses on extracting features and labels from large-scale search logs and using those to train a ranking model.

Semantic matching models are effective at matching queries to documents in situations where traditional lexical matching features fail due to the vocabulary gap [168]. Recent work on semantic matching models use neural networks to improve ranking quality [52]. However, training these models requires large amounts of labeled data. Consequently, semantic matching models may fail to generalize when sufficiently large search logs are not available. In this chapter we investigate *non*-search user interaction data with which we can train semantic matching models: *user activity logs*.

Our goal is to improve the search ranking quality for cloud storage platforms by utilizing *user activity logs*, which record user's interactions with the cloud storage platform. Examples of such interactions include opening, editing or sharing a document. In contrast to search logs, activity logs contain a richer set of interactions beyond clicks, span more than just the search component, and are available in much more abundant quantity. As demonstrated in several user studies on personal search [42, 45], activity-based signals such as the recency and the context in which the documents are accessed play an important role in determining the relevance of the document to the tasks that the user may be working on.

Despite the fact that activity logs are more abundant than search logs, using them

This chapter is based on [73], which is currently under review.

to train semantic matching models is not trivial. In contrast to search logs, activity logs are not necessarily tailored towards ranking and activity-trained semantic matching models may not work well for improving search quality. To overcome this, we propose to use *co-access*, a signal that measures whether two documents are accessed in sequence and within a short time span as a proxy for relevance; we then use this information as a weakly supervised label for training text embeddings and semantic similarity models.

The goal in this chapter is to answer the following research question:

RQ5 Can activity-based user interaction signals improve ranking quality?

Our experiments using large-scale Google Drive search and activity data show that semantic matching models trained with co-access can improve ranking performance significantly compared to lexical matching and semantic matching baselines that are *not* trained on activity logs. Furthermore, we show that incorporating activity-trained semantic matching models with strong hand-crafted features further improves ranking performance, especially in cases where hand-crafted features alone do not provide sufficient signals. Finally, our results show that, by leveraging activity logs, practitioners of cloud storage systems can train effective ranking models without requiring a large amount of search logs. This shows that activity logs can significantly help when search logs are not available in abundant quantity.

To the best of our knowledge, this is the first work to examine the benefits of leveraging non-search related document usage activity for cloud storage search in a large-scale production system. We demonstrate that such activity can be used to not only provide features to ranking models, but also to derive semantic similarity models that improve lexical text matching. As such, this work can shed light on employing user activity in scenarios where direct collection of search-specific interactions (e.g., query and click logs) may be expensive or infeasible, or in cloud storage services that have no existing search functionality.

6.2 Related Work

Cloud storage is widely used by individuals and organizations as a means of storing, organizing and sharing documents. Cloud storage faces several research challenges, for example: scaling to large traffic volume [38], anonymizing data [159], and optimizing search [16]. Cloud storage search is related to email search [16, 23], enterprise domain search [167], desktop search [91], and other personal search problems [42, 45] in that they typically involve searching through private corpora; however, it is also unique in that rich non-search user activities, such as opening, editing and sharing documents, can be logged at large scale in cloud storage systems, which inspires our work.

Learning to Rank (LTR) is a widely used approach to optimize search engine quality. Recent research on LTR focuses on learning from user interactions [76, 105, 172]. State-of-the-art approaches for optimizing search rely on combining large sets of high-quality hand-crafted features into a well performing model. Tree-based methods, specifically LambdaMart [21, 178], excel at combining high-quality features into a single strong performing model. LambdaMart has enjoyed considerable success in

the past and is still considered state-of-the-art [174]. In particular, LambdaMart has repeatedly outperformed all other models on public benchmarks, where the number of training examples is limited [28, 90], even with the advancement of neural ranking methods [19]. Therefore, we use it as the ranking model of choice in this chapter.

More recently, advances in neural networks enabled significant progress on semantic matching models for information retrieval (see [110] for a recent comprehensive survey on this topic). Most generally, these semantic matching models use either search logs [52, 63, 150, 150, 185] or weak supervision [31, 34] to learn text embeddings that best capture *query-document* similarities. Subsequent work introduced multiple advanced variations on these semantic matching models include (among many others) convolutional networks [33] and kernel pooling [181]. However, note that these advanced method cannot be readily applied to our heavily reduced dataset, which, due to privacy constraints, does not preserve word ordering (see Section 6.3.3).

In contrast to prior work on semantic matching models, text embeddings based on *user activity* do not require learning query-document similarities. This is a highly desirable property for cloud storage systems where either no existing search solutions exist or search logs are sparse due to low search volume and / or private nature of search intents. For instance, a study by Ai et al. [8] shows that email search queries are much shorter than those in web search, and are often based on a particular email metadata (e.g., sender or received / sent date). Such queries are less likely to generate robust generalizable semantic matching models.

User activity has been extensively used for user modeling [12, 113], predicting future user behavior [36, 160], or document retrieval [42, 45] to name just a few. However, to the best of our knowledge, there is no prior work on learning semantic matching models from activity logs at large scale, especially in the context of cloud storage.

6.3 Problem Setting

6.3.1 Cloud storage search

Cloud storage search can be formulated as a Learning to Rank (LTR) problem. More formally, we wish to learn a ranking function $f(q, d)$, which, for given query q and document d, produces a score such that relevant documents are assigned high scores and less relevant documents are assigned low scores. The learned function can then be applied to a query q and a collection of candidate documents $D = \{d_1, d_2, \ldots, d_n\}$ to rank the documents by relevance, placing highly relevant documents at the top.

6.3.2 Solution overview

In this chapter we focus on improving the top-5 ranking of the search component of Google Drive (see Figure 6.1). We provide an overview of our solution in Figure 6.2. More specifically, we train a LambdaMart [178] model on a click log spanning several weeks. We chose to use a Gradient Boosted Decision Tree (GBDT) approach as these methods have demonstrated very strong ranking performance [178], are computationally easy to scale, are robust against outliers and can naturally handle input of varying

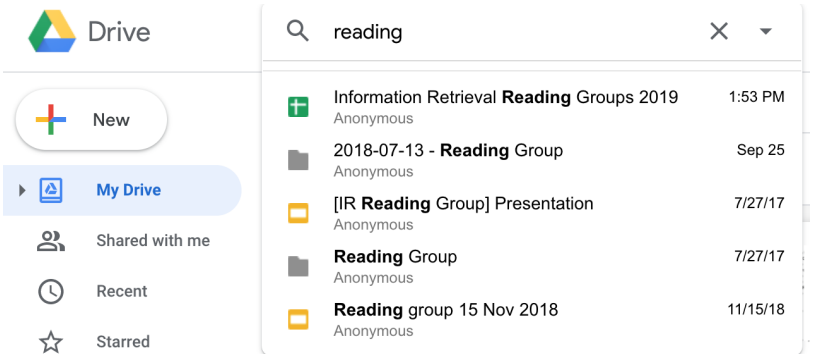

Figure 6.1: An example of Google Drive's top-5 ranking list.

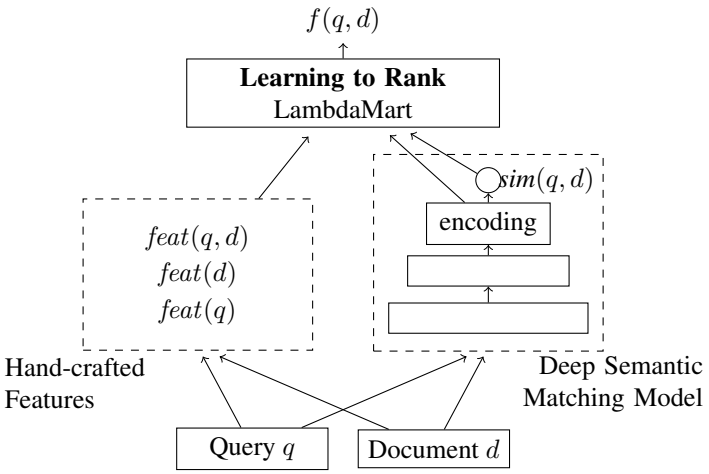

Figure 6.2: An overview of the proposed solution for cloud storage search. On the left we compute hand-crafted features for given query q and document d. On the right we apply a deep semantic matching model to produce both an encoding (the last hidden layer) and a similarity score $sim(q, d)$. Both the hand-crafted and learned features are combined in an LTR model.

distributions [44]. We incorporate the output of activity-trained semantic similarity models as features for training the GBDT.

6.3.3 Privacy

Google Drive contains private document corpora and requires special treatment to protect user privacy. For this reason, the data we use for the experiments are k-anonymized [159] and are inaccessible to individual engineers. Moreover, for our proposed methods, we limit the use of text content to the document titles, not the full content. The titles are also k-anonymized, i.e., only frequent words used by sufficiently many users in the corpus are retained with no word sequence information preserved. Note that this limits our choices of text embedding models to word-level embeddings. Some of our baseline models / features are based on document full content, but these models / features are computed on the fly – document content is never materialized for model training.

6.4 Using Activity Logs for Semantic Matching

Directly applying existing semantic matching models on query logs for cloud storage search is not trivial because click logs for cloud storage search may not be available in sufficient quantity to learn effective semantic matching models, and training them with too few data points may lead to poor generalization. To overcome this limitation we propose to use a different, more abundant, source of data for training semantic matching models: *activity logs*. Our goal is to learn semantic matching models from activity logs and apply them for ranking.

6.4.1 Semantic matching models

A Semantic Matching Model (SMM) learns to compute the semantic similarity of two pieces of text (typically a query and document for search problems) [52, 63]. The model takes a text pair (t, t') as input and produces $sim(t, t')$, a score indicating the similarity between the two texts, as output. This formulation is quite general as the text pair (t, t') could be any two pieces of short text. For example, in ranking we would use a query q and document d to predict the similarity between the query and document: $sim(q, d)$. Similarly, if we were interested in modeling document similarity we could use two documents (d, d') as input and predict their similarity: $sim(d, d')$. As we will see later, this general formulation is beneficial as we will train our semantic matching models on *document pairs* but apply them on *queries and documents* when ranking.

