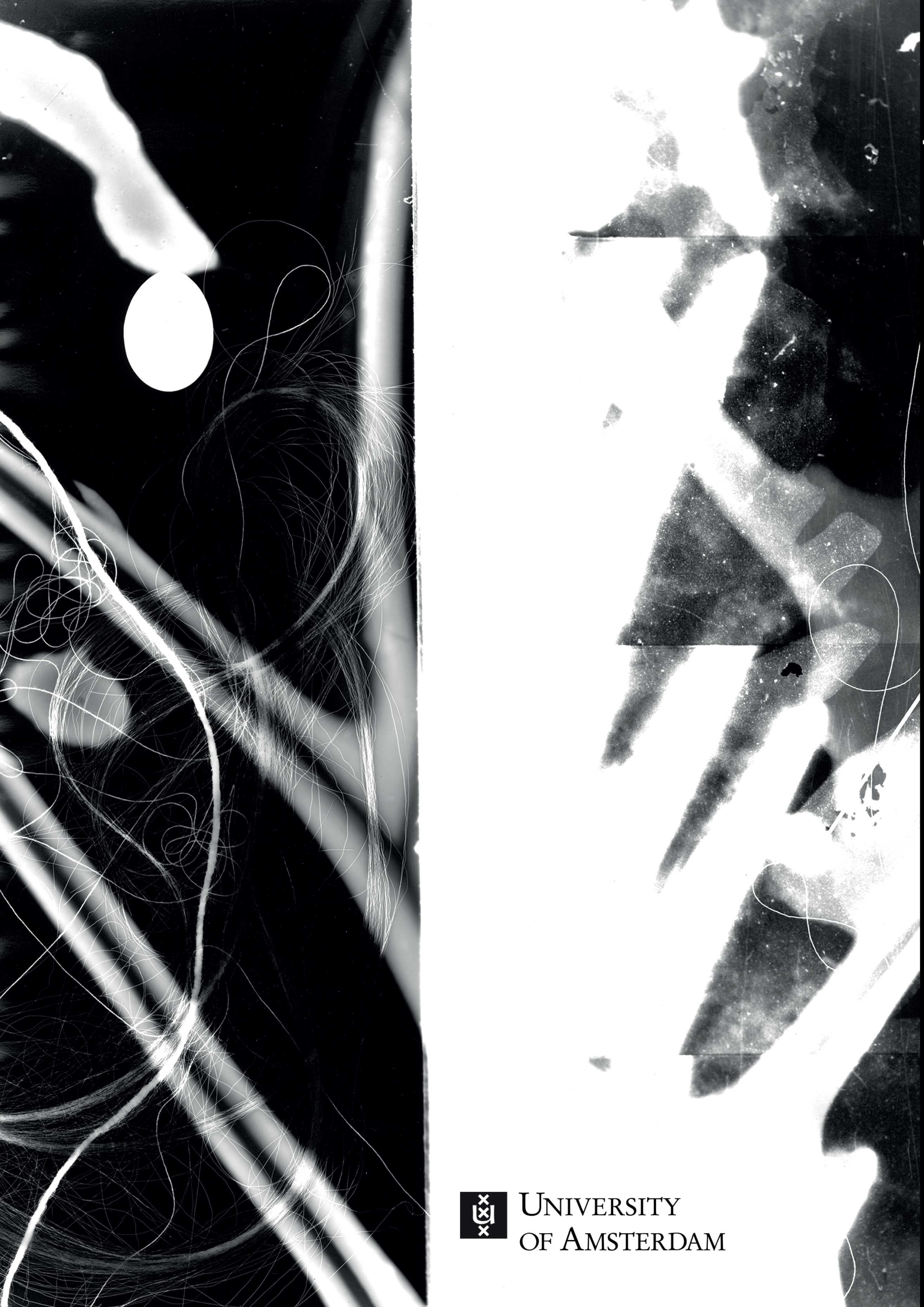
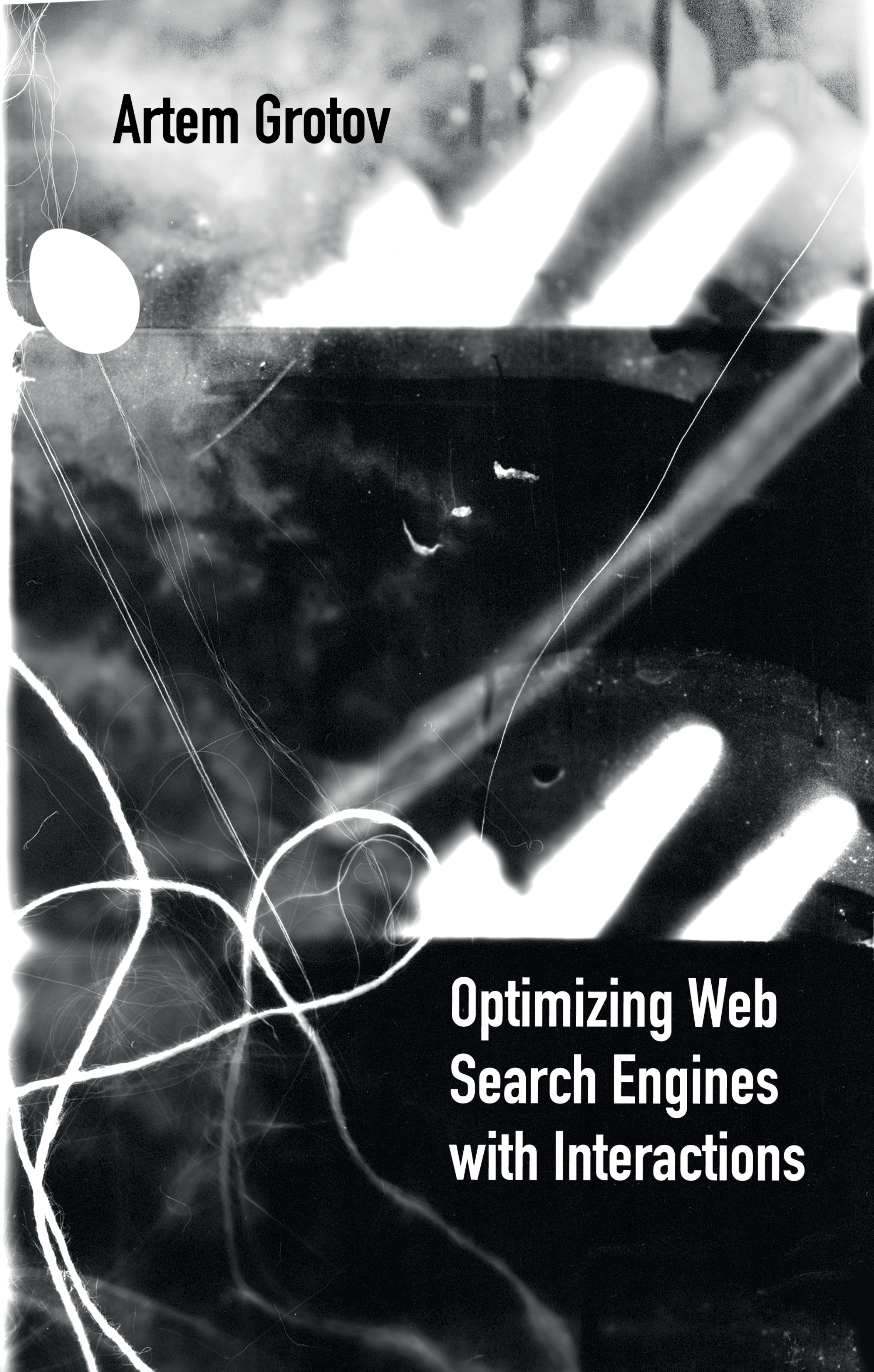


UNIVERSITY
OF AMSTERDAM

Optimizing Web Search Engines with Interactions

Artem Grotov



Artem Grotov

# Optimizing Web Search Engines with Interactions

# Optimizing Web Search Engines with Interactions

Artem Grotov

# Optimizing Web Search Engines with 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
op 7 november 2018, te 11:00 uur

door

Artem Grotov

geboren te Chelyabinsk, Rusland

The research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 612.001.116 and by the Criteo Faculty Research Award program.

ISBN: 978-94-6182-912-2

## Acknowledgements

The thesis would not have come about without the help of many.

I want to thank my supervisors: Maarten de Rijke, Julia Kiseleva, and, during the early stages of my PhD studies, Shimon Whiteson, for their support and patience.

I want to thank my committee members: Lynda Hardman, Claudia Hauff, Jaap Kamps, Evangelos Kanoulas and Cees de Laat.

I want to thank current colleagues: Alexey Borisov, Ana Lucic, Anna Sepliarskaia, Boris Sharchilev, Chang Li, Chuan Wu, Damien Lefortier, Dan Li, Dat Tien Nguyen, David van Dijk, Eva Kenny, Hamidreza Ghader, Harrie Oosterhuis, Hosein Azarbonyad, Ilya Markov, Jiahuan Pei, Jie Zou, Ke Tran Manh, Maartje ter Hoeve, Marzieh Fadaee, Mostafa Dehghani, Nikos Voskarides, Petra Best, Praveen Dakwale, Rolf Jagerman, Shao-Jie Jiang, Xinyi Li, Yifan Chen, Ziming Li, and past colleagues: Abdallah El Ali, Adith Swaminathan, Aleksandr Chuklin, Anne Schuth, Bob van de Velde, Christophe Van Gysel, Daan Odijk, David Graus, Dilek Onal, Evgeny Sherkhonov, Fei Cai, Katya Garmash, Floor Sietsma, Fons Laan, Isaac Sijaranamual, Ivan S. Zapreev, Maria-Hendrike Peetz, Marlies van der Wees, Masrour Zoghi, Richard Berendsen, Shangsong Liang, Spyros Martzoukos, Tom Kenter, Zhaochun Ren.

I want to thank my friends outside of work: Oleg Goryunov and Evgeny Karpov.

I want to thank my family: Tatyana and Olga Grotova and my grandmother Marina.

And I want to thank my partner Yared for loving me all this time!

Artem Grotov
Amsterdam, August 20, 2018

# Contents

# 1 Introduction

Web search engines such as Baidu, Bing, Google and Yandex are some of the most visited websites. Many people use them to access information available on the web. Google alone serves more than six billion queries per day [59]. Half the world population trusts search engines as a source of information and in some countries more than television or news papers [36]. Search engines may influence public opinion [133]. Additionally, web search is one of the biggest businesses on the web [128].

Ranking documents in response to a query is a crucial step in producing a search engine result page (SERP). The goal of a search engine *ranker* is to make the most *relevant* documents easily accessible to its users by putting them at the top of the ranking. Relevance is a multidimensional, dynamic, and complex concept that has been studied extensively within the IR community [12]. Intuitively, relevant documents are the ones that satisfy the user's information need that is expressed as a query.

Ranking is typically done by combining tens, hundreds or even thousands of features into a score that is used to sort the documents [112]. Examples of such features are the tf-idf or BM25 ranking functions – they assign scores to documents with respect to a query by looking at the word overlap between them and statistical properties of the words that overlap such as document and corpus frequencies, which are assumed to capture aspects of a document's relevance [117]. Another example is a query independent ranking function such as PageRank – it ranks documents with respect to their centrality in the hyperlink graph, which is assumed to capture aspects of a document's authoritativeness [75]. If the number of features is small then a ranker can be created manually by using a weighted sum of the features with manually assigned weights as the ranking function. However, this is impractical when the number of features is large or one wishes to use a more complex function for ranking such as forests of decision trees. It is even more impractical when one wants to learn query or document representations using neural networks. In order to learn more complex ranking functions, machine learning – and in particular Learning to Rank (LTR) – has been extensively used by the community [15, 16, 28, 112].

One cannot approach learning without being able to measure its success. Evaluating rankers has been, and continues to be, a widely researched topic in the Information Retrieval (IR) community. Many measures of ranker quality have been proposed in the literature, such as Normalized Discounted Cumulative Gain, Mean Average Precision [119], Expected Browsing Utility [151], Expected Reciprocal Rank [20], and many others. Learning to Rank and measuring ranker quality go hand in hand and the approaches in both learning and evaluation fall into two general categories: using annotated datasets – supervised learning – or by using recorded user interactions with the search engine.

In the supervised learning approach the relevance of a representative set of query document pairs is annotated by human judges. Such LTR datasets can be used both for learning new rankers and for evaluating existing ones. The human judges who annotate the relevance of documents can be either expert annotators or crowdsourced workers. Since the early 1990s community-based efforts such as TREC,[1] CLEF,[2] INEX[3] and NTCIR[4] have produced a number of such datasets [140]. Independently, companies behind large commercial search engines such as Microsoft [76], Yahoo [18] and Yandex [150] have also shared annotated datasets, as have companies behind recommender systems such as Netflix [99] and MovieLens [98]. The supervised approach is well studied and has high internal validity and repeatability of experiments. However, creating such labeled datasets can be expensive [2] and the performance of rankers measured using standard IR metrics may not accurately reflect the users' satisfaction [134, 137].

In order to overcome these limitations, learning and evaluation of rankers can be done using observed user interactions with the search engine result page. User interactions with a search engine can reflect user preferences [65, 66]. They are available in large amounts in the form of search engines logs. They are collected from real users so may be less biased than datasets collected from hired human judges. In this thesis we study some of the key challenges of learning from user interactions collected in this manner. In particular, we focus on how to make sense of the interactions of users with a search engine using click models, how to use them to evaluate rankers, how to use interactions to train deep neural networks, and how to learn from interactions without a pre-specified performance metric.

In the next section we outline the research in this thesis and the questions that are answered within it.

## 1.1 Research Outline and Questions

The main research problem underlying this thesis is: "How can we interactively optimize web search engines using interactions?" We split our main research problem into four concrete RQs:

**RQ1** Which is the best click model to model interactions with a search engine result page?

**RQ2** Does quantifying uncertainty in click models help to evaluate search engine rankers?

**RQ3** How to optimize deep neural networks using implicit feedback to perform learning to rank online?

[1]https://trec.nist.gov
[2]www.clef-initiative.eu
[3]https://inex.mmci.uni-saarland.de
[4]http://research.nii.ac.jp/ntcir/index-en.html

**RQ4** Is it possible to optimize interactive systems using user interactions without explicit reward?

We use a web search engine as a canonical instantiation of an interactive system when we address the first three research questions. The findings in these chapters can generalize to other interactive systems that have ranking as the primary component and there is a strong and known connection between user behavior and user satisfaction such as recommender systems [93], e-commerce [123] and advertising [68]. We use GridWorld [10] as an instantiation of an interactive system when we address the fourth research question. The findings, however, can generalize to cases where the interactive system can be modeled as a Markov Decision Process (MDP) as in dialogue systems [126] and web search sessions [90].

### 1.1.1 **RQ1:** Which is the best click model to model interactions with a search engine result page?

Let us turn to the first research question. We start by investigating ways to model the interactions that are observed between a user and a search engine. We focus on the task of ranking documents on the search engine result page according to their relevance to a user's query. The user interacts with the search engine result page by clicking on the results, scrolling the viewport, moving the mouse cursor and issuing new queries and in other ways depending on the device. In this chapter we only consider clicks.

Clicks are modeled using so-called click models [24] that allow us to extract useful information from noisy and biased click data. They exploit the relationship between clicks and the underlying relevance of the clicked result while removing different types of bias, such as position bias [27] (the phenomenon that items ranked at the top get clicked because they are ranked at the top and not necessarily because they are the best items), and de-noising the signal. Many click models have previously been proposed in the literature, however they were never properly evaluated and compared to each other on the same open dataset using the same set of tasks such as explaining and predicting clicks, ranking and relevance prediction [24]. In particular, we compare the following ones:

- Click-Through Rate models [41]
- Position-Based Model [41]
- Cascade Model [27]
- User Browsing Model [35]
- Dependent Click Model [46]
- Click Chain Model [45]
- Dynamic Bayesian Network [19] .

We evaluate and compare the click models on a number of dimensions, using multiple metrics:

- Which model explains the observed data best as measured by log-likelihood and perplexity?
- Which model predicts future clicks best as measured by root mean squared error?
- Which model can be used to predict relevance as annotated by professional annotators as measured by AUC?
- Which model's predicted relevance can improve the ranking most as measured by NDCG@5?

We find that there is no single click model that is the best across all metrics. Moreover, while clicks and annotated relevance are certainly correlated, they are not the same and optimizing for one or the other one will yield different results in terms of ranking performance. This further exemplifies the gap between expert annotations and user needs noted earlier in this introduction and motivates the development of methods that can learn from user interactions.

### 1.1.2 **RQ2:** Does quantifying uncertainty in click models help to evaluate search engine rankers?

For our second research question we investigate how much one can learn from a fixed interaction log. User interaction logs are never complete and unbiased so it is important to understand how much one can learn from a given interaction log without collecting new data. One way of quantifying this is by looking at which pairs of interactive systems we can compare given the data and which we cannot. It is also an important question because these comparisons can be used for gradient descent which is a common technique in machine learning.

So we investigate the question of quantifying the information in user interaction logs by looking at the ranker comparison task. This is the task of asking whether ranker $A$ is better than ranker $B$ given some dataset of user interactions with the search engine. In order to model the confidence of the comparison we propose a Bayesian framework of modeling the distribution of the relevance of a query document pair from observed user interaction; based on this, we derive the posterior probability of one ranker being better than another ranker.

The resulting algorithm, called Bayesian Ranker Comparison (BARACO), decides if the observed data are enough to conclude that ranker $A$ is better than ranker $B$. The specific research questions that we pursue in Chapter 3 are:

- Can BARACO determine whether the production ranker should be replaced with a candidate ranker better than a non-Bayesian approach to making this decision?
- Does BARACO produce better estimates of the difference in performance of rankers than the non-Bayesian baseline?

### 1.1.3 **RQ3:** How to optimize deep neural networks using implicit feedback to perform learning to rank online?

There have been many publications on online learning to rank [25, 40, 53, 55, 56, 74, 121, 160]. However, these works have one important shortcoming – very simplistic ranking models are used. Most of them either use a linear model or a tabular model. Both types of model seem at odds with ongoing developments in a discipline that has a very strong focus on deep neural networks or ensembles of trees for ranking.

To address this gap we propose to train deep neural nets from user interactions online to perform an image filtering task. The image filtering task is the task of ranking a changing set of documents for a set of standing information needs. Examples include video surveillance with a fixed visual vocabulary, (visual) reputation monitoring, visual information discovery services such as Pinterest,[5] visual ranking systems with a fixed vocabulary such as those built into mobile or desktop photo management applications, and product search where the number of product types is fixed (as is the case in most established e-commerce platforms).