To train semantic matching models we would need to obtain a dataset containing textual pairs and their respective relevance labels. It is not possible to collect labeled examples from professional annotators due to the privacy considerations when dealing with cloud storage data. Even for cases where collecting labels from human annotators is possible, conducting such an annotation process at large scale could be too expensive and infeasible. Our proposed solution to this is to extract relevance labels automatically from users' activity logs.

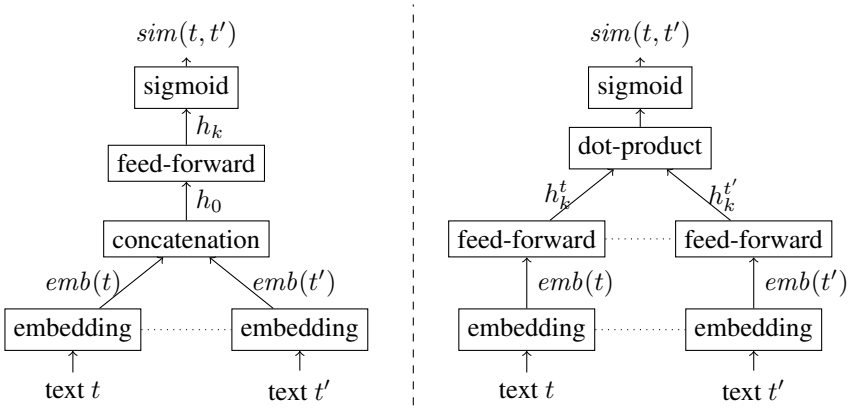

Figure 6.3: On the left is the Concatenation Semantic Matching Model (CONCAT) and on the right is the Siamese Semantic Matching Model (SIAM). Dotted lines indicate shared weights between components.

We emphasize that the specific architecture of the SMM is not the focus of this chapter, nor do we claim that any specific architectures we use are novel. Instead we are interested in understanding how one can learn semantic matching models from *activity data*. We use two popular architectures, a concatenation semantic matching model and a Siamese semantic matching model, described briefly below.

First, we describe the Concatenation Semantic Matching Model (CONCAT): This model first computes an embedding for the texts t and t'. Each character n-gram in the text is mapped to an embedding, where only the most frequent n-grams are retained to limit the vocabulary and make the problem computationally feasible. The character n-grams for t and t' are then averaged to obtain $emb(t)$ and $emb(t')$ respectively. The representations are concatenated to obtain:

$$h_0 = [emb(t), emb(t')] .$$

Note that more advanced text encoders such as recurrent neural networks and transformers are not applicable to our problem. This is because, as mentioned in Section 6.3.3, to protect user privacy, we are only allowed to use k-anonymized words with no sequence information for model training.

This joint representation is then passed through a series of dense feed-forward layers, where each layer h_i is defined as:

$$h_i = \phi(W_i h_{i-1} + b_i),$$

where ϕ is an activation function such as ReLU [112] or $\tanh$. Finally, the last layer of this feed-forward neural network, h_k, is reduced to a scalar value and mapped to a probability via a sigmoid function:

$$sim(t, t') = \text{sigmoid}(W_{final} h_k + b_{final}).$$

Next, we describe the Siamese Semantic Matching Model (SIAM): This model embeds the texts t and t' to their respective representations $emb(t)$ and $emb(t')$. The process of producing these embeddings is exactly the same as in the CONCAT. Next, these representations are passed through a shared feed-forward neural network. More formally:

$$h_0^t = emb(t), \qquad h_0^{t'} = emb(t'),$$
$$h_i^t = \phi(W_i h_{i-1}^t + b_i), \quad h_i^{t'} = \phi(W_i h_{i-1}^{t'} + b_i).$$

Note that the weights W_i and b_i at each layer are shared for both t and t'. The output vectors of the last layer, h_k^t and $h_k^{t'}$, are joined via dot product and then passed through a sigmoid to obtain:

$$sim(t, t') = sigmoid\left(h_k^t \cdot h_k^{t'}\right).$$

To apply the two SMMs for ranking, we feed the given query q and document d into the models, and extract the following output and hidden states as features for a LTR model: (a) the predicted semantic similarity, $sim(q, d)$; (b) the last hidden layer of CONCAT, h_k, which provides a richer representation for the query-document pair; and (c) the last hidden layers in the towers of SIAM, h_k^q and h_k^d, which encode q and d respectively. This is also illustrated in Figure 6.2.

6.4.2 Weak supervision using co-access labels

In order to train semantic matching models we would like to obtain relevance labels $y_{t,t'} \in \{0, 1\}$ that indicate whether text t is semantically related to text t'. We take a weakly-supervised learning approach, and propose a *co-access* label, with the assumption that it can serve as a proxy for relevance.

We say that two documents are *co-accessed* iff a user opens the two documents *in sequence* and *within a k-minute time window*. We illustrate this concept in Figure 6.4 with $k = 2$ *minutes*.[1] While there are multiple alternative ways to design the co-access label, we found that the method proposed here is conceptually simple, yet empirically effective. The co-access label is motivated by the fact that users often open multiple related documents in a single session, yet it strives to reduce the number of false positive labels by keeping a narrow time window and discarding non-consecutive co-accesses.

The following procedure is employed to collect training examples with co-access labels: (1) We first sample segments of a user's activity logs, which we call *activity segments*. Each activity segment contains events from the same user in a consecutive time window; (2) For each activity segment, we collect a set of documents the user accessed, $D = \{d_i\}_{i=1}^{|D|}$. From the document set, we then collect all the unordered pairs of documents in the document set, $\mathcal{P}_D = \{\{d, d'\} \mid d, d' \in D \wedge d \neq d'\}$; and (3) we extract co-access labels for all the document pairs, and the co-access label $y_{d,d'}$ is defined as,

$$y_{d,d'} = \begin{cases} 1, & co\_accesses(d, d') > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{6.1}$$

[1] During our initial investigations we found that co-accesses have a long tail distribution, with 70% of co-accesses occurring within the two minute window. Therefore, we fix the co-access time window to two minutes in the remainder of this chapter.

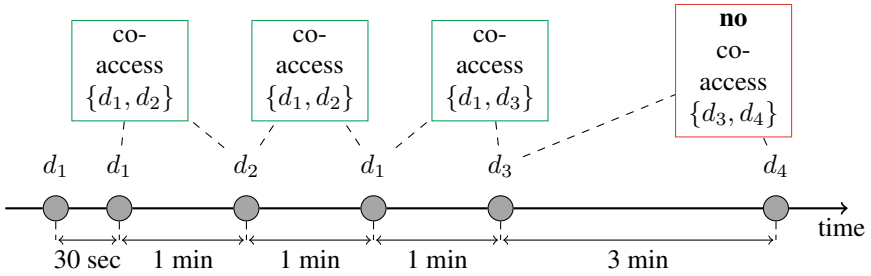

Figure 6.4: Co-access as a label for document similarity. Document pair $\{d_1, d_2\}$ is co-accessed twice, document pair $\{d_1, d_3\}$ is co-accessed once, and document pair $\{d_3, d_4\}$ is not co-accessed since their co-access time is > 2 minutes.

where $co\_accesses(d, d')$ is the number of co-access events between d and d' in the activity segment. To give an example, for the activity segment shown in Figure 6.4, the collected document set is $D = \{d_1, d_2, d_3, d_4\}$ and the extracted co-access labels for all the document pairs are: $y_{d_1,d_2} = y_{d_1,d_3} = 1$, and, $y_{d_1,d_4} = y_{d_2,d_3} = y_{d_2,d_4} = y_{d_3,d_4} = 0$. This procedure yields a training dataset

$$\mathcal{D} = \left\{ (d, d', y_{d,d'}) \mid \{d, d'\} \in \bigcup_{i=1}^{N} \mathcal{P}_D^{(i)} \right\}$$

from a large number of activity segments, where N is the number of the segments. We then train a semantic matching model by minimizing the weighted cross-entropy loss defined as follows,

$$- \sum_{(d,d',y_{d,d'}) \in \mathcal{D}} y_{d,d'} \log(sim(d, d')) + \lambda(1 - y_{d,d'}) \log(1 - sim(d, d')), \qquad (6.2)$$

in which we use document titles as the text representations for each document when scoring $sim(d, d')$, and use $\lambda \in (0, 1]$ as a hyper parameter to down-weight the loss for negative document pairs. We use this weighting to address data imbalance problems in the dataset – in practice only a small percentage of document pairs are co-accessed and the majority of the document pairs are not co-accessed.

6.5 Experimental Setup

6.5.1 Data collection

We collect two datasets from the search and activity logs of Google Drive: (1) Co-access dataset $\mathcal{D}_c$ used for building our Semantic Matching Models, and (2) Search dataset $\mathcal{D}_s$ used mainly for building learning to rank models. We describe them in more detail below.

We collect the co-access dataset $\mathcal{D}_c$ from activity logs as follows: First, we randomly sample different sets of users for training, validation and testing; Then, for each

Table 6.1: Statistics for co-access dataset $\mathcal{D}_c$. The $train$, $vali$, $test$ superscript indicates the dataset is for training, validation or testing respectively.

Data split	#segments	#pairs	% co-accessed
$\mathcal{D}_c^{train}$	44.3M	448M	10.5%
$\mathcal{D}_c^{vali}$	20.6K	205K	10.3%
$\mathcal{D}_c^{test}$	20.6K	205K	10.2%

user, we sample 10 segments of the user's event stream from the activity logs. Each segment contains activity events of 3 consecutive weeks. We sample the start timestamps of the activity segments uniformly within a 2-week time window. We filter out segments that do not contain sufficient activity events, i.e., less than 75 events. Finally, from each activity segment, we collect up to n most recently accessed documents from the first two weeks of the segment. For each collected document pair, we extract document titles and co-access labels from the last week of the segment (i.e., whether the two documents are co-accessed in that week). The size of the document pair set could be very large – up to $n \cdot (n-1)/2$ pairs. To improve efficiency as well as to reduce noise, we filter the document pairs by requiring them to be co-accessed in the first two weeks of the segment. This reduces the size of document pair set dramatically to around 10 per segment on average.