In order to solve this task using deep neural networks one must solve two fundamental issues:

- How to update deep neural networks based on an incoming stream of implicit user feedback? and
- How to explore the space of possible image rankings obtained using deep ranking models?

We propose two loss functions and compare three exploration methods on the MSCOCO image dataset [85] and find that a regression based DCG-loss with epsilon greedy exploration has the highest performance.

### 1.1.4 **RQ4:** Is it possible to optimize interactive systems using user interactions without explicit reward?

For our final research question we start from the observation that interactively learning from user interactions is difficult not only because it is hard to design the right learning algorithm and efficiently explore the action space but also because it is often not clear what is the right metric to optimize. For example, naively optimizing for click-through rate (CTR) leads to a search engine result page populated by click-bait. Therefore, for the task of document ranking dozens of metrics have been proposed, ranging from very simple ones such as Expected Browsing Utility [151], Expected Reciprocal Rank [20] to ones driven by machine learning such as the user satisfaction predictors in [47–49, 64, 70].

However, all these metrics are founded on explicitly modeling the relationship between the observed user interactions and user satisfaction. Sometimes these models are hard to build; for example, dialogue systems have been particularly resistant to automatic evaluation. This motivates us to mine user satisfaction metrics that we can use as learning objectives from user interactions automatically.

[5]http://pinterest.com/

To address our final research question, we introduce a novel algorithm, called Interactive System Optimizer (ISO). It provides a new principled approach to optimizing interactive systems without a predefined satisfaction metric by concurrently inferring data-driven objectives from user interactions and optimizing the interactive system accordingly. Thus, ISO does not depend on any domain knowledge.

## 1.2 Main Contributions

In this section we summarize the main contributions of this thesis. Our contributions come in the form of algorithmic, empirical, and software contributions.

### 1.2.1 Algorithmic contributions

1. BARACO – an MCMC algorithm to compute the posteriors of the Dynamic Bayesian Network click model. It can be used to evaluate ranker quality and compare rankers in Chapter 3.

2. Two loss functions for image filtering using Deep Neural Networks – DCG-loss and PG-loss – in Chapter 4.

3. ISO – an algorithm to optimize interactive systems without pre-specified loss functions in Chapter 5.

### 1.2.2 Empirical contributions

1. We present the outcomes of a detailed analysis of several major click models for web search: Click-Through Rate Models [41], the Position-Based Model [41], the Cascade Model [27], the User Browsing Model [35], the Dependent Click Model [46], the Click Chain Model [45] and the Dynamic Bayesian Network [19]. We use the first 32 million query sessions from the 2011 Yandex Relevance Prediction contest,[6] open-source software,[7] and a range of evaluation techniques such as log-likelihood, perplexity, relevance prediction, click prediction and ranking performance, which makes our results both representative and reproducible. We find that there is no click model that outperforms all others on all metrics. The results of the analysis are presented in Chapter 2.

2. We provide a comparison of the performance of Bayesian and Expectation Maximization (EM) inference of the Dynamic Bayesian Network (DBN) click model and find that Bayesian inference is more precise but much more expensive. We analyze the performance of BARACO under violations of the click model assumptions and using real and simulated clicks and find that it is robust to the model mismatch. The results of the analysis are presented in Chapter 3.

[6]`http://imat-relpred.yandex.ru/en/datasets`
[7]`https://github.com/markovi/PyClick`

3. We also provide a comparison of several exploration strategies and analyze the performance of the algorithms and the effect of list size on ranking performance. We find that DCG-loss with $\epsilon$-greedy exploration is the best choice. We find that bigger list sizes make the problem more difficult. We present the results in Chapter 4.

4. We present the outcomes of an analysis of the performance of Interactive System Optimizer with labeled and unlabeled trajectories, with optimal and suboptimal behavior in Chapter 5.

### 1.2.3 Software contributions

To facilitate future research, we share the implementations that we built to support the experiments that we ran to obtain the results in the thesis:

1. Implementation of Bayesian Ranker Comparison in Chapter 3,

2. Implementation of the image filtering approaches in Chapter 4, and

3. Implementation of Interactive System Optimizer in Chapter 5.

The details of the implementations are given in Appendix B.

## 1.3 Thesis Overview

This section gives an overview of the content of each chapter of this thesis.

Chapters 2–5 are the main research chapters of this thesis. Briefly, the first two chapters are in the context of ad-hoc search and click logs. In Chapter 2 we study different ways of modeling clicks. In Chapter 3 we use click models studied in the previous chapter to evaluate web search rankers. In Chapter 4 we study how to optimize neural ranking systems with list-wise feedback when the user objective is defined and observed. In Chapter 5 we remove the assumption that we know what the user wants and propose an algorithm to optimize interactive systems without predefined user objective. Let us take a closer look now.

In Chapter 2 we look at ways of modeling user interactions, in particular we look at clicks in the context of a search engine. Click models have become an essential tool for understanding user behavior on a search engine result page, for running simulated experiments, and for predicting relevance. We analyze the impact of query frequency and find that log-likelihood of complex click models is more stable across different query frequencies than that of the CTR-based models meaning that they can handle less frequent queries better. We also analyze the impact of click entropy on the performance of a click model and find that the lower the click entropy the easier it is to approximate clicks and, hence, the better the performance of click models.

Now that we have acquired an understanding of click models, we proceed to use them to evaluate search engine rankers in Chapter 3. We address the problem of how to safely compare rankers for information retrieval. In particular, we consider how to control the risks associated with switching from an existing *production ranker* to a new *candidate ranker*. Whereas existing online comparison methods require showing potentially suboptimal result lists to users during the comparison process, which can lead to user frustration [56] and bad abandonment, our approach only requires user interaction data generated through the natural use of the production ranker. Specifically, we propose a Bayesian approach for:

1. Comparing the production ranker to candidate rankers, and
2. Estimating the confidence of this comparison.

The comparison of rankers is performed using click model-based information retrieval metrics, while the confidence of the comparison is derived from Bayesian estimates of uncertainty in the underlying click model. These confidence estimates are then used to determine whether a risk-averse decision criterion for switching to the candidate ranker has been satisfied. Experimental results on the LETOR datasets [112] and on a click log used in the WSDM 2014 Web search personalization challenge,[8] show that the proposed approach outperforms an existing Expectation Maximization-based ranker comparison method.

In Chapter 4 we switch from the evaluation of rankers to their optimization. In this chapter, we focus on Online Learning to Rank (OLTR) from implicit feedback for image filtering, which differs from traditional document retrieval in that high quality features, such as BM25 or PageRank for ad-hoc web search, are typically available for document retrieval while image retrieval relies on raw pixel values and deep neural networks. The main challenges in OLTR are to predict the best ranking and to explore the space of possible rankings so as to be able to learn new rankings. To predict the best ranking of images we propose to use ResNet [51] trained with implicit feedback and two loss functions: one based on regression, DCG-loss, and one based on Policy Gradients, PG-loss. To explore the space of possible rankings, we use and compare five explorations methods: $\epsilon$-greedy exploration [94], Boltzmann exploration [67, 129], bootstrapped exploration [105], pure exploration, and exploitation (i.e., no exploration). We find that DCG-loss with $\epsilon$-greedy exploration performs best in terms of expected future NDCG and that DCG-loss without exploration has the best performance during training. Additionally, our methods are more general than existing OLTR methods because they only require list-wise implicit feedback as opposed to item-level feedback. Using DCG-loss it is now possible to learn optimal rankings of images online.

Finally, in Chapter 5 we look at optimizing interactive systems when a clear notion of user satisfaction has not been provided but instead we restore this notion from observed interaction data. Effective optimization is essential for interactive systems to provide a satisfactory user experience. However, it is often challenging to find an objective to optimize for. In this chapter we propose an approach that infers the objective directly from observed user interactions. These inferences can be made regardless of prior knowledge and across different types of user behavior. We introduce Interactive System Optimizer (ISO), a novel algorithm that uses these inferred objectives for optimization. Our main contribution in this chapter is a new general principled approach to optimizing interactive systems using data-driven objectives – objectives recovered from observed

[8]Personalized Web Search Challenge 2013
`https://www.kaggle.com/c/yandex-personalized-web-search-challenge`

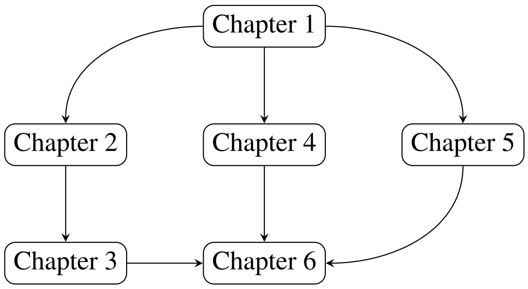


**Figure 1.1: Suggested reading paths through the thesis.**

user interaction rather than defined explicitly by a system designer. We demonstrate the high effectiveness of ISO using several GridWorld simulations.

The second and the third chapter of the thesis are closely related: the former investigates click models and the latter applies them for ranker comparison. Therefore, readers unfamiliar with click models should start with Chapter 2 and those familiar with click models can skip to Chapter 3. Chapters 4 and 5 are independent from each other and can be read in any order. See Figure 1.1 for suggested reading paths through the thesis.

## 1.4 Origins

The research in this thesis is based on the following publications:

**Chapter 2** is based on A. Grotov, A. Chuklin, I. Markov, L. Stout, F. Xumara, and M. de Rijke. A comparative study of click models for web search. In *CLEF*. Springer, September 2015. AG ran the experiments, performed the analysis, worked on the implementation of click models, and worked on the text. AC performed analysis and worked on the text. LS, FX and IM worked on the implementation of click models and worked on the text.

**Chapter 3** is based on A. Grotov, S. Whiteson, and M. de Rijke. Bayesian ranker comparison based on historical user interactions. In *SIGIR*, pages 273–282. ACM, August 2015. AG wrote and ran the code for the experiments. All authors contributed to the text, AG did most of the writing.

**Chapter 4** is based on C. Li, A. Grotov, B. Eikema, I. Markov, and M. de Rijke. Deep online learning to rank for image filtering with implicit feedback. In *Submitted*, 2018. AG, CL, BE wrote and ran the code for the experiments. All authors contributed to the text.

**Chapter 5** is based on Z. Li, A. Grotov, J. Kiseleva, M. de Rijke, and H. Oosterhuis. Optimizing interactive systems with data-driven objectives. *arXiv preprint arXiv:1802.06306*, February 2018. ZL wrote and ran the code for the experiments. AG, ZL, JK developed the theory and algorithms. All authors contributed to the discussions and text.

Indirectly, the thesis also benefited from work on the following publications:

- Y. Norouzzadeh Ravari, I. Markov, A. Grotov, M. Clements, and M. de Rijke. User behavior in location search on mobile devices. In *ECIR*, pages 728–733. Springer, April 2015.
- A. Grotov and M. de Rijke. Online learning to rank for information retrieval: SIGIR 2016 tutorial. In *SIGIR*, pages 1215–1218. ACM, July 2016.
- Z. Li, J. Kiseleva, M. de Rijke, and A. Grotov. Towards learning reward functions from user interactions. In *ICTIR*, pages 941–944. ACM, 2017.

# 2
# A Comparative Study of Click Models for Web Search

In this chapter we address **RQ1:** Which is the best click model to model interactions with a search engine result page? User interactions with a web search engine take many different forms, e.g., clicks, queries, query reformulations, dwell times, transactions etc. In this chapter we focus on one of the most important and widely used signals, which is clicks. There are many models that model clicks on the search engine result page and in this chapter we compare a number of them on a common set of metrics and the same dataset, which has not been done before. The goal is to understand if there is a unique model that is better than all others and how well the models agree with annotated relevance judgements.

We examine several click models, and find that depending on the metric being considered some are better than other ones and there is no clear winner. This means that when choosing a model of interactions it is important to evaluate the model on the task directly rather than some related metric because the performance can be very metric dependent. For example, a click model that has the highest log-likelihood of the observed clicks may not be the best to predict relevance or to serve as a ranking feature.

## 2.1 Introduction

Modeling user behavior on a search engine result page (SERP) is important for understanding users, supporting simulation experiments [52, 55], evaluating web search results [20, 23] and improving document ranking [19, 34]. In recent years, many models of user clicks in web search have been proposed [24]. However, no comprehensive evaluation of these click models has been performed using publicly available datasets and a common set of metrics with a focus on an analysis of the query space. As a result, it is not clear what the practical advantages and drawbacks are of each proposed model, how different models compare to each other, which model should be used in which settings, etc.

In this chapter we aim to compare the performance of different click models using a common dataset, a unified implementation and a common set of evaluation metrics. We

This chapter was published as [84].

**Table 2.1: Notation used in the chapter.**

| Symbol | Description |
|---|---|
| $u$ | A document |
| $q$ | A query |
| $s$ | A search query session |
| $j$ | A document rank |
| $c$ | A click on a document |
| $\mathcal{S}$ | A set of sessions |
| $E$ | A random variable for document examination |
| $R$ | A random variable for document relevance |
| $C$ | A random variable for a click on a document |
| $\epsilon$ | The examination parameter |
| $r$ | The relevance parameter |

consider all major click models for web search ranging from the simple Click-Through Rate model (CTR), Position-Based Model (PBM) and Cascade Model (CM) [27] through the more advanced Dependent Click Model (DCM) [46] to the more complex User Browsing Model (UBM) [35], Dynamic Bayesian Network model (DBN) [19], and Click Chain Model (CCM) [45]. These models are evaluated using log-likelihood, perplexity, click-through rate prediction, relevance prediction, ranking performance and computation time.

We also analyze two different factors that influence the performance of click models, namely, query frequency and click entropy. Intuitively, it is easier to predict clicks for frequent queries than for less frequent ones because of the larger size of the training data and the relatively more uniform click patterns associated with frequent queries. Click entropy can be used to distinguish between navigational and informational queries. Navigational queries tend to have low click entropy (usually only the top result is clicked), while informational queries tend to have high click entropy (several results may be clicked before a user's information need is satisfied).

Our main finding in this chapter is that no single model excels on each of the considered metrics and that sometimes simple models outperform complex ones and that the relative performance of models can be influenced by the dataset characteristics such as query frequency and click entropy. These results can guide the application of existing click models and inform the development of new click models.

## 2.2 Click Models

In this section, we give an overview of all major click models for web search, which we will then use in our comparative study

**Click-Through Rate Models** Three simple click models, all based on click-through rates, predict click probabilities by counting the ratio of clicks to the total number of impressions. In the simplest case of Global CTR (GCTR) this ratio is computed globally for all documents, while in Rank CTR (RCTR) it is computed separately for each rank $j$ and in Document CTR (DCTR) for each document-query pair $uq$:

$$P_{GCTR}(C_u = 1) = r = \frac{1}{\sum_{s \in \mathcal{S}} |s|} \sum_{s \in \mathcal{S}} \sum_{u \in s} c_{uq} \tag{2.1}$$

$$P_{RCTR}(C_{u_j} = 1) = r_j = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} c_j \tag{2.2}$$

$$P_{DCTR}(C_u = 1) = r_{uq} = \frac{1}{|\mathcal{S}_{uq}|} \sum_{s \in \mathcal{S}_{uq}} c_{uq}, \text{ where } \mathcal{S}_{uq} = \{s_q : u \in s_q\}. \tag{2.3}$$

**Position-Based Model** This model builds upon the CTR models and unites DCTR with RCTR. It adds a separate notion of examination probability ($E$) which is subject to *position bias*, where documents with smaller rank are examined more often; the document can only be clicked if it was examined and is relevant:

$$C_{uq} = 1 \Leftrightarrow (E_{j_u} = 1 \text{ and } R_{uq} = 1)\,. \tag{2.4}$$

The examination probability $\epsilon_j = P(E_{j_u} = 1)$ depends on the rank $j$, while the relevance $r_{uq} = P(R_{uq} = 1)$ depends on the document-query pair. Inference of this model is done using the Expectation Maximization algorithm (EM).

**Cascade Model** The Cascade Model (CM) [27] is another extension to the Click-Through Rate (CTR) models. The model introduces the *cascade hypothesis*, whereby a user examines a SERP from top to bottom, deciding whether to click each result before moving to the next one; users stop examining a SERP after the first click. Inference of the parameters of CM is done using Maximum Likelihood Estimation (MLE). The click probability is defined using the examination (2.4) and the cascade assumptions:

$$P(E_1 = 1) = 1 \tag{2.5}$$

$$P(E_j = 1 \mid E_{j-1} = e, C_{j-1} = c) = e \cdot (1 - c), \tag{2.6}$$

where $e$ and $c$ are 0 or 1, and the only parameters of the model are $r_{uq} = P(R_{uq} = 1)$. The fact that users abandon a search session after the first click implies that the model does not provide a complete picture of how multiple clicks arise in a query session and how to estimate document relevance from such data.

**User Browsing Model** Dupret and Piwowarski [35] propose a click model called the User Browsing Model (UBM). The main difference between UBM and other models is that UBM takes into account the distance from the current document $u_j$ to the last clicked document $u_{j'}$ for determining the probability that the user continues browsing:

$$P(E_{j_u} = 1 \mid C_{u_{j'}} = 1, C_{u_{j'+1}} = 0, \ldots, C_{u_{j-1}q} = 0) = \gamma_{jj'}. \tag{2.7}$$

**Dependent Click Model** The Dependent Click Model (DCM) by Guo et al. [46] is an extension of the cascade model that is meant to handle sessions with multiple clicks. This model assumes that after a user clicked a document, they may still continue to examine other documents. In other words, (2.6) is replaced by

$$P(E_j = 1 \mid E_{j-1} = e, C_{j-1} = c) = e \cdot (1 - c + \lambda_j c), \tag{2.8}$$

where $\lambda_j$ is the continuation parameter, which depends on the rank $j$ of a document.

**Click Chain Model** Guo et al. [45] further extend the idea of DCM into the Click Chain Model (CCM). The intuition behind CCM is that the chance that a user continues after a click depends on the relevance of the previous document and that a user might abandon the search after a while. This model can be formalized with (2.4) and the following conditional probabilities:

$$P(E_{j_u+1} = 1 \mid E_{j_u} = 1, C_{uq} = 0) = \tau_1 \tag{2.9}$$
$$P(E_{j_u+1} = 1 \mid E_{j_u} = 1, C_{uq} = 1) = \tau_2(1 - r_{uq}) + \tau_3 r_{uq}. \tag{2.10}$$

**Dynamic Bayesian Network Model** The DBN model [19] takes a different approach in extending the cascade model. Unlike CCM, DBN assumes that the user's perseverance after a click depends not on the relevance $r_{uq}$, but on a different parameter $s_{uq}$ called satisfaction parameter. While $r$ is mostly defined by the snippet on the SERP, the satisfaction parameter $s$ depends on the actual document content available after a click. The DBN model is defined by (2.4) and the following formulas:

$$P(E_{j_u+1} = 1 \mid E_{j_u} = 1, C_{uq} = 0) = \gamma \tag{2.11}$$
$$P(E_{j_u+1} = 1 \mid E_{j_u} = 1, C_{uq} = 1) = \gamma(1 - s_{uq}), \tag{2.12}$$

where $\gamma$ is a continuation probability after a non-satisfactory document (either no click, or click, but no satisfaction).