Table 6.1 shows some statistics for the collected co-access dataset for training, validation and testing. The table reports the number of activity segments and document pairs. Note that the training set $\mathcal{D}_c^{train}$ is fairly large. This is because we can easily extract the co-access data from a large set of eligible Google Drive users and sample multiple activity segments from their activity logs. The table also reports the percentages of co-accessed document pairs (postive rate), which are quite consistent across the training, validation and testing sets. Note that the positive rates are only around 10%. To address this data imbalance issue, we down-weight the negative document pairs when training our models as described in Section 6.4.2. Moreover, the dataset is k-anonymized to protect user privacy as described in Section 6.3.3.

The search dataset $\mathcal{D}_s$ comprises a set of queries and clicks collected from the Google Drive search logs over a period of several weeks. Here we select dates that are after the dates of the co-access dataset $\mathcal{D}_c$ to prevent any potential data peeking issues when training our LTR models on $\mathcal{D}_s$. We then extract all the features (see Section 6.5.4) and search clicks for the sampled queries from the search logs and activity logs. Each query is associated with about 5 documents on average, which is a direct result of the search user interface. We discard all the queries without clicks. We then split the data into training, validation and testing set by dates, using the earlier dates for training, later dates for validation and the latest dates for testing. This data split prevents accidental leaking of future test queries and documents into the training set. In total, we collected 31,421 queries for training, 25,412 queries for validation and 24,595 queries for testing.

6.5.2 Evaluation

We evaluate the performance of each ranking model using Mean Reciprocal Rank (MRR) and Negative Average Click Position (NACP), which are defined as,

$$\text{MRR} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \frac{1}{rank_q}, \quad \text{NACP} = -\frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} rank_q,$$

where $\mathcal{Q} = \{q\}$ is the evaluation query set, $rank_q$ is the rank of the first clicked document for query q. To address click position bias, we use propensity-weighted MRR and NACP which tend to be more consistent with online experiment results [102, 172].

We tune each ranking model on the search validation dataset, and report evaluation results on the test dataset. We performed statistical significance tests using paired t-test with 0.01 as the p-value threshold. In all tables we denote a statistically significant increase and decrease compared to the baseline with $\triangle$ and $\triangledown$ respectively.

6.5.3 Baselines

We compare the following semantic matching baseline models with ours. Note that more advanced models like recurrent neural networks and transformers are not applicable to our problem. As mentioned in Section 6.3.3, to protect user privacy, our dataset only contains k-anonymized words and character n-grams extracted from these words retain no sequence information for model training.

W2V We compare our work to Word2Vec (W2V) [109] since both methods train text embeddings without needing access to search logs. We use a straightforward approach to apply Word2Vec to ranking: we compute the Word2Vec embeddings for query q and document d, producing $emb(q)$ and $emb(d)$. We can then take the dot product of these embeddings to produce a similarity score:

$$sim(q, d) = emb(q) \cdot emb(d).$$

We train the embeddings using the Word2Vec skip-gram approach on the 107 million document titles extracted from the search dataset $\mathcal{D}_s$. We retain the 500,000 most frequent n-grams as the vocabulary and choose 90 as the dimensionality of these embeddings. We use the full document title as the context, since the text does not retain word sequence information.

DSSM The Deep Structured Semantic Model (DSSM) [63] is a commonly used semantic matching model for ranking and we include it in our comparison. The DSSM architecture is implemented using TF-Ranking [126], and is trained on the search dataset $\mathcal{D}_s$. We choose the softmax ranking loss function as our training objective [126]. We use a dimension size of 159 for the character n-gram embeddings and ReLU as the activation function. The hidden layer dimensions, learning rate, dropout rate and vocabulary size are tuned via grid search.

6.5.4 Features

We compare different sets of features combined in LambdaMART. We list all the studied features in Table 6.2 and describe them in groups below.

Table 6.2: The full set of features used to train LTR models.

Feature	Description
Lexical text matching features (TM)	
Overlap	# of query terms in the document title.
Normalized	Same as *Overlap*, but after lexical normalization.
BM25V	A variant of BM25 [141]
Activity-based features (ACT)	
Last Access	Time since last access.
Last Edit	Time since last edit.
Doc Age	Time since the document was created.
Activity-based semantic matching models (Section 6.4)	
CONCAT$_{sim}$	Output of CONCAT: $sim(q, d)$.
CONCAT$_{rep}$	State of last hidden layer of CONCAT: h_k.
SIAM$_{sim}$	Output of SIAM: $sim(q, d)$.
SIAM$_{rep}$	State of last hidden layers of SIAM: h_k^q, h_k^d.
Other semantic matching models (Section 6.5.3)	
W2V$_{sim}$	Output of W2V: $sim(q, d)$.
W2V$_{rep}$	Embeddings of q and d: $emb(q)$, $emb(d)$.
DSSM$_{sim}$	Output of DSSM: $sim(q, d)$.
DSSM$_{rep}$	State of last hidden layer of DSSM: h_k.

Lexical text matching features (TM) These features perform various forms of lexical matching of the query and document.

Activity-based features (ACT) We use three simple but effective activity-based features, that characterize document recency, namely *Last Access*, *Last Edit*, and, *Doc Age*.

Semantic matching features We extract the similarity scores as well as the internal representations from our proposed and baseline Semantic Matching Models. The similarity scores and internal representations are denoted with subscript *sim* and *rep* respectively in Table 6.2. Using both *sim* and *rep* is denoted using subscript *both* in our experiment result tables, e.g., CONCAT$_{both}$ includes both CONCAT$_{sim}$ and CONCAT$_{rep}$. We use a dimension size of 159 for char n-gram embeddings. We cap the vocabulary size at 500k by mapping less frequent char n-gram to out-of-vocabulary embeddings. We tune the hidden layer size, learning rate and activation functions.

Table 6.3: Ranking performance of Semantic Matching Models and activity-based features. The table reports relative performance with respect to $W2V_{sim}$ baseline.

Model	MRR	NACP
$W2V_{sim}$	+0.00%	+0.00%
$DSSM_{sim}$	+1.64%$^{\triangle}$	+2.92%$^{\triangle}$
$SIAM_{sim}$	+6.40%$^{\triangle}$	+6.34%$^{\triangle}$
$CONCAT_{sim}$	-5.56%$^{\triangledown}$	-3.28%$^{\triangledown}$
Last Access	-1.10%	+1.84%
Doc Age	+2.01%$^{\triangle}$	+2.32%$^{\triangle}$
Last Edit	+2.08%$^{\triangle}$	+2.45%$^{\triangle}$
BM25V	+12.74%$^{\triangle}$	+11.76%$^{\triangle}$
TM	+15.90%$^{\triangle}$	+14.77%$^{\triangle}$

6.6 Results

6.6.1 Comparing individual models

In this section, we study the ranking performance of the Semantic Matching Models when used independently. Our results are displayed in Table 6.3. *TM* is a fine-tuned combination of all the lexical text matching features in Table 6.2 used by the production system. All of these models, with the exception of $DSSM_{sim}$ and *TM*, do not require any search training data.

First, we observe that the activity-based features perform reasonably well, reaching levels similar to Semantic Matching Models such as W2V and in the cases of *Doc Age* and *Last Edit*, performing even better. These results seem surprising as features like *Doc Age* and *Last Edit* are comparatively simple features. However, this can be explained by the fact that our task is to re-rank the top 5 results in Google Drive. Nevertheless, these findings reinforce our belief that activity-based signals can be useful for search tasks.

Second, we see that supervised embeddings DSSM and weakly supervised embeddings SIAM are more effective than the self-supervised W2V approach. The CONCAT model underperforms significantly, especially when compared to the very similar SIAM model. We hypothesize that the SIAM architecture has an inductive bias by forcing the representations of both query and document to be directly comparable via dot product. Conversely, for the CONCAT model, the concatenation of the embeddings allows the model to learn complex feature interactions between the texts t and t', which may learn useful patterns for co-access, but likely does not generalize well to generic semantic similarity tasks and also not well to ranking.

To further analyze this behavior we investigate to what extent the different semantic matching models are correlated with the lexical text matching and activity-based features. In Table 6.4 we present the Pearson correlation between each model's similarity score output and those features. The findings here suggest that W2V has the

Table 6.4: Pearson Correlation between semantic matching scores and both lexical matching and activity-based features.

Model	*BM25V*	*Normalized*	*DocAge*	*LastEdit*
W2V_{sim}	0.3014	0.3798	0.0168	0.0104
SIAM_{sim}	0.2347	0.3044	-0.1256	-0.1183
CONCAT_{sim}	0.0799	0.1126	-0.1478	-0.1305
DSSM_{sim}	0.1784	0.2401	-0.0792	-0.0774

Table 6.5: Ranking performance (relative to the *TM* baseline) of Semantic Matching Models when combined with lexical text matching features.

Model	MRR	NACP
TM	+0.00%	+0.00%
$TM + \text{W2V}_{both}$	+1.46%$^{\Delta}$	+1.52%$^{\Delta}$
$TM + \text{DSSM}_{both}$	+1.59%$^{\Delta}$	+2.08%$^{\Delta}$
$TM + \text{CONCAT}_{both}$	+2.67%$^{\Delta}$	+3.53%$^{\Delta}$
$TM + \text{SIAM}_{both}$	+2.88%$^{\Delta}$	+3.37%$^{\Delta}$

strongest correlation with lexical matching features but does not correlate at all with activity-based features, which is expected as the W2V model is trained on unlabeled text corpora not on any user logs. Furthermore, we see that CONCAT correlates weakly with lexical matching but much more strongly with activity-based features, compared with other semantic matching models. This indicates that the CONCAT architecture could capture activity-based patterns more than the other Semantic Matching Models. Finally, it seems that SIAM correlates strongly with both lexical matching features *and* activity-based features (when compared to other models), which explains its strong performance in Table 6.3.

Finally, it is clear that none of the semantic similarity models beat the simple ad-hoc retrieval baselines *BM25V* and *TM*. Because of the strong performance of the lexical matching features, we next investigate whether Semantic Matching Models could improve ranking performance on top of *TM* in the next section.

6.6.2 Combined with lexical matching features

We incorporate the semantic matching models with the text matching features in a LambdaMart ranker. The results are displayed in Table 6.5. For all Semantic Matching Models, we find that adding both the similarity score (*sim*) and the representation (*rep*) as features performs the best. Therefore we only report results when using both *sim* and *rep* features (denoted by subscript *both*) in the rest of this chapter.