In general, inference should be done using the EM algorithm. However, if $\gamma$ is set to 1, the model allows easy MLE inference. We refer to this special case as the Simplified DBN (SDBN) model.

## 2.3 Evaluation Measures

Different studies use different metrics to evaluate click models [24]. In this section we give an overview of these metrics. We will then use all of them in our comparative study.

**Log-likelihood** Log-likelihood evaluates how well a model approximates observed data. In our case, it shows how well a click model approximates clicks of actual users. Given a model $M$ and a set of observed query sessions $\mathcal{S}$, log-likelihood is defined as follows:

$$\mathcal{LL}(M) = \textstyle\sum_{s \in \mathcal{S}} \log P_M \left(C_1, \ldots, C_n\right), \tag{2.13}$$

where $P_M$ is the probability of observing a particular sequence of clicks $C_1, \ldots, C_n$ according to the model $M$.

**Perplexity** Perplexity measures how surprised a model is to see a click at rank $r$ in a session $s$ [35]. It is calculated for every rank individually:

$$p_r(M) = 2^{-\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left(c_r^{(s)} \log_2 q_r^{(s)} + \left(1 - c_r^{(s)}\right) \log_2 \left(1 - q_r^{(s)}\right)\right)}, \tag{2.14}$$

where $c_r^{(s)}$ is the actual click on the document at rank $r$ in the session $s$, while $q_r^{(s)}$ is the probability of a user clicking the document at rank $r$ in the session $s$ as predicted by the model $M$, i.e., $q_r^{(s)} = P_M(C_r = 1)$.

The total perplexity of a model is defined as the average of perplexities over all positions. Lower values of perplexity correspond to higher quality of a click model.

**Click-through rate prediction** Click-Through Rate (CTR) is a ratio of the cases when a particular document was clicked to the cases when it was shown. In [19], the following procedure was proposed to measure the quality of click models using CTR:

- Consider a document $u$ that appears both on the first position and on some other positions (in different query sessions).
- Hold out as a test set all the sessions in which $u$ appears on the first position.
- Train a click model $M$ on the remaining sessions.
- Use the model $M$ to predict clicks on the document $u$ on the held-out test set (predicted CTR).
- Compute the actual CTR of $u$ on the held-out test set.
- Compute the Root Mean Squared Error (RMSE) between the predicted and actual CTRs.

**Relevance prediction** It was noticed in [19] that click models can approximate document relevance. A straightforward way to evaluate this aspect is to compare document relevance as predicted by a model to document relevance labels provided by human annotators. We measure the agreement between the two using the Area Under the ROC Curve (AUC) and Pearson correlation.

**Predicted relevance as a ranking feature** The predicted relevance can also be used to rank documents [19]. The performance of such a ranker can be evaluated using any standard IR measure, such as MAP, DCG, etc. In this study, we use NDCG@5 [63]. To calculate NDCG@5 we only consider documents for which we have relevance labels. The evaluation is performed as follows:

- Retrieve all sessions that have complete editorial judgments.
- Sort sessions by session id.
- The first 75% are training sessions, the remainder are test sessions.
- Train the model on the training sessions and predict relevance for the test sessions.
- Sort the documents w.r.t the predicted relevance given by the model.
- Compute the NDCG@5.
- Average over all sessions.

**Computation time** Historically, in machine learning a big problem in creating accurate models was the amount of data that was available. However, this is no longer the case, and now we are mostly restricted by the time it takes to learn a model based on a large amount of available data. This makes the ability to efficiently compute parameters an important feature of a successful model. Therefore, we also look at the time it takes to train a click model.

## 2.4 Experimental Setup

Our goal is to evaluate and compare the click models presented in Section 2.2 using the evaluation metrics described in Section 2.3. To this end we use the first 32 million query sessions from the 2011 Yandex Relevance Prediction contest.[1] In this contest participants were asked to predict document relevance based on click log data. We split the session set into 32 batches of one million sessions each and measured, for every click model, the log-likelihood, perplexity, RMSE of CTR prediction and computation time for each of the batches. Then we average the measurements across the batches.

The sessions in each batch are sorted based on their session id and divided into a set of training sessions used to train the click models and a set of test sessions used in the evaluation of the models; the number of sessions in these sets have a 3 to 1 ratio.

To measure the quality of relevance prediction and ranking performance we use sessions for which all the documents have relevance labels. For each query all except the last session is used for training and the last session is used for testing. There are $861{,}000$ search sessions and 178 unique queries in the training set and 112 queries in the test set.

To determine whether observed differences are statistically significant we use a two-tailed student-t test with $p$ values below $0.05$ indicating significant differences. The error bars in the plots below are standard errors of the means.

**Performance impacting factors** To evaluate the effect of *query frequency* on click model performance, we split the data into four parts (see Table 2.2).

**Table 2.2: The distribution of sessions with respect to query frequency.**

| Query frequency | Number of sessions |
|---|---:|
| 2 | 6,940,000 |
| 3–5 | 12,800,000 |
| 6–19 | 16,600,000 |
| 20+ | 108,000,000 |

Another factor that may influence click model performance is *click entropy*. Click entropy has been used to analyze queries in [31]. The formal definition of the entropy

[1]`http://imat-relpred.yandex.ru/en/datasets`

of query $q$ is:

$$\mathit{ClickEntropy}(q) = -\textstyle\sum_{d \in \mathcal{P}(q)} P(d \mid q) \log_2 P(d \mid q), \tag{2.15}$$

where $\mathcal{P}(q)$ are documents clicked on for query $q$ and $P(d \mid q)$ is the fraction of clicks on document $d$ among all clicks on $q$,

$$P(d \mid q) = \sum_{p} c_{r_d}^{(q)} \cdot \left( \sum_{u \in \mathcal{P}(q)} c_{r_u}^{(q)} \right)^{-1}. \tag{2.16}$$

Click entropy can be used to distinguish navigational and informational queries. In navigational queries users know what they are looking for so the click entropy will be low because almost all clicks within that query will be on the same document. In an informational query the users explore different results to find the optimal one because they do not know what document they are looking for yet. This gives these queries a high click entropy. We divide our search sessions into three bins with respect to click entropy and report on evaluation measures per bin; statistics of these bins are listed in Table 2.3.

**Table 2.3: The distribution of sessions with respect to click entropy.**

| Click entropy | Number of sessions |
|---|---:|
| 0–1 | 53,400,000 |
| 1–2 | 48,800,000 |
| 2+ | 42,200,000 |

## 2.5 Results

In this section we present the results of our experiments. For every evaluation measure we report the influence of the query frequency and click entropy. Table 2.4 contains the evaluation outcomes for every model when trained on the entire dataset.

**Log-likelihood** Figure 2.1 shows the results of the log-likelihood experiments; shorter bars indicate better results. The Cascade Model (CM) cannot handle multiple clicks in one session and gives zero probability to all clicks below the first one. For such sessions its log-likelihood is $\log 0 = -\infty$ and so the total log-likelihood of CM is $-\infty$.

When evaluated on the whole test set, the User Browsing Model (UBM) shows the best log-likelihood, followed by the Dynamic Bayesian Network (DBN), the Position-Based Model (PBM) and the Click Chain Model (CCM). Note that the Simplified DBN (SDBN) model has lower log-likelihood values compared to its standard counterpart (DBN). The simple CTR-based models show the lowest log-likelihood. This confirms that complex click models explain and approximate user behavior better than simply counting clicks.

Figure 2.1 (left) shows the log-likelihood of click models for different query frequencies. In general, the higher the query frequency (more training data available) the better the performance of click models. When comparing complex click models, there is variation in their relative performance based on the query frequency, but UBM consistently has the highest log-likelihood. SDBN and DCM have considerably lower log-likelihood than the similar models DBN and CCM (apart from the “20+” bin). In contrast, the log-likelihood of the CTR-based models varies considerably across query frequencies. On the “2” and “3–5” bins, Global CTR (GCTR) outperforms SDBN and DCM, while Rank CTR (RCTR) is the second best model overall (after UBM). The Document CTR (DCTR) model has the lowest log-likelihood for all query frequencies but “20+”. There, it outperforms SDBN, DCM and CCM and comes close to PBM. These results show two interesting facts. On the one hand, the log-likelihood of complex click models is more stable across different query frequencies than that of the CTR-based models. On the other hand, for each query frequency bin there is a CTR-based model that has log-likelihood scores comparable to complex models (RCTR for “2–19” and DCTR for “20+”).

Figure 2.1 (right) shows the log-likelihood of click models for queries with different click entropy. In general, the lower the click entropy the easier it is to approximate clicks and, hence, the better the performance of click models. The relative log-likelihood of different click models for different values of click entropy is similar to that for different query frequencies: UBM is followed in different orders by DBN, PBM and CCM; SDBN and DCM have lower log-likelihood than the above; the log-likelihood of the CTR-based models varies across bins (RCTR is better than SDBN and DCM on $(1, 2]$, DCTR is comparable to PBM and CCM on $(2, \infty)$). As a future work, we plan to investigate the relation between query frequency and click entropy.

**Perplexity** Figure 2.2 shows the perplexity of the click models; the lower the better. When evaluated on all test sessions, most of the complex click models (apart from CM and CCM) have comparable perplexity, with DBN and SDBN having the lowest one, but not significantly so. The CTR-based models have higher perplexity than the complex models, which again confirms the usefulness of existing click models for web search.

The trends for different query frequencies (Figure 2.2, left) are similar to those for log-likelihood (Figure 2.1, left): the variation of perplexity of complex click models is not large (but there are different winners on different bins), while the perplexity of the CTR-based models varies considerably (RCTR has the lowest perplexity overall on “2” and “3–5”, DCTR is comparable to other models on “20+”). The trends for different values of click entropy are similar (see Figure 2.2, right). CM performs poorly in all query classes apart from the $[0, 1]$ entropy bin, which is related to the fact that CM is tuned to explain sessions with one click.

**CTR prediction** Figure 2.3 shows the impact of query frequency and click entropy on the CTR prediction task. Here, the simple models, RCTR and CM, outperform some of the more complex ones. This is because the intuition of these models is exactly what this task has set out to measure. The average rank of the documents in the training data

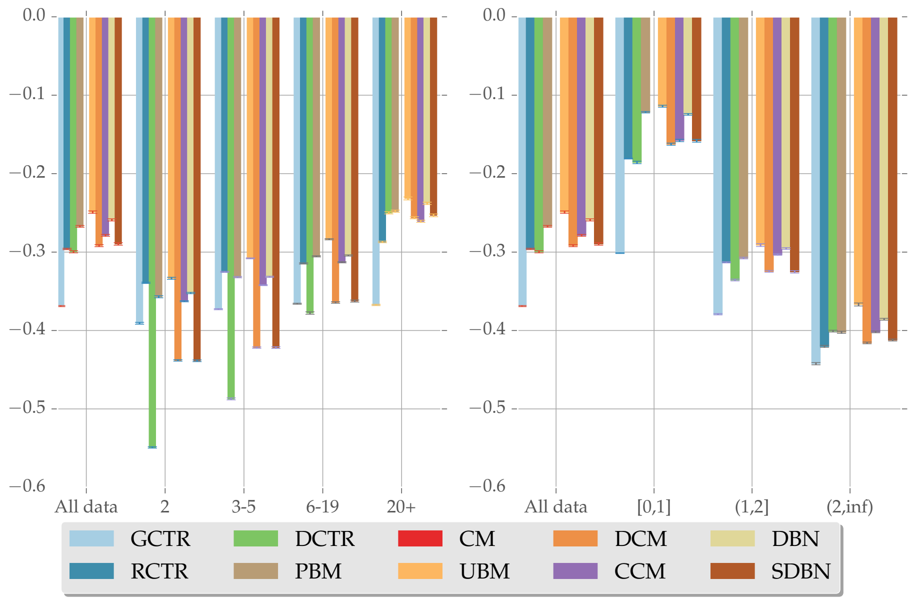


**Figure 2.1: Log-likelihood of click models, grouped by query frequency (left) and click entropy (right).**

set is 2.43, i.e., they were usually in some of the top positions. As the RCTR and CM models both perform well on documents that are ranked high, this high average rank influences the observed performance.

The top performers on this task are SDBN and DCM. It is not clear why there is such a notable gap in performance between DBN and SDBN on this task; it could be speculated that DBN relies more on the satisfactoriness parameters that are not used in this task. Both UBM and PBM have poor performance on this task, we hypothesize that they rely even more on the position dependent parameters and in this task the document under question was presented at a different position.

**Relevance prediction** The results of the relevance prediction task can be seen in Figure 2.4. The plot for different query frequencies could not be generated in a meaningful way, because the queries with judged results do not occur often in the dataset, while the relevance prediction protocol only considers queries that occur at least ten times.

The relevance prediction performance of all click models is relatively low (between 0.500 and 0.581). The GCTR and RCTR models do not have a document-specific parameter and, thus, cannot predict relevance. So their AUC is equal to that of random prediction, i.e., 0.5. UBM and PBM have the highest AUC (0.581), while other models are closer to random prediction (from 0.515 for CM to 0.541 for CCM).

These results show that existing click models still have a long way to go before they can be used for approximating relevance labels produced by human annotators.

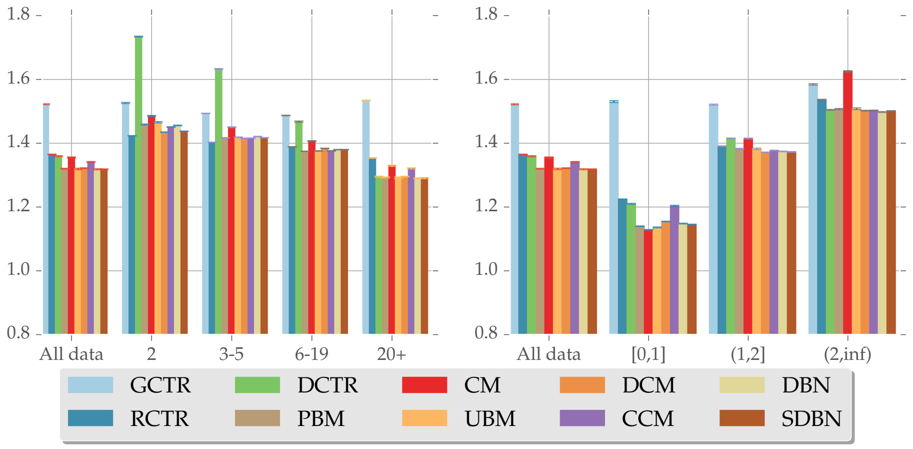


**Figure 2.2: Perplexity of click models, grouped by query frequency (left) and click entropy (right).**

**Predicted relevance as a ranking feature** Figure 2.5 shows the results of using the predicted relevance as a ranking feature. The best model here is CCM, followed by the simple DCTR model. This is not surprising as relevant documents attract more clicks and usually have higher CTRs. Thus, ranking documents based on their CTR values only (as done by DCTR) results in high NDCG@5 scores. Notice, though, that predicting actual relevance labels of documents based on the documents' CTRs is still a difficult task (see the discussion above).

The GCTR and RCTR models do not have document-specific parameters and, thus, cannot rank documents. Therefore, they have the lowest values of NDCG@5. They still have high values of NDCG because no reranking was done for documents with equal relevance estimates, hence the values of NDCG for GCTR and RCTR reflect the ranking quality of the original ranker.

**Computation time** In the rightmost column of Table 2.4 we see that, as expected, the models that use MLE inference are much faster than those with EM inference. When using EM inference to calculate the parameters of a click model, one would ideally use some convergence criteria; we have chosen to do a fixed number of iterations (i.e., 50). Notice that UBM is 5–6 times faster than DBN and CCM, even though they all use EM. DBN and CCM use more complex update rules and this results in such a big difference in training time.

**Overall results** We summarize our experimental results in Table 2.4. There is no perfect click model that outperforms all other models on every evaluation metric. For example, UBM is the best in term of log-likelihood and relevance prediction, while DBN is the best in terms of perplexity and CTR prediction. Even simple CTR-based models have relatively high performance according to some metrics (e.g., DCTR according to NDCG@5).

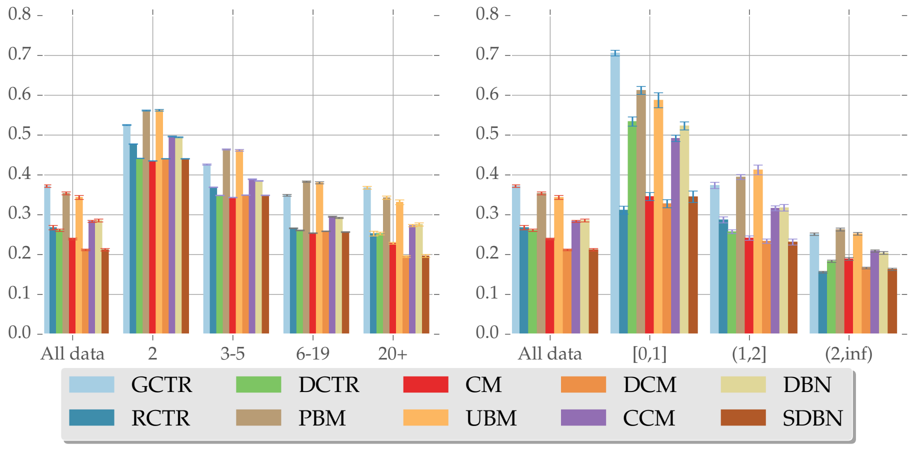


**Figure 2.3: Click-through rate prediction RMSE of click models, grouped by query frequency (left) and click entropy (right).**

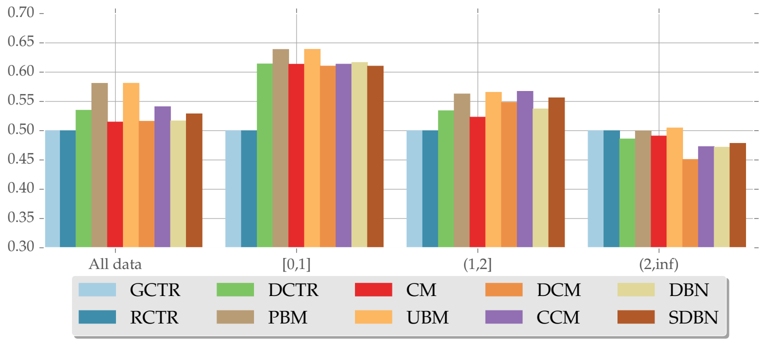


**Figure 2.4: Relevance prediction of click models on click entropy**

## 2.6 Conclusion

In this chapter we addressed RQ1: Which is the best click model to model interactions with a search engine result page? In particular we have studied the problem in the context of clicks on search engine result pages.

We have shown that a universal benchmark is necessary for developing and testing click models. The unified evaluation we performed gave important insights into how click models work. In particular, we found that complex click models dominate most of the evaluation metrics, however, in some cases simple click models outperform state-of-the-art models. We also found that none of the tested click models outperforms all others on all measures, e.g., DBN and SDBN are best when judged by perplexity, UBM is best when judged by likelihood, GCTR and RCTR are the fastest and CCM is

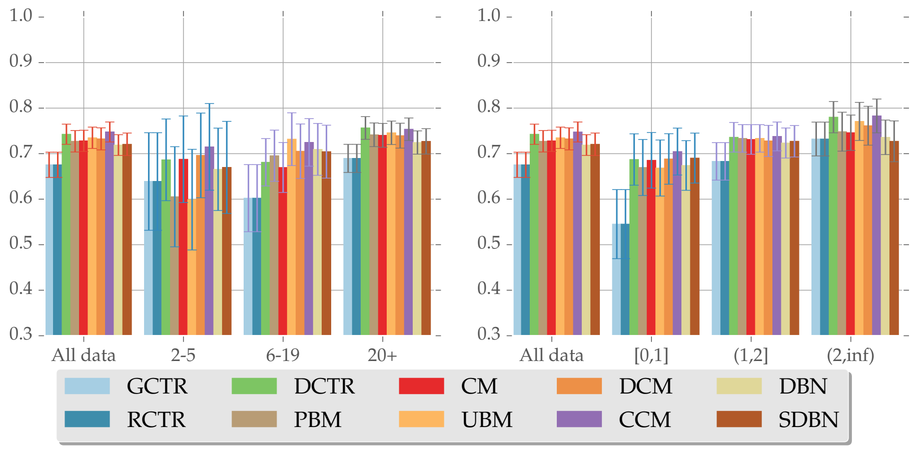


**Figure 2.5: Ranking performance (NDCG@5) of click models, grouped by query frequency (left) and click entropy (right).**

best for ranking documents.

Our results suggest that different click models can excel at some tasks while having inferior performance at others. Hence, when introducing a new click model or improving an existing one, it is important to keep in mind how it is going to be used. If a click model is going to be used for reranking, then the log-likelihood or the perplexity do not matter as much as the ability of the model to rerank documents, and if a click model is going to be used to understand user behavior, then the reranking performance is less important than its ability to explain observations as measured by log-likelihood and perplexity. It is not clear whether a single click model can be designed to cater for all needs. Potentially optimizing the design of a click model to a particular use case may improve performance.

We also showed that considering query frequency and click entropy increases the amount of information that can be gained from click model evaluation. In some of the cases our findings were counter intuitive, e.g., higher query frequency did not always make log-likelihood higher. Also, when ranking models by performance, different rankings are observed depending on query frequency or click entropy. This again suggests that no single model can beat all others and that one may benefit from either designing different models for different settings or using an ensemble of models.

The CTR prediction task seems to mimic the behavior of perplexity at the first rank and as such does not give any additional insights into model performance. Relevance prediction also does not give any new insights, albeit for a different reason, namely the presence of a large set of unseen document-query pairs when evaluating the models.

Our evaluation only covers some of the many click models that have been proposed. The potential for future work is great in the sense that the same evaluation approach can be applied to other click models.

We looked at clicks on the search engine result page and found that there is not a single model that performs best overall and different models have different merits.

**Table 2.4: Performance of click models according to various measures: log-likelihood ($\mathcal{LL}$), perplexity, RMSE of the CTR prediction task, AUC of the relevance prediction task, Pearson correlation between annotated relevances and predicted relevances, ranking performance (NDCG@5), and computation time. The symbol $^\blacktriangle$ denotes a significant difference at $p = 0.01$ as measured by a two tailed t-test.**

| Model | $\mathcal{LL}$ | Perplexity | RMSE | AUC | Pearson correlation | NDCG@5 | Time (sec.) |
|---|---|---|---|---|---|---|---|
| GCTR | -0.369 | 1.522 | 0.372 | 0.500 | 0.000 | 0.676 | 0.597 |
| RCTR | -0.296 | 1.365 | 0.268 | 0.500 | 0.000 | 0.676 | **0.589**$^\blacktriangle$ |
| DCTR | -0.300 | 1.359 | 0.261 | 0.535 | 0.054 | 0.743 | 3.255 |
| PBM | -0.267 | 1.320 | 0.354 | **0.581**$^\blacktriangle$ | 0.128 | 0.727 | 34.299 |
| CM | $\infty$ | 1.355 | 0.239 | 0.515 | 0.024 | 0.728 | 4.872 |
| UBM | **-0.249**$^\blacktriangle$ | 1.320 | 0.343 | **0.581**$^\blacktriangle$ | **0.130**$^\blacktriangle$ | 0.735 | 82.778 |
| DCM | -0.292 | 1.322 | **0.212**$^\blacktriangle$ | 0.516 | 0.035 | 0.733 | 5.965 |
| CCM | -0.279 | 1.341 | 0.283 | 0.541 | 0.106 | **0.748** | 521.103 |
| DBN | -0.259 | **1.318**$^\blacktriangle$ | 0.286 | 0.517 | 0.089 | 0.719 | 457.694 |
| SDBN | -0.290 | **1.318**$^\blacktriangle$ | **0.212**$^\blacktriangle$ | 0.529 | 0.076 | 0.721 | 3.916 |

However, modeling the position and presentation bias nearly always yields substantial performance gains. Interactions with other interactive systems also have bias, all of them have presentation bias – some actions were selected more often than others but often other biases are also present. For example, in recommender systems users are more likely to interact with the items they like and in dialogue systems some utterances are more likely to be produced than others. Modeling and taking into account these biases is crucial for successful optimization of interactive systems using interactions.

# 3 Bayesian Ranker Comparison Based on Historical User Interactions

In this chapter we address address **RQ2:** Does quantifying uncertainty in click models help to evaluate search engine rankers? In particular, we consider the case when we have a finite dataset of observed user interactions. This setting is practical for the task of evaluating and optimizing search engines. We focus on evaluation but the findings can be generalized to optimization as well – at each optimization step the algorithm has access to a finite dataset of observed user interactions and has to make decisions based on it. Understanding the evaluation task will shed light on how much information can be extracted from the dataset, what are the limitations and challenges. In order to quantify the amount of information we can extract from the logged interactions we take a Bayesian stance and explicitly model the confidence of our estimates. We address two tasks: evaluating a ranker and deciding if the logged interactions are enough to confidently decide that one version of the ranker is better than another one. These tasks are important both from practical and a theoretical perspective. From a practical perspective they are important because in practice one usually logs the interactions with the system and routinely has to decide if a new version of the system is better than the current one and how well they are performing. From a theoretical perspective it is important to understand how various biases in the dataset affect the confidence of the estimates and how important it is to take them into account during optimization.

We use web search as a particular instantiation of an interactive system and use the Dynamic Bayesian Network click model from the previous chapter as a model for interactions. In order to model the confidence of the estimates of the DBN click model we develop Bayesian inference for it. We also use the expected effort metric defined in [23] and develop a way to estimate it given a click log and reason about the confidence of this estimate. Using this machinery we can decide if one web search ranker is better than another one. As we said above, this setting is very common in industry and the findings can be generalized well beyond web search rankers to other systems that employ ranking such as music search, product search, recommender systems, retrieval based question answering and much more. All it requires is a model of interactions and an interaction based metric that decomposes into a combination of individual item contributions. Fortunately, often such metrics exist and models of interactions exist for

This chapter was published as [42].

well studied tasks or can be adapted from one task to another.

## 3.1 Introduction

Comparing rankers is an essential problem in information retrieval. In an industrial setting, an existing *production ranker* must often be compared to a new *candidate ranker* to determine whether to replace the former with the latter. Practical constraints make it difficult to make such decisions well. Because the performance of the candidate ranker is unknown, trying it out to gather data about its performance is often too risky: if it proves to be substantially inferior to the production ranker, the user experience may degrade, leading to abandonment. However, existing historical data (typically collected using the production ranker) may not be sufficiently informative about the candidate ranker. Hence, decisions about when to switch to the candidate ranker that are based on such data can be erroneous, also leading to a degraded user experience.

Controlling the risks associated with switching to a candidate ranker requires ranker comparison methods that can (1) estimate the performance of the production and candidate rankers using only historical data, and (2) quantify their uncertainty about such estimates. Quantification of uncertainty is essential for controlling risk because it enables system designers to switch to a candidate ranker only when they are highly confident that its performance will not be substantially worse than that of the production ranker.

Existing ranker evaluation methods do not fully meet these requirements. Traditional methods rely on explicit relevance labels provided by human assessors to measure metrics such as Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) [119]. However, such labels are expensive to obtain and may not reflect the preferences of real users. Click-based approaches overcome these limitations by estimating performance from the implicit signals in users' click behavior, using A/B testing or interleaving experiments [21, 22, 50, 54, 65, 71, 113, 115, 120]. Nonetheless, these methods typically require trying out the candidate ranker to gain data about it, which, as mentioned above, may be too risky. While interleaving methods have been developed that use importance sampling to estimate ranker performance given only historical data [55], they have limited generalization properties and require historical data to be gathered stochastically.

Click models [19, 43], i.e., probabilistic models of user behavior, can also estimate ranker performance given only historical data [23] and do not share these limitations. The uncertainty in the resulting estimates depends on the relationship between the historical data and the rankers to be evaluated. E.g., if a candidate ranker differs from the production ranker only in that it swaps a lowly ranked but relevant document with a highly ranked but irrelevant one, then the click model can confidently conclude that the candidate ranker is better so long as the historical data shows which of the two documents is relevant. It can do so even if the result lists containing such a swap do not appear in the historical data. By contrast, if the candidate ranker gives a high rank to a document that does not appear in the historical data, then its relevance cannot be estimated with confidence. A key limitation of existing click models is that they do not distinguish between cases such as these. Because they do not quantify their uncertainty about the comparisons they perform, they cannot judge whether the information in the logs is sufficient to compare rankers. Only methods that can measure the confidence of the performed comparisons can be used to safely decide whether to switch from the production ranker to a candidate one.

We present *Bayesian Ranker Comparison* (BARACO), a click model-based approach to ranker evaluation that compares the performance of ranker pairs using only historical data and quantifies the uncertainty in such comparisons. The key novelty lies in maintaining a full posterior distribution over the relevance of documents for queries. This posterior is used to estimate the probability that the candidate ranker's performance is not substantially worse than that of the production ranker. By switching to the candidate ranker only when this probability is sufficiently high, the risks associated with switching can be controlled in a principled way.

BARACO is able to estimate not only the probability of a candidate ranker being as good as the production ranker, but also the expected difference in performance between two rankers. Measuring the expected difference can be useful in practice, e.g., to select a single ranker from a set of candidates, all of which have a high probability of beating the production ranker.

We present the results of an empirical evaluation on several learning to rank datasets and on a real click log published by Yandex that compares BARACO to an existing non-Bayesian click model-based ranker evaluation method that estimates only the expected difference in performance between ranker pairs, without quantifying uncertainty.

We address the following research questions:

**RQ2.1** Can BARACO determine whether the production ranker should be replaced with a candidate ranker better than the non-Bayesian baseline?

**RQ2.2** Does BARACO produce better estimates of the difference in performance of the rankers than the non-Bayesian baseline?

Our results show that BARACO has better performance than the baseline method and can identify more good rankers in a set of candidates more reliably. In addition, the full Bayesian approach enables the production of more precise estimates of the expected differences between rankers than the baseline method.

## 3.2 Related Work

Ranker evaluation has long been a central topic in information retrieval. The classical approach is to perform offline evaluation based on the Cranfield paradigm [26, 119], where a collection of documents is manually annotated by human experts. A representative set of queries together with associated user intents are crafted by a group of experts. Then, for each query, documents in the collection are assigned relevance labels. The documents and their relevance labels can then be used in metrics such as NDCG. This approach is widely used, well understood, and benefits from controlled laboratory settings. But it is expensive and assessors' relevance judgements may not adequately reflect real users' opinions.

Online evaluation is a family of techniques that addresses these difficulties by letting users be the judges. In A/B testing [71], the user population is split into two groups, and pairs of rankers are compared by presenting one group with one ranker and the other group with another ranker. The ranker with the best performance on a selected metric (such as CTR) is typically considered to be the winner [71]. Another approach is interleaved comparison: search engine result pages (SERPs) presented to users are obtained by interleaving SERPs of two competing rankers under consideration. The user feedback in the form of clicks is then interpreted as user preference for one ranker over the other [115]. The key problem of online methods is that they require user feedback to evaluate each pair of rankers, which often requires exposing suboptimal SERPs. Consequently, they may be too risky for many real-world settings.

Interleaving methods that exploit importance sampling [55] provide a way to compare rankers using only historical click data. However, importance sampling requires that the source distribution is non-zero everywhere where the target distribution is non-zero (i.e., every SERP that can result from interleaving has a non-zero probability of occurring in the historical data), a requirement that is often not met in practice. Importance sampling guarantees only that the resulting estimator is unbiased, not that it has low variance: the addition of more historical data is not guaranteed to improve estimates, as new data can actually increase the variance [9]. Additionally, importance sampling has poor generalization properties compared to click-based approaches that infer relevance labels of documents. In particular, the latter can generalize across SERPs that differ in the order of documents, while the former cannot.

In recent years, a number of probabilistic models have been developed to describe, understand, and predict user behavior while interacting with an IR system. In particular, click models such as the Dynamic Bayesian Network (DBN) [19], the Dependent Click Model (DCM) [43] and the User Browsing Model (UBM) [35] infer the relevance of documents for queries and model user clicks on documents in the SERP by analyzing search engine log files. These inferred relevance labels can be further used for learning to rank [19] and for ranker comparisons using click model-based metrics [23]. However, to our knowledge, none of these approaches provide a way to quantify the uncertainty in the resulting comparison, which is critical for making informed decisions about when to switch from a production ranker to a candidate ranker. For metrics such as EBU [151], ERR [20] and the utility and effort based metrics in [17, 23], the effect on the comparison of a document seen just once and that of one seen a hundred of times is the same, given that the inferred relevance labels have the same value. Furthermore, these metrics do not take into account the number of previously unseen documents that appear in the SERPs produced by the candidate rankers. Hence, these approaches cannot properly control the risks associated with switching to a candidate ranker, as the uncertainty in the comparison is not measured.

We present a new approach to ranker evaluation that is designed to overcome these limitations. It compares rankers using only the log files collected through natural user interactions with the production ranker. In contrast to importance sampling based interleaving methods [55], our method does not require the data to be obtained stochastically. In contrast to the metrics in [17, 20, 23, 151], our method takes into account the uncertainty associated with the inferred relevance labels and the presence of unseen documents and measures the confidence of the resulting comparison.

## 3.3 Problem Setting

We consider two related problems. The first is the *switching problem*: deciding whether or not to switch from a production ranker $R_p$ to a candidate ranker $R_c$. The second is the *difference estimation problem*: estimating the expected difference in performance between the production ranker $R_p$ and candidate ranker $R_c$.

The switching problem is as follows. System designers must decide whether or not to switch from a production ranker $R_p$ to a candidate ranker $R_c$. They are willing to do so only if they are highly confident that the candidate ranker is at least almost as good as the production ranker, i.e., if

$$p(M_c + \epsilon \geq M_p) \geq 1 - \delta, \tag{3.1}$$

where $M_p$ and $M_c$ are the expected performance of the production and candidate ranker, respectively, according to some metric; $\epsilon$ is the degree to which the candidate ranker is allowed to be worse than the production ranker; and $\delta$ is the probability with which the candidate ranker is allowed to fall outside this threshold. The goal of a ranker comparison method in the switching problem is to determine whether (3.1) holds. Note that this requires explicitly reasoning about the uncertainty of comparisons between $R_p$ and $R_c$.

By contrast, the goal of a ranker comparison method in the difference estimation problem is to accurately estimate the expected difference of the metric values for the two rankers:

$$\mathbb{E}[M_c - M_p]. \tag{3.2}$$

Unlike the switching problem, the difference estimation problem does not require explicitly reasoning about uncertainty. Nonetheless, it can be useful for, e.g., selecting a single ranker from a set of candidates, all of which satisfy (3.1).

For both problems, the ranker comparison method is given only a log $L = [l_1, \ldots, l_{|L|}]$ of user interaction sessions gathered using $R_p$. Each session $l_j$ consists of the following: $q$, the query the user has submitted; $[d_0, \ldots, d_N]$, the list of $N$ documents returned by $R_p$ that make up the SERP; and $[c_0, \ldots, c_N]$, the clicks produced by the user on those documents.

## 3.4 Click Model-Based Metric

In this section, we describe the metric used to define $M_p$ and $M_c$ within BARACO, the Bayesian ranker comparison method we introduce in Section 3.5. While BARACO can work with any click model that provides relevance estimates, our implementation uses DBN [19] because it has shown high performance in predicting user clicks [23].

DBN, the Dynamic Bayesian Network click model [19], represents the relevance of a document for a query as a combination of two variables: *attractiveness* and *satisfactoriness*. In DBN, each user interaction session $l_j$ begins with a query $q$ from which the SERP, consisting of titles and snippets of the documents $[d_{q,0}, \ldots, d_{q,N}]$, is generated. By assumption, the user starts by examining the first document's title and snippet. If they find the document *attractive*, they click on and examine the document, or otherwise proceed to examine the following documents' titles and snippets in order. If a clicked document *satisfies* the user, they terminate the session. Otherwise, they may or may not continue to examine the SERP. DBN assumes the user will not click a document prior to examining its snippet and cannot be satisfied by a document before clicking on it.