In all cases, incorporating Semantic Matching Models together with lexical text matching features improves the ranking performance. Moreover, we observe that our weakly supervised activity-trained semantic models (CONCAT and SIAM) significantly outperform (p-value < 0.01, t-test) both the unsupervised (W2V) and the supervised

Table 6.6: Ranking performance of Semantic Matching Models (relative to the *TM* + *ACT* baseline) when combined with lexical text matching features and activity-based features.

Model	MRR	NACP
TM + *ACT*	+0.00%	+0.00%
TM + *ACT* + W2V$_{both}$	+0.15%	+0.35%$^{\triangle}$
TM + *ACT* + DSSM$_{both}$	-1.39%$^{\triangledown}$	-2.18%$^{\triangledown}$
TM + *ACT* + CONCAT$_{both}$	+0.46%$^{\triangle}$	+0.93%$^{\triangle}$
TM + *ACT* + SIAM$_{both}$	+0.13%	+0.29%

(DSSM) methods. We believe this could be explained by the fact that our Semantic Matching Models could capture some activity-based patterns in addition to semantic similarity, as we discussed in the correlation analysis (Table 6.4) of Section 6.6.2. Next, we find there does not seem to be a discernible difference between the CONCAT and SIAM architectures. This indicates that the difference in ranking performance between CONCAT and SIAM that we observed in the previous section (see Table 6.3) is largely overcome by incorporating text matching features in the model.

6.6.3 Combined with all hand-crafted features

In this section we explore what benefit the Semantic Matching Models have on top of hand-crafted text matching and activity-based features. The main results are provided in Table 6.6.

First, we find the improvement from Semantic Matching Models over *TM* + *ACT* becomes marginal, except for our CONCAT model. CONCAT outperforms (p-value < 0.01) all the other baseline semantic matching models (W2V, DSSM), and is the only model that provides a statistically significant improvement over *TM* + *ACT* on both MRR and NACP, achieving nearly +1% improvement on NACP.

Our further analysis in Table 6.7 shows CONCAT is especially helpful for the hard queries on which the activity-based features have less discriminative power. Specifically, we measure their discriminative power using the range of the *Last Access* feature defined as follows,

$$range(\textit{Last Access}) = \max\{\textit{Last Access}\} - \min\{\textit{Last Access}\}\,.$$

A lower range indicates the documents have similar activity-based features and thus the features are less discriminative for ranking the documents. Table 6.7 reports the improvement of CONCAT over *TM* + *ACT* for queries within different *Last Access* range. We find CONCAT improves over the strong *TM* + *ACT* baseline by +1.16% and +2.22% on MRR and NACP respectively, impacting more than 20% of the test queries.

Lastly, we find DSSM is not helpful when combined with *TM* and *ACT* in Table 6.6. We suspect this could be caused by the limited amount of search training data, thus we collect more search data (up to 23 million queries) to train DSSM, and compare CONCAT, which is trained with activity data, against DSSM trained with varying

Table 6.7: Relative performance improvements of CONCAT over *TM*+*ACT* for queries of different $range$(*Last Access*). % of queries indicates what percentage of the queries have a range within the specified threshold.

$range$ (in days)	% of queries	MRR	NACP
2	10.7%	+1.55%$^{\triangle}$	+2.93%$^{\triangle}$
8	20.7%	+1.16%$^{\triangle}$	+2.22%$^{\triangle}$
18	30.1%	+0.97%$^{\triangle}$	+1.89%$^{\triangle}$
53	50.1%	+0.67%$^{\triangle}$	+1.33%$^{\triangle}$
198	90.1%	+0.48%$^{\triangle}$	+1.94%$^{\triangle}$
$+\infty$	100.0%	+0.46%$^{\triangle}$	+0.93%$^{\triangle}$

Table 6.8: Relative performance of DSSM trained with different size of search training data with respect to CONCAT. #Queries reports the number of queries used for training the semantic matching models.

Model	#Queries	MRR	NACP
$TM + ACT +$ CONCAT$_{both}$	0	+0.00%	+0.00%
$TM + ACT +$ DSSM$_{both}$	31K	-1.84%$^{\triangledown}$	-3.15%$^{\triangledown}$
$TM + ACT +$ DSSM$_{both}$	63K	-1.35%$^{\triangledown}$	-2.13%$^{\triangledown}$
$TM + ACT +$ DSSM$_{both}$	1M	-0.03%	-0.13%
$TM + ACT +$ DSSM$_{both}$	23M	+0.28%	+0.54%

amounts of search training data in Table 6.8. Before analyzing the results, we emphasize that our work aims to leverage co-access labels extracted from activity logs to improve ranking *when the search training data is limited*. This is often the case for cloud storage systems, in which users *access* documents much more frequently than they *search* for documents. That said, it plausible that with abundant search clicks, the supervised model DSSM could be more effective. Thus, we further collect more search data in order to investigate whether our *weakly supervised* semantic matching models trained only on co-access labels can provide similar performance as DSSM trained on abundant search clicks.

In Table 6.8, we find that CONCAT still significantly outperforms DSSM, when the search training data contains less than 1 million queries. When training with 1 million to 23 million queries, DSSM becomes comparable or slightly better than CONCAT, however, the differences are quite small and not statistically significant (p-value > 0.01). These results are exciting, as they indicate that even in cases with sparse and limited search logs, activity-based semantic matching models could provide an adequate substitute.

6.6.4 Discussion

It is clear from Table 6.5 and Table 6.6, that our semantic matching models trained on document co-access labels can effectively improve ranking performance, and they

significantly outperform W2V (text representations learned from unlabeled text corpora) and DSSM (text representations learned from clickthrough data). Moreover, Table 6.8 suggests that CONCAT can provide comparable performance to DSSM even when DSSM is trained with orders of magnitude more search queries.

The concatenation structure of CONCAT allows complex feature interaction between the two text inputs. Because of this, CONCAT significantly outperforms the Siamese network model SIAM in co-access prediction. CONCAT is also better than SIAM at capturing effective signals from the co-access labels that are not fully covered by the lexical and activity-based features (Table 6.6). However, we find CONCAT is less effective than SIAM at capturing textual similarity according to the correlation analysis (Table 6.4). We believe this is because the Siamese structure forces SIAM to learn representations for text matching, and therefore it's more effective when transferred from the co-access prediction task to the search ranking task. Thus, when the lexical text matching features are absent, SIAM significantly outperforms CONCAT (Table 6.3).

At the time of writing, the models described in this chapter have not yet been fully deployed. Nevertheless, our offline experiments using large scale Google Drive search and activity logs already provide strong evidence that activity logs are an adequate substitute for search logs when training semantic matching models. This is especially important for cloud storage systems or other domain-specific applications, where search logs may not be available in large quantity or even at all.

For example, when search is first introduced as a feature to a cloud storage system, search logs will be nonexistent, but activity logs are abundant. As another example, a small enterprise may not have enough search traffic to build a specialized semantic matching model, however it may have enough activity data to do so.

6.7 Conclusion

In this chapter, we demonstrate that leveraging user activity, e.g., document access, editing and sharing, can significantly improve the quality of cloud storage search. In cloud storage search, users often eschew using search altogether, opting out for navigation instead. This makes it challenging to leverage click data as it may not be available in the quantity necessary to train ranking models.

Compared to *search logs* in cloud storage system, user *activity logs* are always available in abundant quantities, as they capture any user interaction with the stored documents, however they are not directly tied to search intents and information needs To this end, we introduce a novel method for automatically learning text embeddings from activity logs, by employing *document co-access* as a label for document similarity. Our experiments demonstrate that such embeddings can significantly outperform standard semantic matching approaches, and can be combined with other features to further improve performance when search data is limited. Thus, we answer RQ5 positively: activity-based user interactions can improve ranking quality.

To the best of our knowledge, we are the first to examine the use of activity logs in a large scale cloud storage search engine. As more document collections are moving to the cloud, the benefits of activity logs for improving cloud storage search is an important finding that opens up multiple directions for future work.

First, in this work we do not use more complex text embedding models such as recurrent neural networks or transformers, due to privacy constraints that are in place to protect the users. In future work, we would like to explore ways of using such models while retaining user privacy. Second, in future work we would also like to consider more expressive models of user activity, such as recurrent neural networks to model the historical activity of a user in order to predict future document usage.

In this chapter we considered *activity logs* as opposed to traditional search logs for improving a LTR system. In the next chapter, we will reflect on the main findings of each research question posed throughout this dissertation and propose directions for future work.

7 Conclusions

This dissertation is about efficient, safe and adaptive learning from user interactions. In the five research chapters that preceed this conclusion, we have seen: (1) a comparison between online and counterfactual learning from user interactions, (2) accelerating the counterfactual learning process, (3) performing safe interventions with counterfactual learning, (4) dealing with non-stationarity, and (5) going beyond clicks to other user interaction signals. In this chapter we reflect on the main findings on each of these topics.

7.1 Main Findings

In this section we revisit the research questions that were posed in Chapter 1 and summarize the most important findings.

RQ1 How should LTR practicioners choose which method to apply from either counterfactual or online LTR methodologies?

The answer to this question depends on the particular circumstances of the LTR scenario that a practicioner is confronted with. We identify three factors that are critical in the performance of online and counterfactual LTR: (1) the severity of position bias, (2) whether selection bias is present (i.e. top-10 rankings), and (3) the amount of interaction noise. Counterfactual learning does not perform on the level of online learning when either position bias is severe or when selection bias is present. We address these two problems in RQ2 and RQ3 respectively.

RQ2 Can counterfactual learning from user interactions be made more efficient?

We find that when inverse propensity scores are more extreme, the convergence of IPS-weighted SGD as a learning mechanism for counterfactual problems is slow. Specifically we find that IPS-weighted SGD has a convergence rate that scales with the *maximum* IPS weight in the dataset. When position bias is severe, this can be problematic. By leveraging a sample-based learning algorithm called CounterSample the convergence rate can be massively improved. Our results show that the convergence rate of CounterSample no longer scales with the *maximum* IPS weight, but instead with the *average* IPS weight. This, in turn, leads to significantly faster learning for a number of biased LTR scenarios.

RQ3 Can counterfactual approaches perform interventions without harming the user experience?