Specifically, for a query $q$ the document at position $i$ in the SERP is modelled with four binary variables: whether it is

- attractive ($A_{q,i}$),
- satisfactory ($S_{q,i}$),
- examined ($E_{q,i}$), and
- clicked ($C_{q,i}$).

The following relationships hold between variables for a given query:

- $E_{q,0} = 1$: the first document is examined,
- $A_{q,i} = 1, E_{q,i} = 1 \Leftrightarrow C_{q,i} = 1$: a document is clicked if, and only if, it is attractive and examined,
- $C_{q,i} = 0 \implies S_{q,i} = 0$: only clicked documents can be satisfactory,
- $S_{q,i} = 1 \implies E_{q,i+1} = 0$: satisfied users abandon the search, and
- $E_{q,i} = 0 \implies E_{q,i+1} = 0$: users do not skip documents.

In addition, the following stochastic relationships hold:

- $p(E_{q,i+1} = 1 | E_{q,i} = 1, S_{q,i} = 0) = \gamma$: users continue to examine the list after not being satisfied with probability $\gamma$,
- $p(A_{q,i} = 1) = a_{q,i}$, and
- $p(S_{q,i} = 1 | C_{q,i} = 1) = s_{q,i}$.

Thus, attractiveness and satisfactoriness are governed by stationary Bernoulli distributions with unknown parameters $a_{q,i}$ and $s_{q,i}$, which must be inferred from clicks, the only observed variables. As in [19, 23], we assume $\gamma$ is fixed and known. Figure 3.1 depicts the relationships between all variables in a given session.

Once $a_{q,i}$ and $s_{q,i}$ have been inferred, they can be used to compute a metric such as EBU [151], ERR [20], and the utility and effort-based metrics described in [17, 23]. In this chapter, we use the expected effort metric defined in [23]. For a query $q$ that yields a document list with $N$ documents, the metric is defined as:

$$rrMetric(q) = \sum_{i=1}^{N} \left( \frac{s_{q,i} a_{q,i}}{i} \prod_{j=1}^{i-1} (1 - s_{q,j} a_{q,j}) \right). \tag{3.3}$$

For a given ranker $R_x$, the metric $M_x$ required by (3.1) can then be defined as the expected value of the $rrMetric(q)$ across queries:

$$M_x = \sum_{q \in Q} rrMetric(q) p(q). \tag{3.4}$$

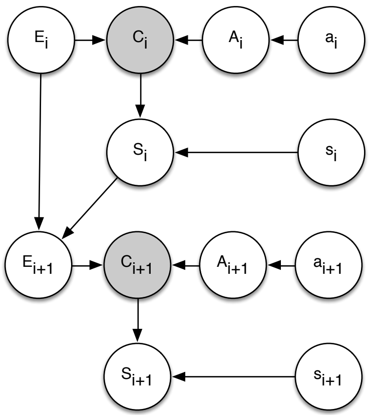


**Figure 3.1: Graphical representation of the DBN click model. The grey circles represent the observed variables.**

Comparing the production ranker to a candidate ranker is complicated by the presence of unseen documents in the candidate rankings. In order to compute the values of a metric for rankings with unseen documents, the unseen documents can be either assigned some a priori values of attractiveness and satisfactoriness or excluded from the ranking. The latter strategy is taken in [23] and in the baseline method used in our experiments.

Using an approach based on Expectation Maximization (EM), a maximum a posteriori estimate of $M_x$ can be computed [19]. Hence, the difference estimation problem can be addressed by simply estimating $M_c$ and $M_p$ using this EM-based method and then computing the difference between them. However, this approach suffers from a key limitation. Because it is based only on maximum a posteriori estimates, the difference computed by such an EM-based approach is not the true expected difference of the metric values for the two rankers. Instead, it is only the first mode of the posterior distribution of the difference of the metric values, which may not be equal to the expectation if the distribution is multimodal or asymmetric. Because BARACO is a fully Bayesian approach, it can estimate the true expected value of the difference between $M_c$ and $M_p$, which leads to better quality estimates, as we will see.

Furthermore, the switching problem cannot be directly addressed using an EM-based approach because such an approach gives no information about the uncertainty in the resulting estimate. That is, it does not estimate the *distribution* over the metric values of the two rankers. Consequently, it provides no way to estimate the probability that $M_c + \epsilon$ is greater than or equal to $M_p$. Instead, addressing the switching problem using an EM-based method requires resorting to a heuristic approach, e.g., switching only when the estimated difference exceeds some manually tuned threshold. This forms the core of the baseline method we compare against in Section 3.7.

Below, we present a method that addresses the shortcomings of an EM-based approach and enables better solutions to both the switching problem and the difference estimation problem.

## 3.5 BARACO

In this section, we present BARACO. First, Section 3.5.1 describes how the posterior distributions over $a_{q,i}$ and $s_{q,i}$ can be inferred from $L$, the set of user interaction sessions; Section 3.5.2 describes how to solve the switching problem by evaluating (3.1) when $M_p$ and $M_c$ are defined according to (3.4), given posterior distributions over $a_{q,i}$ and $s_{q,i}$; Section 3.5.3 describes how to solve the difference estimation problem.

### 3.5.1 Inferring click model posteriors

Evaluating (3.1) and (3.2) requires knowing the posterior probabilities $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$ for each ranker, query, and document. In this subsection, we describe how to estimate these posteriors.

The algorithm works by iterating over the sessions. To process the first session, the posteriors $p(a_{q,i}|l_1)$ and $p(s_{q,i}|l_1)$ are computed given uniform priors $p(a_{q,i})$ and $p(s_{q,i})$. Then, for each subsequent session $l_j$, $p(a_{q,i}|L_j)$ and $p(s_{q,i}|L_j)$ are computed given $p(a_{q,i}|L_{j-1})$ and $p(s_{q,i}|L_{j-1})$, where $L_j = [l_1, \ldots, l_j]$. The algorithm terminates when $l = |L|$, yielding $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$.

We now describe how to compute $p(a_{q,i}|L_j)$ and $p(s_{q,i}|L_j)$ given $p(a_{q,i}|L_{j-1})$ and $p(s_{q,i}|L_{j-1})$ and the next session $l_j$. Because $A_{q,i}$ and $S_{q,i}$ are Bernoulli variables, we can model the distribution over their parameters $a_{q,i}$ and $s_{q,i}$ using a Beta distribution. Focusing on $a_{q,i}$, this yields:

$$\begin{aligned} p(a_{q,i}|L_j) &= Beta(\alpha, \beta) \\ &= \frac{{a_{q,i}}^{\alpha-1}(1-a_{q,i})^{\beta-1}}{B(\alpha,\beta)}, \end{aligned} \tag{3.5}$$

where $\alpha$ is the number of observations of $A_{q,i} = 1$ and $\beta$ is the number of observations of $A_{q,i} = 0$ that have occurred up to and including session $j$, and $B(\alpha, \beta)$ is a Beta function. $Beta(1, 1)$ corresponds to a uniform distribution, i.e., no prior knowledge about $a_{q,i}$. If $A_{q,i}$ is observed in session $l_j$, then $p(a_{q,i}|L_j)$ can be updated using Bayes' rule:

$$\begin{aligned} p(a_{q,i}|L_j) &= p(a_{q,i}|A_{q,i}, L_{j-1}) \\ &= \frac{p(A_{q,i}|a_{q,i})p(a_{q,i}|L_{j-1})}{p(A_{q,i}|L_{j-1})}, \end{aligned} \tag{3.6}$$

In this case, the update reduces to simply incrementing $\alpha$ or $\beta$. Since the Beta distribution is the conjugate prior for the Bernoulli variable, the posterior remains a Beta distribution.

In our setting, $A_{q,i}$ is not directly observed. However, when we know $E_{q,i}$, we can directly infer $A_{q,i}$ from $C_{q,i}$ because the user always clicks on an attractive examined document. Thus, the difficult case is when we do not know $E_{q,i}$, which occurs whenever $i > c$, where $c$ is the index of the last clicked document. The remainder of this subsection describes how to address this case.

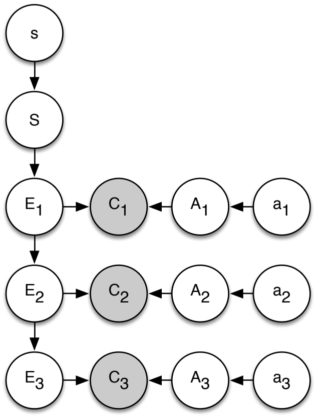


**Figure 3.2: Graphical representation of the DBN click model for the part of the SERP below and including the last clicked document in a session. Grey circles represent observed variables.**

There are two steps. Since we do not know $E_{q,i}$ when $i > c$, in the first step we must instead compute a posterior over it: $p(E_{q,i}|L_j)$. Then, in the second step, we use $p(E_{q,i}|L_j)$ to estimate $p(a_{q,i}|L_j)$ and $p(s_{q,i}|L_j)$.

To perform the first step, we use the *sum-product* message passing algorithm [9]. In particular, we extract the subgraph of the graphical model that represents the documents below the last clicked one and remove the nodes representing the satisfactoriness ($S_{q,i}$ and $s_{q,i}$) for documents without clicks. This is because it is not possible to say anything about $s_{q,i}$ for $i > c$ as it is not observed and has no effect on what is observed. Since the resulting graph, shown in Figure 3.2, is a polytree, the sum-product algorithm enables us to perform exact inference, which yields $p(E_{q,i}|L_j)$.

For the second step, we compute $p(a_{q,i}|L_j)$ and $p(s_{q,i}|L_j)$ given $p(E_{q,i}|L_j)$. Focusing now on $p(a_{q,i}|L_j)$ and starting from Bayes' rule, we have:

$$\begin{aligned} p(a_{q,i}|L_j) &= p(a_{q,i}|C_{q,i}, L_{j-1}) \\ &= \frac{p(C_{q,i}|a_{q,i})p(a_{q,i}|L_{j-1})}{p(C_{q,i}|L_{j-1})}. \end{aligned} \tag{3.7}$$

The likelihood term $p(C_{q,i}|a_{q,i})$ can be computed by marginalizing across $A$ and $E$:

$$\begin{aligned} p(C_{q,i}|a_{q,i}) = \sum_{A\in\{0,1\}} \sum_{E\in\{0,1\}} \Big( & p(C_{q,i}|E_{q,i} = E, A_{q,i} = A) \\ & p(E_{q,i} = E)p(A_{q,i} = A|a_{q,i})\Big). \end{aligned} \tag{3.8}$$

Sticking (3.8) into (3.7) and ignoring normalization gives:

$$\begin{aligned}
p(a_{q,i}|C_{q,i}, L_{j-1}) \quad &\propto \sum_{A\in\{0,1\}} \sum_{E\in\{0,1\}} \Big(p(C_{q,i}|E_{q,i}=E, A_{q,i}=A) \\
&\qquad p(E_{q,i}=E)p(A_{q,i}=A|a_{q,i})p(a_{q,i}|L_{j-1})\Big) \\
&\propto p(a_{q,i}|L_{j-1}) \sum_{A\in\{0,1\}} \Big(p(A_{q,i}=A|a_{q,i}) \\
&\qquad \big(p(C_{q,i}|E_{q,i}=0, A_{q,i}=A)p(E=0)+ \\
&\qquad + p(C_{q,i}|E_{q,i}=1, A_{q,i}=A)p(E_{q,i})\big)\Big).
\end{aligned}$$

Since we are focusing on the case where $i > c$, we know that $C_{q,i} = 0$, i.e., $d_{q,i}$ was not clicked. Furthermore, we know that $p(C_{q,i} = 0|E_{q,i} = 0, A_{q,i} = A) = 1$ and $p(C_{q,i} = 0|E_{q,i} = 1, A_{q,i} = A) = 1 - A$, yielding:

$$\begin{aligned}
p(a_{q,i}|C_{q,i}=0, L_{j-1}) &\propto p(a_{q,i}|L_{j-1}) \sum_{A\in\{0,1\}} \Big(p(A_{q,i}=A|a_{q,i}) \\
&\qquad (1\cdot p(E_{q,i}=0)+(1-A_{q,i})\cdot p(E_{q,i}=1))\Big) \\
&\propto p(a_{q,i}|L_{j-1}) \sum_{A\in\{0,1\}} \Big(p(A_{q,i}=A|a_{q,i}) \\
&\qquad (1-p(E_{q,i}=1)+p(E_{q,i}=1)-A\cdot p(E_{q,i}=1))\Big) \\
&\propto p(a_{q,i}|L_{j-1}) \sum_{A\in\{0,1\}} \Big(p(A_{q,i}=A|a_{q,i}) \\
&\qquad (1-A\cdot p(E_{q,i}=1))\Big).
\end{aligned} \tag{3.9}$$

Now we can substitute $p(A_{q,i} = A|a_{q,i})$ for the values of $A$: $p(A_{q,i} = 1|a_{q,i}) = a_{q,i}$ and $p(A_{q,i} = 0|a_{q,i}) = 1 - a_{q,i}$ yielding:

$$p(a_{q,i}|C_{q,i} = 0, L_{j-1}) \propto p(a_{q,i}|L_{j-1})(1 - a_{q,i} \cdot p(E_{q,i} = 1)). \tag{3.10}$$

Thus, (3.10) is the Bayesian update rule for attractiveness. When $p(E_{q,i} = 1) = 1$, e.g., if the document was clicked, the posterior is a Beta distribution because plugging $p(E_{q,i} = 1) = 1$ in (3.10) yields a term conjugate to the parametrization in (3.5). Otherwise, we use $p(E_{q,i}|L_j)$, as computed in the first step via a message-passing algorithm that takes into account the satisfactoriness of the last clicked document and the attractivenesses of the unexamined documents. Instead of a Beta distribution, this yields a more general polynomial function. The normalization constant can be found by integration: since the function is a polynomial with known coefficients, it can be integrated analytically and then the definite integral evaluated on the interval $[0, 1]$. The

**Algorithm 1** ComputePosteriors($L$)

```
1: InitializePriors()
2: for $l_j$ in $L$ do
3:     for $i$ in $l_j$ do
4:         $p(E_{q,i}) \leftarrow sumProduct(l_j)$
5:         if $C_{q,i} = 1$ then
6:             $p(a_{q,i}|C_{q,i} = 1, L_{j-1}) \leftarrow p(a_{q,i}|L_{j-1})a_{q,i}$
7:             $p(s_{q,i}|C_{q,i+1} = 0, L_{j-1}) \leftarrow p(s_{q,i}|L_{j-1})$
                   $(p(E_{q,i+1} = 1) + s_{q,i}p(E_{q,i+1} = 0))$
8:         else
9:             $p(a_{q,i}|C_{q,i} = 0, L_{j-1}) \leftarrow p(a_{q,i}|L_{j-1})$
                   $(1 - a_{q,i} \cdot p(E_{q,i} = 1))$
   return $p(a|L), p(s|L)$
```

cumulative distribution function is therefore also a polynomial and easy to compute. The same holds for the expectation of the distribution.

Analogously, we can derive an update rule for satisfactoriness:

$$\begin{aligned} p(s_{q,i}|C_{q,i+1} = 0, L_{j-1}) \propto \\ p(s_{q,i}|L_{j-1})(p(E_{q,i+1} = 1) + s_{q,i}p(E_{q,i+1} = 0)). \end{aligned} \tag{3.11}$$

Algorithm 1 summarizes the steps involved in computing $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$. First, the attractiveness and satisfactoriness of all document query pairs are assigned uniform $Beta(1, 1)$ priors. Then, the log file is processed session by session and the attractiveness and satisfactoriness of all document query pairs in the session are updated using the Bayesian update rules described in this section: (3.6) and (3.10) for attractiveness and (3.11) for satisfactoriness.

## 3.5.2 The switching problem

To decide whether or not to switch from $R_p$ to $R_c$, we must determine whether (3.1) holds. Determining this from $L$ requires evaluating the following double integral:

$$\begin{aligned} p(M_c + \epsilon \geq M_p \mid L) = \\ \int_{M_c=0}^{M_c=M^*} p(M_c|L) \int_{M_p=0}^{M_p=M_c+\epsilon} p(M_p|L)\, dM_c\, dM_p, \end{aligned} \tag{3.12}$$

where $M^*$ is the maximum possible value of the metric defined in (3.4). Evaluating this integral requires knowing the posterior probabilities $p(M_p|L)$ and $p(M_c|L)$, which can be computed from (3.3) and (3.4) given $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$ for each ranker, query, and document.

Computing (3.12) analytically is hard, but it can be estimated using a Monte-Carlo sampling scheme. Algorithm 2 describes this scheme, which yields a version of BARACO that solves the Switching Problem.

**Algorithm 2** BARACO-SP$(R_p, R_c, L, \epsilon, \delta)$

```
1: $p(a|L), p(s|L) \leftarrow computePosteriors(L)$                ▷ Section 3.5.1
2: for $k$ in 1:$K$ do                                              ▷ Section 3.5.2
3:     $M_p^k \leftarrow sample(R_p, p(a|L), p(s|L))$
4:     $M_c^k \leftarrow sample(R_c, p(a|L), p(s|L))$
5:     if $M_c^k + \epsilon \geq M_p^k$ then
6:         $N_{M_c+\epsilon \geq M_p} \leftarrow N_{M_c+\epsilon \geq M_p} + 1$
7: $p(M_c + \epsilon \geq M_p) \leftarrow \frac{N_{M_c+\epsilon \geq M_p}}{K}$
   return $p(M_c + \epsilon \geq M_p) \geq 1 - \delta$
```

First, we compute the posteriors $p(a|L)$ and $p(s|L)$ using the approach described in Section 3.5.1 (line 1). Then, each sampling iteration $k$ consists of drawing a sample $a_{q,i}$ and $s_{q,i}$ for each ranker, document, and query from $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$. Using these samples, we compute $M_c^k$ and $M_p^k$, the values of the metrics given the sampled probabilities from the $k$-th iteration (lines 3–4). Estimating (3.12) then reduces to computing the fraction of sampling iterations for which $M_c^k + \epsilon \geq M_p^k$ (lines 6–7). If this fraction is greater than $1 - \delta$, we can safely switch to the candidate ranker.

However, sampling $a_{q,i}$ and $s_{q,i}$ from $p(a_{q,i}|L)$ and $p(s_{q,i}|L)$ is itself hard because their cumulative distribution functions are high degree polynomials that are hard to invert. Therefore, we employ the Metropolis-Hastings algorithm [9] with the expected value of the sampled distribution, which can be calculated analytically through integration, as the starting position of the random walk. The proposal distribution is a Gaussian with fixed variance.

### 3.5.3 The difference estimation problem

If we are interested only in the expected value of a ranker $R_x$, and not the uncertainty of this estimate, we can compute it as follows:

$$\mathbb{E}[M_x] = \int_{M_x=0}^{M_x=M^*} M_x p(M_x)\, dM_x.$$

Using the same sampling procedure described above, we can solve the difference estimation problem by estimating this expected value:

$$\mathbb{E}[M_x] \approx \frac{1}{K} \sum_{k=1}^{K} M_x^k, \tag{3.13}$$

where $K$ is the number of sampling iterations. We can similarly estimate the expected difference between $M_c$ and $M_p$:

$$\mathbb{E}[M_c - M_p] \approx \frac{1}{K} \sum_{k=1}^{K} (M_c^k - M_p^k). \tag{3.14}$$

Algorithm 3 summarizes the resulting version of BARACO that solves the difference estimation Problem. It is essentially the same as Algorithm 2 except that it estimates the expected difference between the production and the candidate rankers' metrics.

**Algorithm 3** BARACO-DEP$(R_p, R_c, L, \epsilon, \delta)$

```
1: $p(a|L), p(s|L) \leftarrow computePosteriors(L)$    ▷ Section 3.5.1
2: $\sum_{\Delta_M} \leftarrow 0$
3: for $k$ in 1:$K$ do    ▷ Section 3.5.3
4:     $M_p^k \leftarrow sample(R_p, p(a|L), p(s|L))$
5:     $M_c^k \leftarrow sample(R_c, p(a|L), p(s|L))$
6:     $\sum_{\Delta_M} \leftarrow \sum_{\Delta_M} + M_c^k - M_p^k$
7: $\mathbb{E}[M_x] \leftarrow \frac{\sum_{\Delta_M}}{K}$
   return $\mathbb{E}[M_x]$
```

## 3.6 Experimental setup

In this section, we describe the experimental setup used to evaluate BARACO. This evaluation is complicated by the fact that the user's information need, and therefore the true relevance labels, are unknown. We present two evaluations which deal with this problem in different ways. The first evaluation, based on the LETOR datasets [112], uses manual relevance assessments as ground-truth labels and synthetic clicks as feedback to BARACO. The second evaluation, based on the WSDM 2014 Web search personalization challenge,[1] uses dwell time as ground-truth labels and real clicks as feedback to BARACO.

Another possibility would be to evaluate BARACO using explicit relevance labels provided by human assessors to measure metrics such as Normalized Discounted Cumulative Gain (NDCG) but this approach is known to have low agreement with metrics based user behavior such as A/B testing or interleaving [20, 22, 115, 152], so it is natural to expect that it would have a low agreement with BARACO as well. It would also be possible to evaluate BARACO using A/B tests or interleavings, but A/B tests have low agreement with interleavings and different metrics collected during A/B tests such as click through rate, clicks@1 and others have low agreement with each other [115]. The only definitive way to evaluate BARACO would be in an industrial setting that measures long-term metrics such as engagement. Such results, however, would be difficult to reproduce. Consequently, the LETOR evaluation is the most reliable and reproducible, as it depends on indisputably unbiased ground truth. In addition, it allows us to explore how the discrepancy between the click model and user behavior affects BARACO's performance. However, the WSDM evaluation, though less reliable, is nonetheless useful because it gives insight into how BARACO performs in a real-world setting with real users.

We compare BARACO to a baseline that, in lieu of our Bayesian approach, uses the EM-based approach described in [19] to compute maximum a posteriori estimates of (3.4) for $M_p$ and $M_c$. These estimates can then be directly used in the difference estimation problem. Because this approach does not compute full posteriors, it does not quantify uncertainty in the resulting estimates and therefore cannot be directly

[1]Personalized Web Search Challenge 2014, `https://www.kaggle.com/c/yandex-personalized-web-search-challenge`

used in the switching problem. Instead, the baseline method, which we call Manual Thresholding (MT), resorts to a heuristic approach to determine whether (3.1) holds. In particular, $R_c$ is deemed safe when $\hat{M}_c - \hat{M}_p > \epsilon_m$, where $\hat{M}_c$ and $\hat{M}_p$ are the maximum a posteriori estimates of $M_c$ and $M_p$ produced by the EM-based method and $\epsilon_m$ is a threshold parameter whose value must be tuned manually. Because we want to switch whenever $M_c - M_p > -\epsilon$, the quantity $\epsilon + \epsilon_m$ acts as a buffer, i.e., an extra gap that $R_c$ must exceed to be considered safe. Adding this buffer heuristically accounts for the uncertainty in $\hat{M}_c$ and $\hat{M}_p$.

The need to tune $\epsilon_m$ for MT poses significant difficulties in practice. To understand the effect of $\epsilon_m$ on the behavior of MT would require access to ground truth about a set of candidate rankers, i.e., whether they are in fact no more than $\epsilon$ worse than the production ranker. While such ground truth could be obtained using off-line or on-line evaluations, such evaluations pose exactly the difficulties that motivate the need for methods like BARACO: the former is expensive and may not reflect real user preferences. The latter requires showing potentially poor rankers to real users. Furthermore, while such ground truth, even if it could be obtained, would shed light on how $\epsilon_m$ affects to which candidate rankers MT switches, it would still not make it possible to select the $\epsilon_m$ that is best for a given $\delta$ supplied by the system designers. Doing so would require a quantification of the uncertainty about each candidate ranker that is inherent to BARACO but absent in MT.

Importantly, BARACO does *not* require tuning $\epsilon_m$ or any analogous parameter. On the contrary, $\delta$ and $\epsilon$, which are supplied by the system designers, are simply quantifications of their risk tolerance.

### 3.6.1 LETOR evaluation

The first evaluation is based on the LETOR datasets [112], which include manual relevance assessments. However, they do not include user clicks. In addition, even if they did, these would likely correlate only poorly with the relevance assessments, since the assessors may not interpret the users' information need correctly. To address this difficulty, we instead axiomatically define the relevance assessments to be correct ground-truth labels and use click models to generate the clicks. Since BARACO also relies on click models, we evaluate it in settings where the clicks are generated and interpreted using different click models, to assess BARACO's robustness to errors in its modeling assumptions.

LETOR is split into six sub-datasets: HP2003, HP2004, NP2003, NP2004, TD2003, and TD2004. For each run of each algorithm, we use the data from one sub-dataset. First we train a ranker $R_{Ada}$ using AdaRank [149] on all the data in this sub-dataset. AdaRank, which performs reasonably on all these datasets [149], trains a linear ranking function, i.e., a linear combination of the ranking features for a given query-document pair. The documents are then sorted based on the values produced by this ranking function.

To ensure that some candidate rankers will be better than the production ranker, we craft $R_p$ by "damaging" $R_{Ada}$, i.e., randomly by adding random vectors to it. These vectors were generated by randomly sampling a normal distribution with mean equal to 0 and standard deviation of 0.2. Finally, to generate a population of candidate rankers $R_c$, we again perturb $R_p$ 1,000 times using the same sampling method. This ranker generation methodology is motivated by the gradient descent algorithm [154]; the standard deviation value was chosen so that some of the rankers would be similar enough to the production ranker and some too different from it for the algorithm to be confident about their performance.

The next step is to generate the log file. To this end, we *generate* user click interaction data using the DBN, SDBN and UBM click models with three user model settings: perfect, navigational and informational. The clicks are then *interpreted* using the DBN click model. The parameters of the click models are summarized in Table 3.1, where $p(C|R)$ and $p(C|NR)$ denote the probability of a user clicking a relevant document and an irrelevant document, respectively, and $p(s|R)$ and $p(s|NR)$ denote the probability of abandoning the SERP after clicking a relevant document and an irrelevant document, respectively. The closer $p(C|R)$ is to $p(C|NR)$ and $p(s|R)$ to $p(s|NR)$, the more noise there is in the feedback and the more difficult inference becomes. The user interaction data is generated for the production ranker by randomly sampling 500 queries and generating the SERPs and clicks.

**Table 3.1: Overview of the user model settings.**

| Model | $p(C\|R)$ | $p(C\|NR)$ | $p(s\|R)$ | $p(s\|NR)$ |
|---|---|---|---|---|
| Perfect | 1.0 | 0.0 | 0.0 | 0.0 |
| Navigational | 0.95 | 0.05 | 0.9 | 0.2 |
| Informational | 0.9 | 0.4 | 0.5 | 0.1 |

The perfect user model setting is used to obtain an upper bound in performance: the user clicks all relevant documents and no irrelevant ones. The navigational and informational user models are based on typical user behavior in web search [44]. The navigational user model reflects user behavior while searching for an item they know to exist, such as a company's homepage. Because it is easy for users to distinguish between relevant and irrelevant documents, the noise levels are low. The informational user model reflects user behavior while looking for information about a topic, which can be distributed over several pages. Because this task is more difficult, there is more noise in the feedback.

The clicks are generated and interpreted using either the same or different click models. When they are generated and interpreted using the same click model, our findings are not affected by the accuracy of the assumptions in the click model, allowing us to focus on the differences between BARACO and the baseline method. Of course, some assumptions made by DBN, which is used to interpret clicks, may not always hold in practice. For example, DBN assumes that the document that was clicked and led to abandonment is the document that satisfied the user. This assumption typically holds for navigational queries, but may not be valid for informational queries. Therefore, experiments in which the clicks are generated and interpreted using different click models help measure the robustness of BARACO to settings whether the assumptions underlying DBN do not always hold.

In the LETOR evaluation setup, we compare the performance of BARACO and MT using Area Under the ROC Curve (AUC), Pearson correlation, and the square root of the mean squared error (RMSE). The rankers are compared using the metric $rrMetric$ (3.3).

## 3.6.2 WSDM evaluation

We additionally evaluate BARACO using an anonymized click log released by Yandex for the WSDM 2014 Web search personalization challenge. The click log contains sessions with queries submitted by a user. For each query there is a list of search results returned by the engine and the clicks produced by the user. The queries and the clicks have timestamps. However, the units of time are not disclosed. The queries, query terms, documents, and users are represented by IDs in order to protect the privacy of users.

The organizers of the challenge have defined three levels of relevance based on the clicks that the documents receive: documents that receive clicks with dwell time less than 50 time units have relevance 0, clicks with dwell time between 50 and 150 units have relevance 1, and clicks with dwell time of more than 2 time units as well as clicks that are the last clicks for a given query have relevance 2. We use all but the last session for a query for training and use the last session for extracting the relevance labels for query-document pairs.

The candidate rankers are generated by perturbing the result lists observed in the click log in a random way. In order to ensure that some of the candidates are better than the production ranker, the relevant documents have a higher chance to be promoted to top than the irrelevant ones.

In the WSDM Evaluation setup, we compare the performance of BARACO and MT using the following metrics: AUC and Pearson correlation as before. But we do not use RMSE because the graded relevance and the estimated relevance have different scales from 0 to 2, and from 0 to 1 respectively. The candidates are compared using DCG instead of the metric in (3.3) because (3.3) requires a mapping from relevance labels to attractiveness and satisfactoriness that is not available for the graded relevance in the click log—it could be computed using, e.g., DBN but then the evaluation would be less sound because the same click model would be used for training the parameters of the documents and for training the metric.[2]

## 3.6.3 Parameter settings

Both BARACO and MT are instantiated with the user persistence parameter $\gamma = 0.9$, as in [19, 23], $\epsilon = 0.01$. For the Metropolis-Hastings sampling procedure described in Section 3.5.2, the variance of the proposal distribution was set to 0.1, which was determined experimentally to provide a rejection ratio of around 0.6. For each query/document pair, $N_{samples} = 1000$ samples are drawn and shuffled to reduce autocorrelation in order to evaluate (3.1). All results are averaged over 30 independent runs for each algorithm. In order to check the significance of observed differences between results, we perform Wilcoxon signed-rank tests; in the result tables, $^{\blacktriangle}$ denotes a significant difference at $p = 0.01$, and $^{\vartriangle}$ at $p = 0.05$.

[2]In the case of the LETOR evaluation experiments, this mapping is known and can be read from Table 3.1.

## 3.7 Results

In this section, we present our experimental results aimed at answering Research Questions 2.1 and 2.2. In Section 3.7.1, we analyse the performance of BARACO and MT on the LETOR data; in Section 3.7.2, we analyse their performance on the WSDM data.

### 3.7.1 LETOR results

In Section 3.7.1, we compare BARACO and MT on the switching problem; in Section 3.7.1, we compare BARACO and the EM-based approach [19] that underlies MT on the difference estimation problem.

#### Results for the switching problem

To address RQ2.1, we compare the ROC curves of BARACO and MT on the switching problem. Such curves show how the true and false positives of both methods change when we fix $\epsilon = 0.01$ and vary across different values of $\delta \in [0, 1]$ for BARACO and $\epsilon_m \in [-4\epsilon, \epsilon]$ for MT. In this context, a true positive occurs when the algorithm recommends switching to $R_c$ and $M_c + \epsilon \geq M_p$, while a false positive occurs when it recommends switching but $M_c + \epsilon < M_p$. We then compare the AUC of both methods for different datasets and user and click models.

Note that this comparison is fundamentally unfair to BARACO because its parameter, $\delta$, does not require tuning but instead is input by the system designers as a quantification of their risk tolerance. By contrast, $\epsilon_m$ is a parameter that requires manual tuning and cannot be derived from $\delta$, which MT ignores. As discussed in Section 3.6, tuning this parameter is quite difficult in practice. Because the ROC curves show performance across different values of $\delta$ and $\epsilon_m$, they allow MT to "cheat" by optimizing its critical parameter, a step that is unnecessary in BARACO. In other words, these ROC curves answer the following question: for each value of $\delta$ that a system designer could input to BARACO, how well can MT match the quality of BARACO's decisions about when to switch to $R_c$ *if an oracle provides MT with the best possible value of* $\epsilon_m$? Thus, BARACO can be considered a success if it can match the performance of MT when MT is given this advantage.

Figures 3.3, 3.4 and 3.5 plot the area under the ROC curves for BARACO and MT for all six data sets, three click models, and three user model settings. Figure 3.6 shows the ROC curve for the TD2004 dataset, informational user model with the DBN model used for generation. The other ROC curves, omitted for brevity, are qualitatively similar.

The results in Figure 3.3 show that, even at the best value of $\epsilon_m$, BARACO substantially outperforms MT on four of the six datasets (HP2003, HP2004, NP2003, TD2004). On the other two datasets (NP2003 and TD2003), the two methods perform similarly. Analysis of the latter two datasets shows that the production ranker was at a local minimum. Hence, nearly all candidate rankers are better than the production ranker and the best performance is obtained by always switching. As this degenerate policy can be represented just as well by MT as by BARACO, the two perform similarly.

Furthermore, the fact that BARACO performs nearly as well when clicks are generated and interpreted with different models, as shown in Figures 3.4 and 3.5, shows that BARACO is robust to violations of its modeling assumptions. The baseline substantially outperforms BARACO only for the perfect SDBN user model on the TD2003 and TD2004 datasets. The baseline shows superior performance because there are many more relevant documents in the TD datasets, and many of them are not presented to the user by the production ranker. In the case of the perfect user model, the user only clicks on relevant documents. Therefore, there are many queries for which no clicks are produced because no relevant documents were shown. The baseline essentially removes such queries from consideration through condensing [22], which may be an effective strategy in this case. Overall, these results demonstrate that BARACO can offer a robust and effective means for deciding when to switch rankers: especially in cases where its modeling assumptions hold, it outperforms MT with a tuned $\epsilon_m$ parameter for nearly all combinations of dataset and user model.

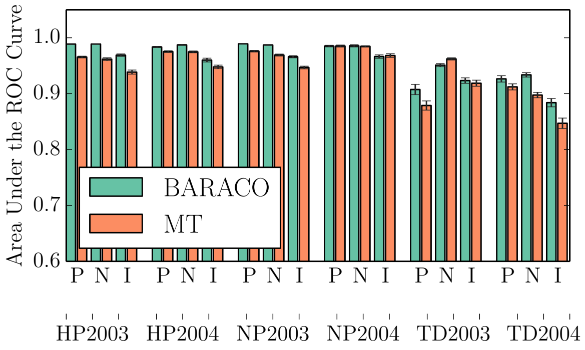


**Figure 3.3: Using DBN for generation and interpretation.**

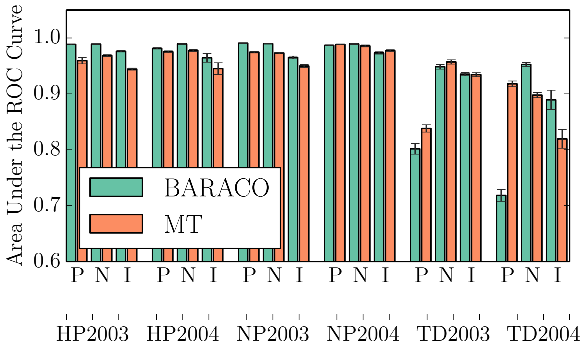


**Figure 3.4: Using SDBN for generation and DBN for interpretation.**

The values of $\epsilon_m$ shown here were chosen because their inflection point lies in the interval $\delta \in [0, 1]$. These are all negative values of $\epsilon_m$ because MT has a consistent negative bias: almost all candidate rankers receive a score lower than the production

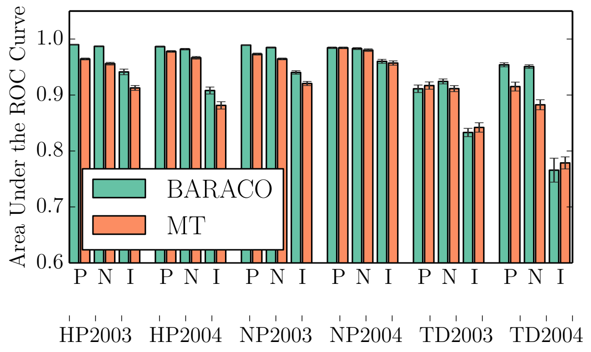


**Figure 3.5: Using UBM for generation and DBN for interpretation.**

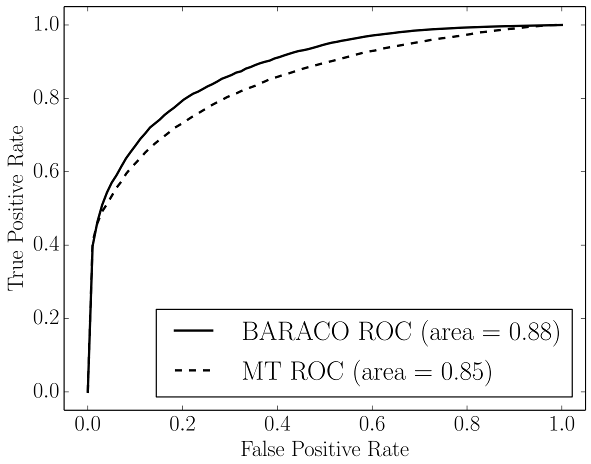


**Figure 3.6: The ROC curves for BARACO and MT, using DBN for generation and interpretation, TD2004 dataset, informational user model (LETOR evaluation).**

ranker. This bias is a consequence of condensing [22]: almost all candidate rankers have unseen documents that do not contribute to the metric. This further highlights the brittleness of MT: as more data is collected, the relative number of unseen documents may decrease, which would reduce the effect of condensing and therefore the amount of bias, necessitating a retuning of $\epsilon_m$.