We find that counterfactual learning algorithms are able to perform interventions without harming the user experience by employing high-confidence off-policy estimators and we introduce the Safe Exploration Algorithm (SEA). SEA provides formal guarantees of safety, and our empirical results confirm this behavior: SEA is always at least as good as a baseline system. However, our empirical findings suggests that safety is not free and that the conservative nature of SEA makes it unable to explore in situations that could benefit from more aggresive exploration strategies.

RQ4 How can counterfactual approaches be adapted to deal with non-stationary environments?

Our findings are that counterfactual methods can deal with non-stationary environments by employing sliding-window and exponential-decay estimators. We show that standard IPS estimators introduce bias when subjected to a non-stationary environments and this bias can be reduced by either using sliding-window or exponential-decay style estimators. Reducing bias in this manner is, however, not free and comes at the cost of increased variance, leading to the traditional bias-variance tradeoff.

RQ5 Can activity-based user interaction signals improve ranking quality?

Finally, we look beyond clicks and consider other user interaction signals to improve ranking. Our findings here show that in situations where clicks are not abundantly available, leveraging other user interaction signals can provide an adequate substitution when it comes to training semantic matching models. We validate these findings on real-world user interaction data for the task of cloud storage search.

7.2 Future Work

This thesis provides several solutions to make learning from user interactions more efficient, safe and adaptive. Nevertheless, our research is not without limitations and we find ourselves asking new questions based on the findings presented in this thesis.

First, the empirical comparison performed in Chapter 2 is performed on a single dataset and uses a single metric. Although the dataset and metric are widely used in both Counterfactual and Online LTR, it remains to be seen whether our findings generalize to other datasets and metrics. As we have seen in Chapter 4, different LTR datasets have different characteristics, for example in terms of document sparsity. Furthermore our results in Chapter 4 indicate that, depending on the characteristics of the LTR dataset, the behavior of the learning algorithm can differ. Future work should compare online and counterfactual LTR approaches on a broader spectrum of datasets and metrics.

Second, in Chapter 3 we address the inefficiency of IPS-weighted SGD by introducing a sample-based learning algorithm. Our convergence rate analysis assumes that the *propensities* are known a priori. In practice these propensities are usually not

known and are instead estimated using randomization experiments. Understanding the robustness against misspecified and noisy propensities remains an open challenge. Furthermore, this chapter does not consider propensity clipping, a common heuristic to deal with large variance in IPS weights. Propensity clipping directly affects the maximum IPS weight in our dataset. A comparison against propensity clipping should be considered as future work.

Next, Chapter 4 trades off exploration and safety by having the learning algorithm perform interventions. The empirical results suggest that the confidence bounds used for SEA are quite loose and we find that the learned models could have been deployed much sooner. As such, designing estimators with tighter confidence bounds are an important line of future work. Specifically, when we compare SEA to its boundless variant BSEA, we find that BSEA is empirically just as safe but consistently performs better. Furthermore, it may be possible to extend SEA to choose when to deploy on a per-query basis. This would allow SEA to remain conservative on queries that are difficult while more freely exploring on queries that are easier. This line of work requires new estimators that can estimate ranking performance per query.

Then, in Chapter 5 we expand to a non-stationary environment for recommendation in off-policy evaluation. Our work is focused on evaluation however extending our work to non-stationary counterfactual *learning* is challenging. Recent work has proposed an off-policy optimization method for piecewise-stationary environments [61], however their approach is not directly applicable to the smooth non-stationary environments that we consider in Chapter 5. Finally, the field of time-series analysis can provide useful insights for counterfactual evaluation and learning in non-stationary settings. Recent work has performed some initial work for counterfactual evaluation with time-series approaches [165]. Applying time-series approaches to counterfactual learning should be considered as future work.

Finally, in Chapter 6 we find that activity data can be an adequate substitute for click data in situations where click data is not abundantly available. This work introduces a conceptually simple heuristic *co-access* which acts as a weak supervision signal for document similarity. This signal still heavily depends on clicks and is tightly coupled to the user interface of cloud storage search. In future work we should consider more diverse interaction signals beyond clicks. The recent introduction of voice assistants on mobile phones or as dedicated devices is changing how users interact with information retrieval systems. In dialog systems it is impossible to collect clicks and we need to consider different interaction signals to learn from and to evaluate with. How to apply counterfactual learning or evaluation to other types of interaction data such as dialogues remains an open problem.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI*, pages 265–283, 2016. (Cited on page 40.)

[2] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *TOIS*, 23(1):103–145, 2005. (Cited on page 51.)

[3] A. Agarwal, M. J. Wainwright, P. L. Bartlett, and P. K. Ravikumar. Information-theoretic lower bounds on the oracle complexity of convex optimization. In *NeurIPS*, pages 1–9, 2009. (Cited on page 37.)

[4] A. Agarwal, I. Zaitsev, and T. Joachims. Counterfactual learning-to-rank for additive metrics and deep models. *arXiv preprint arXiv:1805.00065*, 2018. (Cited on pages 11, 12, 16, 17, and 19.)

[5] A. Agarwal, K. Takatsu, I. Zaitsev, and T. Joachims. A general framework for counterfactual learning-to-rank. In *SIGIR*, pages 5–14. ACM, 2019. (Cited on pages 32, 33, 34, and 43.)

[6] A. Agarwal, I. Zaitsev, X. Wang, C. Li, M. Najork, and T. Joachims. Estimating position bias without intrusive interventions. In *WSDM*, pages 474–482. ACM, 2019. (Cited on pages 29, 30, 31, and 33.)

[7] S. Agrawal and N. Goyal. Thompson sampling for contextual bandits with linear payoffs. In *ICML*, pages 127–135, 2013. (Cited on pages 51, 52, 54, 56, and 65.)

[8] Q. Ai, S. T. Dumais, N. Craswell, and D. Liebling. Characterizing email search using large-scale behavioral logs and surveys. In *WWW*, pages 1511–1520. ACM, 2017. (Cited on page 99.)

[9] Q. Ai, K. Bi, C. Luo, J. Guo, and W. B. Croft. Unbiased learning to rank with unbiased propensity estimation. In *SIGIR*, pages 385–394. ACM, 2018. (Cited on pages 11, 13, 18, 19, 27, and 34.)

[10] Q. Ai, J. Mao, Y. Liu, and W. B. Croft. Unbiased learning to rank: Theory and practice. In *CIKM*, pages 2305–2306. ACM, 2018. (Cited on pages 26, 29, 30, and 33.)

[11] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015. (Cited on page 31.)

[12] T. Alrashed, A. H. Awadallah, and S. Dumais. The lifetime of email messages: A large-scale analysis of email revisitation. In *CHIIR*, pages 120–129. ACM, 2018. (Cited on page 99.)

[13] M. U. Anwaar, D. Rybalko, and M. Kleinsteuber. Counterfactual learning from logs for improved ranking of e-commerce products. *arXiv preprint arXiv:1907.10409*, 2019. (Cited on page 29.)

[14] E. Aronson. *The Social Animal*. Worth/Freeman, 10th edition, 2008. (Cited on page 79.)

[15] H. Bang and J. M. Robins. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973, 2005. (Cited on pages 77 and 78.)

[16] M. Bendersky, X. Wang, D. Metzler, and M. Najork. Learning from user interactions in personal search via attribute parameterization. In *WSDM*, pages 791–799. ACM, 2017. (Cited on pages 1, 9, 13, 34, and 98.)

[17] O. Besbes, Y. Gur, and A. Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. In *NeurIPS*, volume 1, pages 199–207, 2014. (Cited on page 79.)

[18] A. Beygelzimer and J. Langford. The offset tree for learning with partial labels. In *KDD*, pages 129–138. ACM, 2009. (Cited on page 63.)

[19] S. Bruch, M. Zoghi, M. Bendersky, and M. Najork. Revisiting approximate metric optimization in the age of deep neural networks. In *SIGIR*, pages 1241–1244. ACM, 2019. (Cited on page 99.)

[20] C. Burges, K. Svore, P. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. In *Proceedings of the Learning to Rank Challenge*, pages 25–35, 2011. (Cited on page 1.)

[21] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010. (Cited on pages 1 and 98.)

[22] I. Cantador, P. Brusilovsky, and T. Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *RecSys*, New York, NY, USA, 2011. ACM. (Cited on pages 79 and 88.)

[23] D. Carmel, G. Halawi, L. Lewin-Eytan, Y. Maarek, and A. Raviv. Rank by time or by relevance?: Revisiting email search. In *CIKM*, pages 283–292. ACM, 2015. (Cited on page 98.)

[24] B. Carterette and P. Chandar. Offline comparative evaluation with incremental, minimally-invasive online feedback. In *SIGIR*, pages 705–714. ACM, 2018. (Cited on page 33.)

[25] N. Cesa-Bianchi, C. Gentile, and G. Zappella. A gang of bandits. In *NeurIPS*, pages 737–745, 2013. (Cited on page 88.)