### Results for the difference estimation problem

To address RQ2.2, we compare BARACO to the EM-based method [19] that underlies MT, on the difference estimation problem. BARACO uses (3.14) to estimate the expected difference while the EM-based method uses $\hat{M}_c - \hat{M}_p$, where $\hat{M}_x$ is the maximum a posteriori estimate of $M_x$. First, we consider how the RMSE scores of the two approaches differ. Error is defined here to be the difference between the true and estimated value of $M_c - M_p$. The true values are computed from the expert-generated

**Table 3.2: RMSE between the predicted outcome of the comparison and the ground truth, P – perfect user model setting, I – informational, N – navigational (LETOR evaluation).**

| | | DBN | | SDBN | | UBM | |
|---|---|---|---|---|---|---|---|
| Dataset | UM | BARACO | EM | BARACO | EM | BARACO | EM |
| HP2003 | P | 3.1e−5$^{\blacktriangle}$ | 3.0e−4 | 3.8e−5$^{\blacktriangle}$ | 3.1e−4 | 3.5e−5$^{\blacktriangle}$ | 3.2e−4 |
| | N | 2.8e−5$^{\blacktriangle}$ | 3.5e−4 | 2.6e−5$^{\blacktriangle}$ | 3.1e−4 | 4.7e−5$^{\blacktriangle}$ | 3.0e−4 |
| | I | 6.1e−5$^{\blacktriangle}$ | 2.7e−4 | 4.1e−5$^{\blacktriangle}$ | 2.7e−4 | 1.1e−4$^{\blacktriangle}$ | 2.6e−4 |
| HP2004 | P | 2.7e−5$^{\blacktriangle}$ | 2.3e−4 | 4.3e−5$^{\blacktriangle}$ | 2.0e−4 | 2.7e−5$^{\blacktriangle}$ | 2.4e−4 |
| | N | 1.6e−5$^{\blacktriangle}$ | 2.4e−4 | 1.9e−5$^{\blacktriangle}$ | 2.3e−4 | 6.0e−5$^{\blacktriangle}$ | 1.9e−4 |
| | I | 6.2e−5$^{\blacktriangle}$ | 2.0e−4 | 5.3e−5$^{\vartriangle}$ | 1.6e−4 | 2.2e−4 | 2.4e−4 |
| NP2003 | P | 1.3e−5$^{\blacktriangle}$ | 2.9e−4 | 1.5e−5$^{\blacktriangle}$ | 2.8e−4 | 1.4e−5$^{\blacktriangle}$ | 2.9e−4 |
| | N | 1.4e−5$^{\blacktriangle}$ | 3.0e−4 | 1.3e−5$^{\blacktriangle}$ | 2.8e−4 | 3.3e−5$^{\blacktriangle}$ | 2.5e−4 |
| | I | 3.7e−5$^{\blacktriangle}$ | 2.8e−4 | 3.8e−5$^{\blacktriangle}$ | 2.7e−4 | 1.1e−4$^{\blacktriangle}$ | 2.7e−4 |
| NP2004 | P | 4.1e−6$^{\blacktriangle}$ | 6.2e−5 | 6.6e−6$^{\blacktriangle}$ | 5.8e−5 | 4.3e−6$^{\blacktriangle}$ | 6.6e−5 |
| | N | 4.9e−6$^{\blacktriangle}$ | 8.9e−5 | 3.6e−6$^{\blacktriangle}$ | 7.8e−5 | 7.9e−6$^{\blacktriangle}$ | 6.4e−5 |
| | I | 1.5e−5$^{\blacktriangle}$ | 9.8e−5 | 1.5e−5$^{\blacktriangle}$ | 1.0e−4 | 2.5e−5$^{\blacktriangle}$ | 1.2e−4 |
| TD2003 | P | 7.0e−5$^{\blacktriangle}$ | 1.1e−4 | 3.7e−4$^{\blacktriangledown}$ | 1.2e−4 | 6.8e−5$^{\blacktriangle}$ | 1.7e−4 |
| | N | 8.0e−5$^{\blacktriangle}$ | 3.5e−4 | 6.2e−5$^{\blacktriangle}$ | 3.8e−4 | 7.6e−5$^{\blacktriangle}$ | 1.9e−4 |
| | I | 6.3e−5$^{\blacktriangle}$ | 2.7e−4 | 6.9e−5$^{\blacktriangle}$ | 2.7e−4 | 1.5e−4$^{\blacktriangle}$ | 3.0e−4 |
| TD2004 | P | 4.0e−4$^{\blacktriangle}$ | 7.8e−4 | 1.2e−3$^{\blacktriangledown}$ | 5.9e−4 | 5.1e−4$^{\blacktriangle}$ | 9.2e−4 |
| | N | 3.2e−4$^{\blacktriangle}$ | 2.3e−3 | 2.5e−4$^{\blacktriangle}$ | 2.3e−3 | 6.0e−4$^{\blacktriangle}$ | 1.2e−3 |
| | I | 4.7e−4$^{\blacktriangle}$ | 2.0e−3 | 4.2e−4$^{\blacktriangle}$ | 1.7e−3 | 1.2e−3$^{\blacktriangle}$ | 1.5e−3 |

relevance labels in the datasets. Table 3.2 summarizes the RMSE of BARACO and MT. These results show that BARACO consistently outperforms the EM-based approach, in many cases by an order of magnitude. The only exception is the TD2004 dataset for clicks generated using UBM. This exception occurs because there are many unseen relevant documents in the TD2004 dataset and, when the user model assumptions do not hold, the baseline's condensing strategy [22] may be more effective because it does not rely on these assumptions.

However, the fact that BARACO has lower RMSE scores is important only if we are interested in the absolute values of the metric differences. Instead, if we want to be able to rank the candidate rankers by their metric values, we need a different way to compare the methods. To this end, we measure the Pearson's correlation between the ground truth value $M_c - M_p$ and the estimates produced by BARACO or MT. For example, if the correlation with the ground truth was perfect, ordering all the candidate rankers by their ground truth difference with the production ranker would be the same as ordering them by the estimated difference. Thus, the correlation with the ground truth is more informative than RMSE in cases where we care about preserving the ground-truth ranking. This occurs, e.g., when several candidate rankers confidently outperform the production ranker. In such cases, it is desirable to switch to the one that outperforms it by the largest margin, while the exact absolute values of the estimated metrics are not important.

Table 3.5 summarizes the correlations between the ground truth difference of rankers and the difference of rankers computed by BARACO and the EM-based method. Higher correlations with the ground truth mean that the way the rankers are ranked is closer to the ground truth. These results show that BARACO again outperforms the EM-based method. The negative correlation in the informational setting of NP2003 dataset is due to a heavily skewed distribution of candidate rankers when the production ranker is at a local minimum of the ranker space and almost all candidate rankers are better for the queries in which the production ranker has not presented any relevant documents. This situation is unlikely to occur in the real world since production rankers are typically highly engineered and thus more likely to be local maxima than local minima. As with our earlier results, we see that the performance on the TD2004 dataset using UBM for generation is qualitatively different from the other conditions for the reasons mentioned earlier.

**Table 3.3: AUC and Correlation between the predicted outcome of the comparison and the ground truth (WSDM evaluation).**

| | BARACO | Manual Thresholding |
|---|---|---|
| AUC | 0.936 | 0.934 |
| Correlation | 0.751 | 0.735 |

To answer RQ2.2, we observe that both the RMSE and correlation results show that BARACO outperforms MT: BARACO achieves better estimates in both absolute and relative terms, except on the TD2004 dataset with the UBM click model for generation, whose special nature has been recognized before.

## 3.7.2 WSDM results

In this section, we compare BARACO and MT on the switching problem and difference estimation problem, respectively, using the WSDM experimental setup.

### Results for the switching problem

To address RQ2.1, we compare the ROC curves of BARACO and MT on the switching problem. The AUC of these curves for both methods are stated in Table 3.3. The AUCs illustrate that both methods are able to distinguish between strong and weak candidates. Both methods suffer from a weak, systematic bias—they consistently underestimate the quality of the candidates because the relevance labels used in the evaluation are biased towards top results.

The real proportion of candidate rankers that are better than the production ranker across different probability levels computed by BARACO is summarized in Table 3.4. The probabilities output by BARACO are not perfectly calibrated and instead tend to

**Table 3.4: Proportion of candidate rankers that beat the production ranker across probability levels computed by BARACO (WSDM evaluation).**

| Probability Level | 0–0.1 | 0.1–0.2 | 0.2–0.3 | 0.3–0.4 | 0.4–0.5 |
|---|---|---|---|---|---|
| Proportion | 0.004 | 0.037 | 0.139 | 0.312 | 0.546 |
| Probability Level | 0.5–0.6 | 0.6–0.7 | 0.7–0.8 | 0.8–0.9 | 0.9–1 |
| Proportion | 0.742 | 0.886 | 0.971 | 0.993 | 1.0 |

**Table 3.5: Correlation between the predicted outcome of the comparison and the ground truth, P – perfect user model setting, I – informational, N – navigational (LETOR evaluation)**

| | | DBN | | SDBN | | UBM | |
|---|---|---|---|---|---|---|---|
| Dataset | UM | BARACO | EM | BARACO | EM | BARACO | EM |
| HP2003 | P | $0.961^{\blacktriangle}$ | 0.846 | $0.950^{\blacktriangle}$ | 0.858 | $0.963^{\blacktriangle}$ | 0.857 |
| | N | $0.967^{\blacktriangle}$ | 0.856 | $0.961^{\blacktriangle}$ | 0.856 | $0.950^{\blacktriangle}$ | 0.808 |
| | I | $0.914^{\blacktriangle}$ | 0.763 | $0.930^{\blacktriangle}$ | 0.744 | $0.510^{\blacktriangledown}$ | 0.664 |
| HP2004 | P | $0.963^{\blacktriangle}$ | 0.933 | 0.946 | 0.937 | $0.969^{\blacktriangle}$ | 0.940 |
| | N | $0.980^{\blacktriangle}$ | 0.926 | $0.981^{\blacktriangle}$ | 0.943 | $0.960^{\blacktriangle}$ | 0.908 |
| | I | $0.931^{\blacktriangle}$ | 0.825 | $0.935^{\vartriangle}$ | 0.825 | $0.262^{\blacktriangledown}$ | 0.612 |
| NP2003 | P | $0.979^{\blacktriangle}$ | 0.890 | $0.982^{\blacktriangle}$ | 0.901 | $0.982^{\blacktriangle}$ | 0.897 |
| | N | $0.973^{\blacktriangle}$ | 0.878 | $0.982^{\blacktriangle}$ | 0.891 | $0.969^{\blacktriangle}$ | 0.856 |
| | I | $0.902^{\blacktriangle}$ | 0.777 | $0.887^{\blacktriangle}$ | 0.769 | $0.523^{\blacktriangledown}$ | 0.678 |
| NP2004 | P | $0.922^{\blacktriangle}$ | 0.841 | $0.915^{\blacktriangle}$ | 0.871 | $0.925^{\blacktriangle}$ | 0.861 |
| | N | $0.923^{\blacktriangle}$ | 0.809 | $0.943^{\blacktriangle}$ | 0.835 | $0.864^{\blacktriangle}$ | 0.751 |
| | I | $0.700^{\blacktriangle}$ | 0.571 | $0.678^{\blacktriangle}$ | 0.546 | $0.464^{\vartriangle}$ | 0.393 |
| TD2003 | P | $0.738^{\vartriangle}$ | 0.678 | $0.398^{\blacktriangledown}$ | 0.619 | 0.732 | 0.723 |
| | N | $0.838^{\blacktriangle}$ | 0.813 | $0.873^{\vartriangle}$ | 0.830 | $0.758^{\blacktriangle}$ | 0.675 |
| | I | $0.777^{\blacktriangle}$ | 0.689 | $0.776^{\blacktriangle}$ | 0.702 | $0.009{,}67^{\blacktriangledown}$ | 0.404 |
| TD2004 | P | $0.846^{\vartriangle}$ | 0.783 | $0.434^{\blacktriangledown}$ | 0.817 | $0.887^{\blacktriangle}$ | 0.795 |
| | N | $0.862^{\blacktriangle}$ | 0.776 | $0.890^{\blacktriangle}$ | 0.753 | $0.858^{\blacktriangle}$ | 0.693 |
| | I | $0.755^{\blacktriangle}$ | 0.633 | 0.829 | 0.637 | $-0.156^{\blacktriangledown}$ | 0.464 |

be underestimates. Thus, not all risk prescribed by $\delta$ and $\epsilon$ can be utilized, making the system somewhat overly conservative.

Unfortunately, a limitation of these WSDM experiments is that there is no way to ascertain how much of this bias is due to discrepancies between the relevance labels and the true information needs of the users who generated the clicks and how much is due to discrepancies between BARACO's click model and those users' behavior. However, because the bias is consistent, correcting for this bias, e.g., by learning a constant offset, is straightforward.

### Results for the difference estimation problem

The correlations between the true and estimated value of $M_c - M_p$ computed by BARACO and MT are also stated in Table 3.3. The estimated difference between rankers is strongly correlated with the ground truth in the WSDM dataset, suggesting that both methods can estimate the difference between rankers well given the logged user interactions with the production ranker.

In both experimental setups, i.e., the LETOR and WSDM setup, both BARACO and MT have good performance in most cases and high agreement with the ground truth. In the LETOR setup the performance is often superior to that in the WSDM experiments, especially when there is little noise and no discrepancy between the user behavior and the click model. However, when there is much noise and the clicks are generated and interpreted using different click models, the performance drops to levels lower than in the WSDM experiments. Overall, these results show that, given a reasonable click model, BARACO makes it possible to make informed decisions whether or not to switch to a candidate ranker given historical user interaction data obtained using the production ranker.

## 3.8 Conclusions and Future Work

We presented BARACO, a new click model-based method of ranker comparison with two key features: (1) it compares the performance of rankers using only historical data, and (2) it quantifies the uncertainty in such comparisons. Using BARACO, it is possible to decide, using only historical data collected with the current production ranker, whether one can confidently replace the production ranker with a candidate ranker. The algorithm takes as input $\epsilon$, the degree to which the candidate ranker is allowed to be worse than the production ranker; $\delta$, the probability with which the candidate ranker is allowed to fall outside this threshold; and user interaction logs collected with the production system.

Our experiments show that BARACO can correctly and confidently identify candidate rankers that are $\epsilon$ as good as the production ranker. BARACO outperforms or has performance comparable to that of the MT baseline that requires manual tuning of the threshold $\epsilon_m$ through offline assessments or online user experiments.

A natural application of BARACO is within online learning to rank algorithms, many of which require an evaluation subroutine. For example, in Dueling Bandit Gradient Descent (DBGD) [154], the algorithm randomly picks a candidate ranker from the neighborhood of the current ranker, compares it to the current ranker, and, if the candidate ranker appears to be better, updates the current ranker so that is is closer to the candidate ranker. The evaluation step is usually done using interleaving, which requires showing potentially poor rankings to the user. Using BARACO, DBGD could be performed while restricting the rankings shown to users to those generated by production rankers or candidate rankers in which we have sufficient confidence.

In this chapter we addressed research question RQ2: Does quantifying uncertainty in click models help to evaluate search engine rankers? We explicitly modeled the user interactions with an interactive system using a click model. We also modeled user satisfaction with a click based metric. This, together with the Bayesian approach we took, allowed us to both estimate the quality of an interactive system given a finite and biased interaction log and to decide confidently if one system is better than another one. We showed that it is important to explicitly model the biases in the logged interaction dataset. In the next chapter we turn away from evaluation to the question of optimizing interactive systems.

# 4 Deep Online Learning to Rank for Image Filtering with Implicit Feedback

Next we turn to interactively optimizing interactive systems using interactions and answer the research question **RQ3:** How to optimize deep neural networks using implicit feedback to perform learning to rank online? This topic is difficult by itself and it is difficult to study. How would we know if an algorithm we propose works and works better than state-of-the-art baselines? Of course, the best choice is to deploy it in the wild and measure user satisfaction. However, it happens that most researchers do not have access to an interactive system with real users interacting with it.

One of the possible ways forward would be to collect an interaction log and use it to evaluate our algorithm. In the previous chapter we have shown that it is possible to compare some versions of an interactive system to some other ones based on interaction data collected using a production system. However, one can confidently decide if one system is better than another one only if there is enough data supporting it. As a result, if a system is too different from the one used to collect the data, then we cannot confidently estimate its quality.

We take an alternative approach that was first proposed by Joachims [65] and then followed by the community [53, 55, 121]. While also resorting to simulations we take a conservative stance. We do not directly model user interactions like we did with click models before. Instead, we assume that we have access to an interaction based metric, which is a lesser assumption. We still assume that we can decompose the overall metric to an independent combination of its parts, weighted sum of document relevance scores in our case. This assumption can be lifted by using a context sensitive scoring model like in [104].

We use image filtering using deep neural networks as our task. This task is important because before most of the methods in the literature use simpler online ranking models [40]. Deep neural networks have shown success in a lot of tasks such as computer vision, machine translation, dialogue systems, web search ranking, linguistics and many others so they can be used for many interactive systems. It is important to see how we can learn them interactively. We use image filtering task because it is easy to make a simulation for it without too many manipulations. One can use an annotated image dataset and use the categories that it comes with as queries. Then one can use the

This chapter was published as [81].

labels of the images to compute the metric for any ranking proposed by the evaluated algorithm. In this chapter we demonstrate that it is possible to start with a completely random neural network and train it to perform an image filtering task in an interactive fashion using interactions.

## 4.1 Introduction

One of the core problems in information retrieval is ranking in interactive systems. Given a user's query, the task is to return a list of relevant items and to constantly improve this list based on an incoming stream of user feedback such as clicks. Current Online Learning to Rank (OLTR) methods are limited to tabular or linear models and are therefore suitable for applications where the number of items to be ranked is relatively small or where the items have high-quality feature representations [121, 154]. In this chapter, we study OLTR in the setting of image filtering, a setting that is neither tabular nor linear.