[26] N. Cesa-Bianchi, C. Gentile, G. Neu, and G. Lugosi. Boltzmann exploration done right. In *NeurIPS*,

pages 6275–6284, 2017. (Cited on page 54.)
[27] P. Chandar and B. Carterette. Estimating clickthrough bias in the cascade model. In *CIKM*, pages 1587–1590. ACM, 2018. (Cited on page 33.)
[28] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*, pages 1–24, 2011. (Cited on pages 1, 9, 18, 41, 64, and 99.)
[29] A. Chuklin, P. Serdyukov, and M. de Rijke. Modeling clicks beyond the first result page. In *CIKM*, pages 1217–1220. ACM, 2013. (Cited on page 64.)
[30] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool Publishers, August 2015. (Cited on pages 26 and 55.)
[31] D. Cohen, S. M. Jordan, and W. B. Croft. Learning a better negative sampling policy with deep neural networks for search. In *ICTIR*, pages 19–26. ACM, 2019. (Cited on page 99.)
[32] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94. ACM, 2008. (Cited on pages 2, 29, 32, and 33.)
[33] Z. Dai, C. Xiong, J. Callan, and Z. Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*, pages 126–134. ACM, 2018. (Cited on page 99.)
[34] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. Neural ranking models with weak supervision. In *SIGIR*, pages 65–74. ACM, 2017. (Cited on pages 1 and 99.)
[35] Delicious. Delicious website. `http://www.delicious.com`, 2018. Accessed: 2018-08-09. (Cited on pages 79 and 88.)
[36] D. Di Castro, Z. Karnin, L. Lewin-Eytan, and Y. Maarek. You've got mail, and here is what you could do with it!: Analyzing and predicting actions on email messages. In *WSDM*, pages 307–316. ACM, 2016. (Cited on page 99.)
[37] F. Diaz. Integration of news content into web results. In *WSDM*, pages 182–191. ACM, 2009. (Cited on page 79.)
[38] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside dropbox: Understanding personal cloud storage services. In *IMC*, pages 481–494. ACM, 2012. (Cited on page 98.)
[39] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011. (Cited on page 44.)
[40] M. Dudík, J. Langford, and L. Li. Doubly robust policy evaluation and learning. In *ICML*, pages 1097–1104, 2011. (Cited on pages 77, 78, 84, 94, and 95.)
[41] M. Dudík, D. Erhan, J. Langford, and L. Li. Sample-efficient nonstationary policy evaluation for contextual bandits. In *UAI*, page 247–254, Arlington, Virginia, USA, 2012. AUAI Press. ISBN 9780974903989. (Cited on pages 79 and 90.)
[42] S. Dumais, E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *SIGIR*, pages 72–79. ACM, 2003. (Cited on pages 97, 98, and 99.)
[43] Z. Fang, A. Agarwal, and T. Joachims. Intervention harvesting for context-dependent examination-bias estimation. In *SIGIR*, pages 825–834. ACM, 2019. (Cited on pages 31 and 33.)
[44] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001. (Cited on page 101.)
[45] M. Fuller, L. Kelly, and G. J. Jones. Applying contextual memory cues for retrieval from personal information archives. In *CHI-PIM Workshop*, 2008. (Cited on pages 97, 98, and 99.)
[46] N. Galichet, M. Sebag, and O. Teytaud. Exploration vs exploitation vs safety: Risk-aware multi-armed bandits. In *ACML*, pages 245–260. PMLR, 2013. (Cited on page 55.)
[47] J. Garcia and F. Fernández. Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564, 2012. (Cited on page 55.)
[48] A. Garivier and E. Moulines. On upper-confidence bound policies for switching bandit problems. *ALT*, pages 174–188, 2011. (Cited on page 79.)
[49] D. Glowacka. Bandit algorithms in information retrieval. *Foundations and Trends in Information Retrieval*, 13(4):299–424, 2019. (Cited on page 51.)
[50] A. Grotov and M. de Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *SIGIR*, pages 1215–1218. ACM, 2016. (Cited on pages 26 and 29.)
[51] A. Grotov, S. Whiteson, and M. de Rijke. Bayesian ranker comparison based on historical user interactions. In *SIGIR*, pages 273–282. ACM, 2015. (Cited on page 52.)
[52] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. A deep relevance matching model for ad-hoc retrieval. In *CIKM*, pages 55–64. ACM, 2016. (Cited on pages 97, 99, and 101.)
[53] M. A. Hasan, N. Parikh, G. Singh, and N. Sundaresan. Query suggestion for e-commerce sites. In *WSDM*, pages 765–774. ACM, 2011. (Cited on page 29.)

[54] E. Hazan and S. Kale. Beyond the regret minimization barrier: Optimal algorithms for stochastic strongly-convex optimization. *JMLR*, 15(1):2489–2512, 2014. (Cited on pages 31 and 37.)
[55] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007. (Cited on pages 31 and 37.)
[56] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR*, pages 251–263. Springer, 2011. (Cited on pages 51, 53, and 64.)
[57] K. Hofmann, S. Whiteson, and M. de Rijke. Contextual bandits for information retrieval. In *NeurIPS-Workshop on Bayesian Optimization, Experimental Design, and Bandits*, December 2011. (Cited on pages 51, 53, 54, and 73.)
[58] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM*, pages 249–258. ACM, 2011. (Cited on page 14.)
[59] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster online learning to rank for IR. In *WSDM*, pages 183–192. ACM, 2013. (Cited on pages 9, 13, 15, 17, 18, 19, 52, 54, and 77.)
[60] K. Hofmann, S. Whiteson, and M. de Rijke. Fidelity, soundness, and efficiency of interleaved comparison methods. *TOIS*, 31(4):17, 2013. (Cited on page 64.)
[61] J. Hong, B. Kveton, M. Zaheer, Y. Chow, and A. Ahmed. Piecewise-stationary off-policy optimization. *arXiv preprint arXiv:2006.08236*, 2020. (Cited on page 117.)
[62] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952. (Cited on pages 78 and 81.)
[63] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338. ACM, 2013. (Cited on pages 99, 101, and 106.)
[64] J. J. Hull. A database for handwritten text recognition research. *Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994. (Cited on page 63.)
[65] X. Huo and F. Fu. Risk-aware multi-armed bandit problem with application to portfolio selection. *arXiv preprint arXiv:1709.04415*, 2017. (Cited on page 55.)
[66] R. Jagerman and M. de Rijke. Accelerated convergence for counterfactual learning to rank. In *SIGIR*, page 469–478. ACM, 2020. (Cited on page 29.)
[67] R. Jagerman, K. Balog, P. Schaer, J. Schaible, N. Tavakolpoursaleh, and M. de Rijke. Overview of TREC OpenSearch 2017. In *TREC*, 2017.
[68] R. Jagerman, C. Eickhoff, and M. de Rijke. Computing web-scale topic models using an asynchronous parameter server. In *SIGIR*, pages 1337–1340. ACM, 2017.
[69] R. Jagerman, J. Kiseleva, and M. de Rijke. Modeling label ambiguity for neural list-wise learning to rank. In *SIGIR-NeuIR Workshop*. ACM, 2017.
[70] R. Jagerman, K. Balog, and M. de Rijke. OpenSearch: Lessons learned from an online evaluation campaign. *JDIQ*, 10(3):Article 13, 2018. (Cited on page 77.)
[71] R. Jagerman, I. Markov, and M. de Rijke. When people change their mind: Off-policy evaluation in non-stationary recommendation environments. In *WSDM*, pages 447–455. ACM, 2019. (Cited on page 77.)
[72] R. Jagerman, H. Oosterhuis, and M. de Rijke. To model or to intervene: A comparison of counterfactual and online learning to rank from user interactions. In *SIGIR*, pages 15–24. ACM, 2019. (Cited on pages 2, 9, 33, 34, 54, 55, and 69.)
[73] R. Jagerman, W. Kong, R. K. Pasumarthi, Z. Qin, M. Bendersky, and M. Najork. Improving cloud storage search with activity data. In *Under Review*, 2020. (Cited on page 97.)
[74] R. Jagerman, I. Markov, and M. de Rijke. Safe exploration for optimizing contextual bandits. *TOIS*, 38(3):Article 24, April 2020. (Cited on page 51.)
[75] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20(4):422–446, Oct. 2002. ISSN 1046-8188. (Cited on pages 12, 19, 43, and 66.)
[76] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, pages 133–142. ACM, 2002. (Cited on pages 15, 32, 65, 77, 97, and 98.)
[77] T. Joachims. Evaluating retrieval performance using clickthrough data. In *Text Mining*. Physica/Springer, 2003. (Cited on page 14.)
[78] T. Joachims and A. Swaminathan. Counterfactual evaluation and learning for search, recommendation and ad placement. In *SIGIR*, pages 1199–1201, New York, NY, USA, 2016. ACM. (Cited on page 26.)
[79] T. Joachims, L. A. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough

data as implicit feedback. In *SIGIR*, pages 154–161. ACM, 2005. (Cited on pages 11, 29, and 64.)

[80] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *TOIS*, 25(2):7, 2007. (Cited on pages 1 and 9.)

[81] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased learning-to-rank with biased feedback. In *WSDM*, pages 781–789, New York, NY, USA, 2017. ACM. (Cited on pages 2 and 51.)

[82] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased learning-to-rank with biased feedback. In *WSDM*, pages 781–789, New York, NY, USA, 2017. ACM. ISBN 9781450346757. (Cited on pages 9, 11, 12, 16, 18, 19, 21, 22, 26, 29, 30, 31, 32, 33, 41, 42, 43, 63, and 64.)

[83] T. Joachims, A. Swaminathan, and M. de Rijke. Deep learning with logged bandit feedback. In *ICLR*, 2018. (Cited on pages 29, 30, 33, 56, 57, and 65.)

[84] L. P. Kaelbling. Associative reinforcement learning: Functions in k-dnf. *Machine Learning*, 15(3): 279–298, 1994. (Cited on pages 51 and 56.)

[85] D. Kahneman, P. Slovic, and A. Tversky, editors. *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge University Press, 1982. (Cited on page 79.)

[86] S. M. Kakade, S. Shalev-Shwartz, and A. Tewari. Efficient bandit algorithms for online multiclass prediction. In *ICML*, pages 440–447. ACM, 2008. (Cited on page 56.)

[87] N. Karampatziakis and J. Langford. Online importance weight aware updates. In *UAI*, page 392–399. AUAI Press, 2011. (Cited on page 31.)

[88] A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018. (Cited on page 31.)

[89] A. Kazerouni, M. Ghavamzadeh, Y. A. Yadkori, and B. Van Roy. Conservative contextual linear bandits. In *NeurIPS*, pages 3910–3919, 2017. (Cited on pages 55, 58, 61, and 65.)

[90] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*, pages 3149–3157. Curran Associates Inc., 2017. (Cited on page 99.)

[91] J. Kim and W. B. Croft. Ranking using multiple document types in desktop search. In *SIGIR*, pages 50–57. ACM, 2010. doi: 10.1145/1835449.1835461. (Cited on page 98.)

[92] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 44.)

[93] A. Kulkarni, J. Teevan, K. M. Svore, and S. T. Dumais. Understanding temporal query dynamics. In *WSDM*, pages 167–176. ACM, 2011. (Cited on page 77.)

[94] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, volume 10, pages 331–339, 1995. (Cited on page 63.)

[95] J. Langford, A. Strehl, and J. Wortman. Exploration scavenging. In *ICML*, pages 528–535. ACM, 2008. (Cited on pages 51 and 53.)

[96] Last.fm. Last.fm website. `http://www.lastfm.com`, 2018. Accessed: 2018-08-09. (Cited on pages 79 and 88.)

[97] D. Lefortier, P. Serdyukov, and M. De Rijke. Online exploration for detecting shifts in fresh intent. In *CIKM*, pages 589–598. ACM, 2014. (Cited on page 79.)

[98] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5(Apr):361–397, 2004. (Cited on page 63.)

[99] C. Li, B. Kveton, T. Lattimore, I. Markov, M. de Rijke, C. Szepesvari, and M. Zoghi. Bubblerank: Safe online learning to re-rank via implicit click feedback. In *UAI*, 2019. (Cited on pages 11, 55, and 75.)

[100] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670. ACM, 2010. (Cited on pages 51, 52, 53, 54, 56, 65, 77, and 89.)

[101] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM*, pages 297–306, New York, NY, USA, 2011. ACM. ISBN 9781450304931. (Cited on pages 25, 27, 53, 57, 77, 78, and 80.)

[102] L. Li, S. Chen, J. Kleban, and A. Gupta. Counterfactual estimation and optimization of click metrics in search engines: A case study. In *WWW*, pages 929–934. ACM, 2015. (Cited on page 106.)

[103] F. Liu, J. Lee, and N. Shroff. A change-detection based framework for piecewise-stationary multi-armed bandit problem. In *AAAI*, pages 3651–3658. AAAI Press, 2018. (Cited on page 79.)

[104] J. Liu, P. Dolan, and E. R. Pedersen. Personalized news recommendation based on click behavior. In *IUI*, pages 31–40. ACM, 2010. (Cited on page 77.)

[105] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Re-*

trieval, 3(3):225–331, 2009. (Cited on pages 9, 26, 29, and 98.)
[106] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. Post-learning optimization of tree ensembles for efficient ranking. In *SIGIR*, pages 949–952. ACM, 2016. (Cited on pages 41 and 64.)
[107] C. Lucchese, F. M. Nardini, R. K. Pasumarthi, S. Bruch, M. Bendersky, X. Wang, H. Oosterhuis, R. Jagerman, and M. de Rijke. Learning to rank in theory and practice: From gradient boosting to neural networks and unbiased learning. In *SIGIR*, pages 1419–1420. ACM, 2019.
[108] T. S. McElroy, B. C. Monsell, and R. J. Hutchinson. Modeling of holiday effects and seasonality in daily time series. Technical report, Center for Statistical Research and Methodology, 2018. (Cited on page 79.)
[109] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119. 2013. (Cited on page 106.)
[110] B. Mitra and N. Craswell. An introduction to neural information retrieval. *Foundations and Trends in Information Retrieval*, 13(1):1–126, 2018. (Cited on page 99.)
[111] J. L. Moore, S. Chen, D. Turnbull, and T. Joachims. Taste over time: The temporal dynamics of user preferences. In *ISMIR*, pages 401–406, 2013. (Cited on pages 77, 79, 81, and 88.)
[112] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814. Omnipress, 2010. (Cited on page 102.)
[113] K. Narang, S. T. Dumais, N. Craswell, D. Liebling, and Q. Ai. Large-scale analysis of email search and organizational strategies. In *CHIIR*, pages 215–223. ACM, 2017. (Cited on page 99.)
[114] O. Nasraoui, J. Cerwinske, C. Rojas, and F. Gonzalez. Performance of recommendation systems in dynamic streaming environments. In *SDM*, pages 569–574. SIAM, 2007. (Cited on pages 77, 81, and 88.)
[115] D. Needell, R. Ward, and N. Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *NeurIPS*, pages 1017–1025, 2014. (Cited on page 31.)
[116] H. Oosterhuis. Learning to rank and evaluation in the online setting. 12th Russian Summer School in Information Retrieval (RuSSIR 2018), 2018. (Cited on page 26.)
[117] H. Oosterhuis and M. de Rijke. Balancing speed and quality in online learning to rank for information retrieval. In *CIKM*, pages 277–286. ACM, 2017. (Cited on pages 14, 15, 52, and 64.)
[118] H. Oosterhuis and M. de Rijke. Sensitive and scalable online evaluation with theoretical guarantees. In *CIKM*, pages 77–86. ACM, 2017. (Cited on page 15.)
[119] H. Oosterhuis and M. de Rijke. Differentiable unbiased online learning to rank. In *CIKM*, pages 1293–1302. ACM, 2018. (Cited on pages 9, 11, 13, 14, 15, 16, 17, 18, and 19.)
[120] H. Oosterhuis and M. de Rijke. Optimizing ranking models in the online setting. In *ECIR*, pages 382–396. Springer, 2019. (Cited on pages 2, 15, and 16.)
[121] H. Oosterhuis and M. de Rijke. Policy-aware unbiased learning to rank for top-k rankings. In *SIGIR*, page 489–498. ACM, 2020. doi: 10.1145/3397271.3401102. (Cited on page 2.)
[122] H. Oosterhuis, A. Schuth, and M. de Rijke. Probabilistic multileave gradient descent. In *ECIR*, pages 661–668. Springer, 2016. ISBN 978-3-319-30671-1. (Cited on pages 15 and 64.)
[123] H. Oosterhuis, R. Jagerman, and M. de Rijke. Unbiased learning to rank: Counterfactual and online approaches. In *WWW*, pages 299–300. ACM, 2020. (Cited on page 2.)
[124] A. B. Owen. *Monte Carlo theory, methods and examples*. 2013. URL `https://statweb.stanford.edu/~owen/mc/`. (Cited on page 62.)
[125] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. (Cited on page 1.)
[126] R. K. Pasumarthi, S. Bruch, X. Wang, C. Li, M. Bendersky, M. Najork, et al. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *KDD*, pages 2970–2978, 2019. (Cited on page 106.)
[127] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NeurIPS*, 2017. (Cited on page 40.)
[128] F. S. F. Pereira, J. Gama, S. de Amo, and G. M. B. Oliveira. On analyzing user preference dynamics with temporal social networks. *Machine Learning*, 107(11):1745–1773, 2018. (Cited on page 79.)
[129] D. Precup, R. S. Sutton, and S. P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, pages 759–766, 2000. (Cited on pages 77 and 78.)
[130] T. Qin and T. Liu. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013. (Cited on page 64.)
[131] T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010. ISSN 1386-4564. doi: 10.1007/s10791-009-9123-y. (Cited on pages 1 and 9.)

[132] M. Quadrana, M. Reznakova, T. Ye, E. Schmidt, and H. Vahabi. Modeling musical taste evolution with recurrent neural networks. *arXiv preprint arXiv:1806.06535*, 2018. (Cited on pages 79 and 80.)
[133] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz. Modeling and predicting behavioral dynamics on the web. In *WWW*, pages 599–608. ACM, 2012. (Cited on page 77.)
[134] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM*, pages 245–254. ACM, 2013. (Cited on page 14.)
[135] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *AAAI*, pages 1406–1412. AAAI Press, 2006. (Cited on page 97.)
[136] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM*, pages 43–52, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. (Cited on page 65.)
[137] A. Rakhlin, O. Shamir, and K. Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. In *ICML*, 2011. (Cited on pages 31 and 37.)
[138] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *ICLR*, 2019. (Cited on page 45.)
[139] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951. (Cited on page 31.)
[140] S. Robertson and H. Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009. (Cited on page 1.)
[141] S. E. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33(4):294–304, 1977. (Cited on page 107.)
[142] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 1987. (Cited on page 35.)
[143] T. Schnabel, P. N. Bennett, S. T. Dumais, and T. Joachims. Short-term satisfaction and long-term coverage: Understanding how users tolerate algorithmic exploration. In *WSDM*, pages 513–521. ACM, 2018. (Cited on page 77.)
[144] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke. Multileaved comparisons for fast online evaluation. In *CIKM*, pages 71–80. ACM, 2014. (Cited on pages 13 and 15.)
[145] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke. Multileave gradient descent for fast online learning to rank. In *WSDM*, pages 457–466. ACM, 2016. (Cited on pages 15, 17, 18, and 19.)
[146] N. Schwarz and F. Strack. Context effects in attitude surveys: Applying cognitive theory to social research. *European Review of Social Psychology*, 2:31–50, 1991. (Cited on page 79.)
[147] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1): 1–114, 2012. (Cited on pages 14 and 17.)
[148] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. (Cited on pages 31, 34, 35, and 36.)
[149] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *ICML*, pages 71–79, 2013. (Cited on pages 31 and 37.)
[150] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW*, pages 373–374. ACM, 2014. (Cited on page 99.)
[151] A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from logged implicit exploration data. In *NeurIPS*, pages 2217–2225, 2010. (Cited on pages 51, 52, and 56.)
[152] A. L. Strehl, C. Mesterharm, M. L. Littman, and H. Hirsh. Experience-efficient learning in associative bandit problems. In *ICML*, pages 889–896. ACM, 2006. (Cited on page 56.)
[153] W. Sun, D. Dey, and A. Kapoor. Risk-aware algorithms for adversarial contextual bandits. *arXiv preprint arXiv:1610.05129*, 2016. (Cited on page 55.)
[154] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 1998. (Cited on pages 79 and 89.)
[155] A. Swaminathan and T. Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *JMLR*, 16(1731–1755), 2015. (Cited on pages 30, 51, and 54.)
[156] A. Swaminathan and T. Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *ICML*, pages 814–823. JMLR.org, 2015. (Cited on pages 11, 30, 49, 51, 52, 54, 56, 57, and 62.)
[157] A. Swaminathan and T. Joachims. The self-normalized estimator for counterfactual learning. In *NeurIPS*, pages 3231–3239, 2015. (Cited on page 49.)
[158] A. Swaminathan, A. Krishnamurthy, A. Agarwal, M. Dudik, J. Langford, D. Jose, and I. Zitouni. Off-policy evaluation for slate recommendation. In *NeurIPS*, pages 3632–3642, 2017. (Cited on pages 25, 27, 56, 77, and 78.)
[159] L. Sweeney. K-anonymity: A model for protecting privacy. *International Journal of Uncertainty,*

Fuzziness and Knowledge-Based Systems, 10(5):557–570, Oct. 2002. (Cited on pages 98 and 101.)
[160] S. Tata, A. Popescul, M. Najork, M. Colagrosso, J. Gibbons, A. Green, A. Mah, M. Smith, D. Garg, C. Meyer, and R. Kan. Quick access: Building a smart experience for google drive. In *KDD*, pages 1643–1651. ACM, 2017. (Cited on page 99.)
[161] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51(6):757–772, 2015. (Cited on page 26.)
[162] P. S. Thomas and E. Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, volume 48, pages 2139–2148. JMLR.org, 2016. (Cited on page 78.)
[163] P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence off-policy evaluation. In *AAAI*, pages 3000–3006. AAAI Press, 2015. (Cited on pages 25, 27, 57, 58, 61, and 77.)
[164] P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence policy improvement. In *ICML*, page 2380–2388. JMLR.org, 2015. (Cited on page 55.)
[165] P. S. Thomas, G. Theocharous, M. Ghavamzadeh, I. Durugkar, and E. Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In *AAAI*, pages 4740–4745. AAAI Press, 2017. (Cited on pages 79 and 117.)
[166] R. Tourangeau. Context effects on attitude responses: The role of retrieval and necessary structures. In N. Schwarz and S. Sudman, editors, *Context Effects in Social and Psychological Research*, pages 35–47. Springer, 1992. (Cited on page 79.)
[167] B. Tran, M. Karimzadehgan, R. K. Pasumarthi, M. Bendersky, and D. Metzler. Domain adaptation for enterprise email search. In *SIGIR*, pages 25–34. ACM, 2019. (Cited on page 98.)
[168] C. Van Gysel, M. de Rijke, and E. Kanoulas. Learning latent vector spaces for product search. In *CIKM*, pages 165–174. ACM, 2016. (Cited on page 97.)
[169] D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics with Applications*. Cengage Learning, 2014. (Cited on page 86.)
[170] A. J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974. (Cited on page 40.)
[171] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *TOMS*, 3(3):253–256, 1977. (Cited on page 40.)
[172] X. Wang, M. Bendersky, D. Metzler, and M. Najork. Learning to rank with selection bias in personal search. In *SIGIR*, pages 115–124. ACM, 2016. (Cited on pages 2, 13, 30, 33, 97, 98, and 106.)
[173] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, and M. Najork. Position bias estimation for unbiased learning to rank in personal search. In *WSDM*, pages 610–618. ACM, 2018. (Cited on pages 11, 13, 31, 34, 52, and 97.)
[174] X. Wang, C. Li, N. Golbandi, M. Bendersky, and M. Najork. The lambdaloss framework for ranking metric optimization. In *CIKM*, pages 1313–1322. ACM, 2018. (Cited on pages 1 and 99.)
[175] Y.-X. Wang, A. Agarwal, and M. Dudik. Optimal and adaptive off-policy evaluation in contextual bandits. In *ICML*, pages 3589–3597, 2017. (Cited on pages 77, 78, and 95.)
[176] Z. Wei, J. Xu, Y. Lan, J. Guo, and X. Cheng. Reinforcement learning to rank with markov decision process. In *SIGIR*, pages 945–948. ACM, 2017. (Cited on pages 51 and 56.)
[177] P. Whittle. Restless bandits: Activity allocation in a changing world. *Applied Probability*, 25(A): 287–298, 1988. (Cited on page 79.)
[178] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010. (Cited on pages 75, 98, and 99.)
[179] Q. Wu, N. Iyer, and H. Wang. Learning contextual bandits in a non-stationary environment. In *SIGIR*. ACM, 2018. (Cited on pages 77, 79, 87, 88, 89, and 91.)
[180] Y. Wu, R. Shariff, T. Lattimore, and C. Szepesvári. Conservative bandits. In *ICML*, pages 1254–1262, 2016. (Cited on pages 55 and 58.)
[181] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR*, pages 55–64. ACM, 2017. (Cited on page 99.)
[182] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin. From infrastructure to culture: A/b testing challenges in large scale social networks. In *KDD*, pages 2227–2236. ACM, 2015. (Cited on page 77.)
[183] J. Y. Yu and S. Mannor. Piecewise-stationary bandit problems with side observations. In *ICML*, pages 1177–1184. ACM, 2009. (Cited on page 87.)
[184] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*, pages 1201–1208, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. (Cited on pages 2, 9, 13, 14, and 65.)
[185] H. Zamani, B. Mitra, X. Song, N. Craswell, and S. Tiwary. Neural ranking models with multiple

document fields. In *WSDM*, pages 700–708. ACM, 2018. (Cited on page 99.)

[186] P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML*, pages 1–9, 2015. (Cited on page 31.)

[187] T. Zhao and I. King. Constructing reliable gradient exploration for online learning to rank. In *CIKM*, pages 1643–1652. ACM, 2016. (Cited on page 15.)

Summary

Information Retrieval (IR) systems such as search engines or recommender systems face a complex challenge of ranking a set of candidate items by their relevance with respect to a query, user profile, or context. Learning to Rank (LTR) is commonly used as a way of learning a model that can rank candidate items by their relevance. Historically LTR models have been trained with annotated datasets: data where it is known which items are relevant and which ones are not. However, in recent years the limitations of annotated datasets have become more apparant and interest in using user interactions to train LTR models has increased.

User interactions occur naturally in modern IR systems and can provide implicit feedback for a retrieval system. A big challenge in using user interactions is that they are both *noisy* and *biased*. Counterfactual learning and online learning are the two main directions that enable unbiased learning from implicit feedback. In this dissertation we identify and propose solutions for three major challenges in unbiased learning from user interactions: *efficiency*, *safety*, and *adaptiveness*.

First, counterfactual learning can be *inefficient* due to the high variance introduced by the inverse propensity scores. To address this problem, this dissertation studies the convergence rate of IPS-weighted Stochastic Gradient Descent and proposes a novel learning algorithm with provably better convergence rate.

Second, we find that historical interaction data may be limited and interactions on potentially high quality items may be missing. This is especially problematic when selection bias is present: a phenomenon where users only interact with a limited subset of the ranked result list. To address this problem we need learning algorithms that perform interventions: changing what is shown to the user during data collection. However, interventions may be unsafe: we risk exposing the user to poor rankings. To tackle this challenge we introduce a *safe* counterfactual learning algorithm that can periodically deploy its learned model to change what user interactions are gathered.

Third, we look at the *adaptiveness* of counterfactual learning in situations where user preferences change over time. In non-stationary settings we study the bias and variance of counterfactual estimators and find that standard IPS-based estimators are biased. We propose two new estimators, based on weighting historical interactions, that enjoy reduced bias in the non-stationary setting.

Finally, the thesis looks beyond search-specific interactions to other types of interactions, looking specifically at activity logs in a cloud file storage system. This line of work explores how interaction logs beyond traditional search logs can be used to improve the ranking quality of modern IR systems.

Samenvatting

Information Retrieval (IR) systemen, zoals zoekmachines en aanbevelingssystemen, staan voor een complexe uitdaging: ze proberen een set kandidaatitems te rangschikken op basis van hun relevantie met betrekking tot een zoekopdracht, gebruikersprofiel of context. Learning to Rank (LTR) is een veelgebruikte manier om een model te leren dat kandidaatitems kan rangschikken op basis van hun relevantie. Historisch gezien zijn LTR-modellen getraind met geannoteerde datasets: datasets waarvan bekend is welke kandidaatitems relevant zijn en welke niet. In de afgelopen jaren zijn de beperkingen van geannoteerde datasets aan het licht gekomen en is de belangstelling voor het trainen van LTR-modellen op basis van gebruikersinteracties toegenomen.

Gebruikersinteracties komen van nature veel voor in moderne IR-systemen en kunnen impliciete feedback geven voor een retrieval-systeem. Een grote uitdaging bij het inzetten van gebruikersinteracties is dat ze zowel *noisy* als *biased* zijn. Counterfactual en online leren zijn de twee belangrijkste richtingen die *unbiased* leren mogelijk maken. In dit proefschrift identificeren we en geven we oplossingen voor drie grote uitdagingen in unbiased leren van gebruikersinteracties: efficiëntie, veiligheid en aanpassingsvermogen.

Ten eerste zien we dat counterfactual leren *inefficiënt* kan zijn vanwege de hoge variantie die wordt geïntroduceerd door de inverse propensity scores. Om dit probleem aan te pakken, bestuderen we de convergentiesnelheid van IPS-weighted SGD en stellen we een nieuw leeralgoritme voor dat een betere convergentiesnelheid heeft.

Ten tweede vinden we dat historische interactiegegevens mogelijk beperkt zijn en belangrijke interacties kunnen ontbreken. Dit probleem doet zich voornamelijk voor wanneer selectiebias aanwezig is: een fenomeen waarbij gebruikers alleen op een beperkte subset van de gerangschikte resultatenlijst klikken. Om dit probleem aan te pakken hebben we leeralgoritmen nodig die interventies uitvoeren: algoritmen die veranderen wat de gebruiker te zien krijgt. Interventies kunnen echter onveilig zijn: we lopen het risico de gebruiker bloot te stellen aan slechte resultaatlijsten. Om deze uitdaging aan te gaan, introduceren we een *veilig* counterfactual leeralgoritme dat periodiek het geleerde model kan inzetten om te veranderen welke resultaten er worden getoond en welke gebruikersinteracties er worden verzameld.

Ten derde kijken we naar het *aanpassingsvermogen* van counterfactual leren in situaties waarin gebruikersvoorkeuren in de loop van de tijd veranderen. In niet-stationaire omgevingen bestuderen we zowel de bias als de variantie van counterfactual estimators en vinden we dat standaard IPS-gebaseerde estimators niet langer unbiased zijn. We stellen twee nieuwe estimators voor, gebaseerd op het wegen van historische interacties, die minder biased zijn in de niet-stationaire setting.

Ten slotte kijkt dit proefschrift verder dan zoekspecifieke interacties naar andere soorten interacties, met name naar activiteits-gebaseerde interacties in een cloud opslagsysteem. Specifiek, onderzoeken we hoe activiteits-gebaseerde interacties kunnen worden ingezet om de zoekkwaliteit van moderne IR-systemen verder te verbeteren.