Document filtering is a task that has long been studied [140]. In contrast to traditional ad hoc retrieval scenarios it is characterized by having a set of standing information needs but a changing set of documents to be ranked against representations of those needs. In *image filtering* the document collection is a collection of images; thus, image filtering is the task of identifying relevant images given a standing query [7, 29]. We are particularly interested in scenarios that are characterized by a fixed set of information needs and a set of images that needs to be ranked against each of those needs. Examples include video surveillance with a fixed visual vocabulary [11], (visual) reputation monitoring [3], visual information discovery services such as Pinterest,[1] visual ranking systems with a fixed vocabulary such as those built into mobile or desktop photo management applications either, and product search where the number of product types is fixed (as is the case in most established e-commerce platforms).

Existing tabular OLTR methods such as cascading bandits [74] are not applicable to image filtering because the number of ranked items can be arbitrarily large; moreover, the item set is dynamic and tabular methods, by definition, do not generalize to unseen items. Linear methods such as dueling bandit gradient descent [154] are not applicable because they require effective high quality features – features that are simply not available for images. Modern image understanding heavily relies on Deep Neural Networks (DNN) [51]. However, there exists no OLTR method that is applicable to DNN. To develop an OLTR method, one needs to address the following challenges:

1. how to update DNNs using an incoming stream of implicit user feedback, and
2. how to explore the space of possible image rankings obtained using deep ranking models.

In this chapter we address these challenges and propose novel OLTR methods for image retrieval. To tackle the first challenge, we propose two learning approaches: DCG-loss, which is based on regression, and PG-loss, which is based on policy gradients [67]; both rank images using DNNs and can learn from implicit user feedback. To tackle the second

[1]http://www.pinterest.com

challenge, we compare several exploration policies, namely $\epsilon$-greedy exploration [94], Boltzmann exploration [67, 129] and bootstrapped exploration [105], as well as two natural baselines, pure exploration (i.e., explore randomly) and pure exploitation (i.e., only exploit).

On top of that, existing OLTR methods typically require implicit feedback at the level of individual items, e.g., seeing whether a particular item was clicked or not. However, many metrics apply not to the level of individual items but only to the entire list. Moreover, previous work [103] shows that the list-wise or search engine result page (SERP)-based implicit feedback can improve the result relevance over item-level click-based methods in the image retrieval task. Clearly, all item-level feedback can be turned into list-level feedback but not vice versa. The methods we propose operate on feedback at the list-level and, hence, they are more generally applicable than existing OLTR methods.

We evaluate the proposed OLTR methods, i.e., various combinations of deep ranking models and exploration policies, on an image filtering task using the MSCOCO [85] dataset, which was released in 2017 as part of the COCO and Places Challenge. Examples of input images are shown in Figures 4.1 and 4.2.

**Figure 4.1: A cat in a bag, image from the MSCOCO dataset.**

A single image typically belongs to several categories. For example, Figure 4.1 belongs to the categories “cat” and “bag,” while Figure 4.2 belongs to “person,” “dog,” and “frisbee.” In the MSCOCO dataset, there are 80 categories. Our experimental results show that DCG-loss with $\epsilon$-greedy has the best final performance on the test set, while DCG-loss with pure exploitation performs best during online training.

The main contributions of this chapter can be summarized as follows:

**Figure 4.2: A person playing frisbee with a dog.**

1. We propose two loss functions for training deep ranking models for image filtering, namely DCG-loss and PG-loss, both of which learn from implicit user feedback.
2. We adopt several exploration policies, namely $\epsilon$-greedy, Boltzmann exploration and bootstrapped exploration, to efficiently explore the space of possible image rankings obtained using the above deep ranking models.
3. We propose the first OLTR methods that can learn from list-level implicit feedback instead of the more specific item-level feedback that today's OLTR methods assume.
4. We find that DCG-loss has superior performance compared to PG-loss especially when the list size increases.

The rests of this chapter is organized as follows. We present the problem setting in Section 4.2. In Section 4.3 we present our deep OLTR methods. Section 4.4 presents our experimental setup and in Section 4.5 we present our results. Finally, related work is presented in Section 4.6 and we conclude with Section 4.7.

## 4.2 Problem Setting

In this section, we briefly describe standard OLTR and outline specific properties of the OLTR setting considered in this chapter.

Roughly speaking, standard OLTR methods work as follows: (1) a user submits a query to a ranking system (image filtering in our case); (2) the system returns a ranking

**Table 4.1: Notation used in the chapter.**

| Notation | Meaning |
|---|---|
| $q$ | Query |
| $\mathbf{x}$ | Raw image representation |
| $\mathbf{X}$ | Set of images |
| $f(\mathbf{x})$ | Score of image $\mathbf{x}$ |
| $l = [\mathbf{x}_1, \ldots, \mathbf{x}_k]$ | Ranked list of images |
| $r(l)$ | Reward for ranked list $l$ |

of items according to its current ranking model; (3) the user interacts with the returned ranking and provides implicit feedback (e.g., clicks, dwell time, etc.); (4) the system updates its ranking model based on the observed feedback. This process is performed for every user query.

The OLTR setting considered in this chapter is different from the standard OLTR setting in two aspects. First, we consider the image filtering scenario. This means that an incoming stream of images is filtered according to user interests, which we assume to be expressed in the form a predefined set of queries [7, 29]. Thus, our set of queries is finite as opposed to an infinite set of queries in traditional ad hoc information retrieval and standard OLTR. Second, we consider a more general type of implicit user feedback, i.e., list-level feedback (such as time to success, abandonment, etc.) instead of the item-level feedback (such as clicks, dwell time, etc.) that is used by standard OLTR methods.

The notation used in this chapter is given in Table 4.1.

## 4.3 Method

In this section, we describe the proposed deep OLTR algorithms. Figure 4.3 provides a graphical overview of our method. Our OLTR algorithms employ deep neural networks to assign scores to image candidates (second component from the bottom in Figure 4.3, detailed in Section 4.3.1) and then use exploration policies to decide how to use these scores to rank the candidates (third component from the bottom in Figure 4.3, detailed in Section 4.3.2). After that, we use implicit user feedback to update our scoring model (top component in Figure 4.3, detailed in Section 4.3.3).

We proceed as follows. First, we describe ResNet, the scoring function we use. Second, we describe how we use the scores produced by ResNet to do exploration; here we consider five alternatives. Third, we describe DCG-loss and PG-loss, which are alternative ways of updating the ResNet model based on implicit feedback.

### 4.3.1 Score network

In order to rank image results in response to a query it is a common approach to score those results according to some relationship to the query. In this chapter we propose to use a deep neural network to score results according to a relationship to the query.

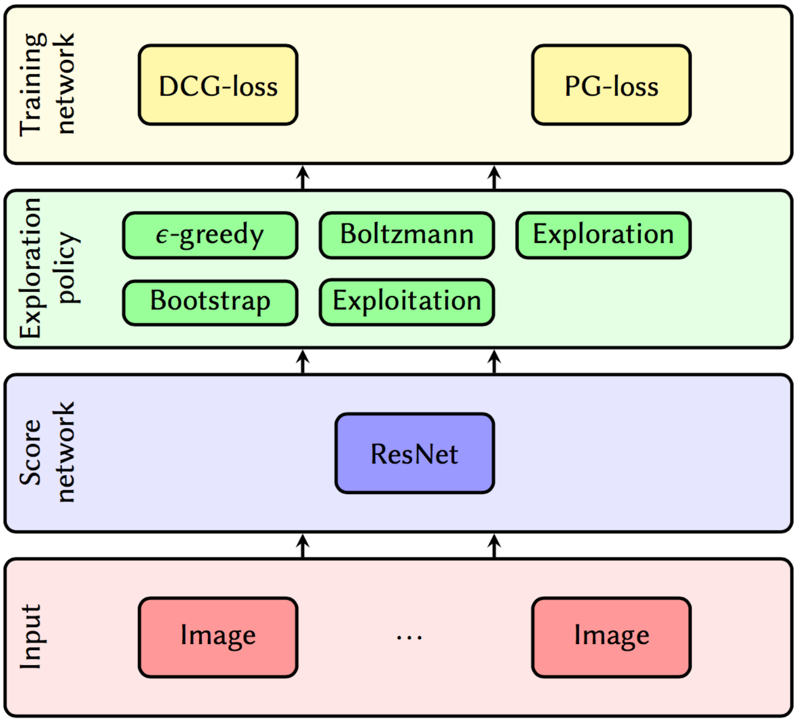


**Figure 4.3: Overview of our deep OLTR method with alternative choices for the exploration policy and for the training network.**

For the image filtering task that we address, we use the ResNet-50 architecture [51]. We have chosen ResNet-50 because it produces high quality results for the object recognition task and it is fast to train. Instead of ResNet-50, we could have chosen other versions of ResNet [51], DenseNet [58], or VGG-Net [125] without having to change anything in the framework depicted in Figure 4.3.

The architecture of ResNet can be summarized as a deep convolutional network with residual connections between layers and a softmax layer; the number of output nodes equals the number of object categories. The depth of the network is equal to 50. In order to obtain a score for an image given a standing information need, we feed the image to the network, obtain scores for all standing information needs, and select the score for the desired information need.

Traditionally, object recognition methods have been trained using cross entropy, full information feedback, and no exploration [125]. Instead of cross entropy, in this chapter we propose two loss functions instead, DCG-loss and PG-loss, which we introduce in Section 4.3.3.

### 4.3.2 Exploration policies

In this section, we detail the exploration policies we plan to use: pure exploitation, pure exploration, $\epsilon$-greedy exploration, Boltzmann exploration [67], and bootstrapped exploration [105] .

#### Exploitation

The first policy we use is pure exploitation (or exploitation, for short), which ranks images only based on the scores produced by the ResNet-50. Since this policy limits the exploration space to the best known subset, it might be good for short-term users' experiences. However, this policy might be bad for training ResNet-50, because ResNet-50 might never see the optimal ranking without the sufficient exploration, and get stuck in a poor local optima.

#### Exploration

In contrast to pure exploitation, pure exploration (or exploration, for short) always explores the space of possible image rankings uniformly. With this policy, ResNet-50 will be trained with diverse ranked lists, and is likely to learn the optimal score function. However, since the results are random during training, exploration is potentially harmful to users' experiences.

#### $\epsilon$-greedy

To balance exploitation vs. exploration dilemma, we describe $\epsilon$-greedy, a standard policy widely used to tackle this dilemma in online and reinforcement learning [67, 129]. $\epsilon \in [0, 1]$ is the only hyper-parameter used in $\epsilon$-greedy. With probability $\epsilon$, the algorithm ranks image candidates randomly (exploration), while with probability $1 - \epsilon$ the algorithm ranks image candidates based on the scores produced by the network, i.e. ResNet-50.

The advantage of exploring with $\epsilon$-greedy is that the exploration is easily controlled by tuning the value of $\epsilon$, where a larger value of $\epsilon$ means a higher degree of exploration. However, $\epsilon$-greedy lacks a convergence guarantee, which may lead to a degradation of the user experience.

#### Boltzmann exploration

Another policy we use is Boltzmann exploration, which is a more sophisticated exploration policy in online and reinforcement learning [67, 129]. In this case, the scores of candidates for a given query are taken into consideration when building the ranked list. Given a query $q_j$, for each position we choose an image according to the distribution

$$P(\mathbf{x}) = \frac{\exp(f(\mathbf{x}))}{\sum_{\mathbf{x}' \in \hat{\mathbf{X}}_j} \exp(f(x'))},$$

where $f(\mathbf{x})$ is the score produced by the network and $\hat{\mathbf{X}}_j \subset \mathbf{X}_j$ is the set of remaining image candidates.

#### Bootstrapped exploration

The final policy we use is bootstrapped exploration, which shows state-of-the-art exploration strategy performance in the Arcade Learning Environment [105]. To fit this policy, we change the outputs of the score network (ResNet-50). The new structure is shown in Figure 4.4. We duplicate the outputs of ResNet $z$ times, where $z$ is the hyper-parameter controlling the exploration. Each duplicate output is called a *head*, which has the exact same structure but is initialized differently. At each round, we uniformly choose one head out of $z$ to score the image. In the update phase, we only update the shared network and the chosen head. This method facilitates exploration by approximating Thompson sampling – a well known exploration strategy [135] using the bootstrap approach [37]. Intuitively, different heads will learn different scorings and explore more diverse sets of rankings.

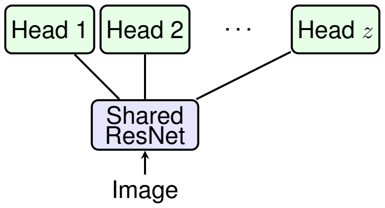


**Figure 4.4: Structure of bootstrapped exploration, one of our exploration policies.**

### 4.3.3 Two ways of training a score network using implicit user feedback

Now that we have obtained rankings, we can collect feedback on those rankings and update our ResNet-50 model. First, we identify our reward signal. After that we introduce two loss functions. To begin, we take the idea of reward regression from Q-learning, which directly estimates the rewards of a list for a given query (here, DCG), and propose DCG-loss. Next, we build on the idea of policy gradient, which estimates the probability of each possible ranking, and propose PG-loss.

#### Reward signal

We use DCG as our reward signal [144]. Clearly, this is an idealized setup rather than a practical one, since we cannot compute the true DCG of a given list in practice. However, this work shows the way of online learning from implicit feedback in the challenging setting of image filtering. We can always change the last layer of DCG-loss, mentioned in the next section, to mimic other measurements, like the probability of clicks of a list for a given query. We leave this as a future work.

#### DCG-loss

As the name suggests, the key idea of DCG-loss is to directly predict the DCG of a ranked list for a given query, e.g., $r(l_j)$ for query $q_j$, where $r(l_j)$ is the DCG (reward) of list $l_j$ defined in Section 4.2. Since we do not know the true relevance scores of images for a given query, the first step will be image scoring as discussed in Section 4.3.1. Then, we sum up these scores with the discounted weight of DCG to get the predicted DCG $\hat{r}(l_j)$. The loss function is the $L2$-loss:

$$\mathcal{L}(r(l_j), \hat{r}(l_j)) = \frac{1}{2}(r(l_j) - \hat{r}(l_j))^2. \tag{4.1}$$

The structure of DCG-loss is shown in Figure 4.5. The last layer is the fixed weight layer, whose weights are the discounted weights used in DCG. This layer sums up the score of each image and outputs the predicted DCG score $\hat{r}(l_j)$ for the given list $l_j$. Except the last discounted weight layer, other components share the same structure and weights. In this chapter, we use ResNet-50 as a shared component. Each component is given a raw image as input, and outputs $M$ relevance scores that describe how relevant the image is to queries, where $M$ is the number of categories described in Section 4.2; they represent the standing information needs in our image filtering scenario. Then DCG-loss passes the relevance scores of query $q_j$ to the last layer. After we obtain the relevance scores, we can use them to build a ranked list.

Since the last layer is a fixed layer and we use the $L2$-loss, which is differentiable, we can back propagate the loss to the score network.

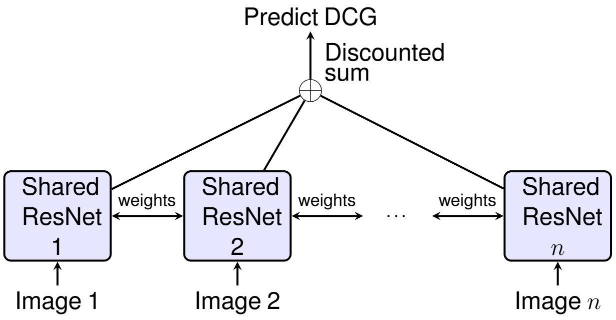


**Figure 4.5: Structure of DCG-loss, used for training a score network using implicit user feedback.**

### PG-loss

In order to apply policy gradients to the image ranking task, we need to model a distribution over possible image rankings, for which we use a Plackett-Luce model [89, 110]. The Plackett-Luce model defines the probability of a ranking as in Eq. (4.2), where we have added the conditional dependence on the query $q$:

$$PL(l_j \mid q_j, X_j) = \prod_{i=1}^{k} \frac{\exp(f(\mathbf{x}_i))}{\sum_{m=i}^{k} \exp(f(\mathbf{x}_m))}. \tag{4.2}$$

Here, $l_j$ is a random variable representing the ranking, $X_j$ is the set of documents to rank, and $f(\mathbf{x}_i)$ assigns a score to $\mathbf{x}_i$ given query $q_j$. We model the scores using a neural network, which is subsequently trained using policy gradients. From a reinforcement learning point of view, we are modeling the policy over the top document using a neural network. Usually, the policy estimator in policy gradient methods outputs a probability distribution over actions directly. However, for combinatorially challenging problems such as image filtering, the number of actions grows with $O(n!)$, which would make things infeasible as we increase the problem size. Thus, we follow the Plackett-Luce model and sample without replacement from a probability distribution over the set of documents [61]. In this way we create a ranking and perform an action from the reinforcement learning perspective, with a probability as defined in Eq. (4.2). Figure 4.6 provides a high-level overview of PG-loss.

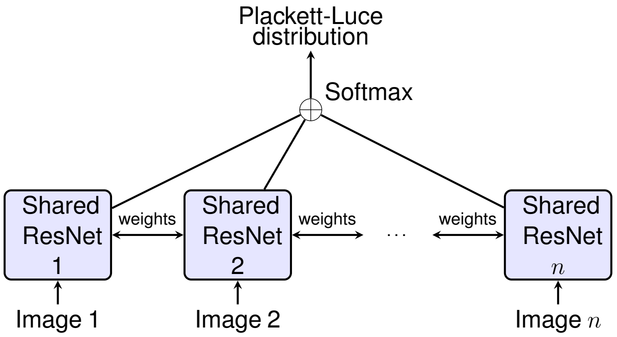


**Figure 4.6: Structure of PG-loss, used for training a score network using implicit user feedback based on policy gradients.**

We train the neural network using policy gradients, using the average-reward formulation [131] as in Eq. (4.3). Note that it contains a double expectation over the query distribution and the ranking distribution. In image filtering we have a uniform unparameterized query distribution, which turns out to be important in order to get a tractable form of the gradient. The ranking distribution is given in Eq. (4.2) and is parameterized with neural network parameters $\theta$; we will therefore denote this distribution $PL_\theta(l_j \mid q_j)$, where we will also drop the dependence on the set of available documents. The reward function $r(l_j, q_j)$ is defined as the NDCG score [63] of the ranking based on its relevance labels, which are either 1 if the image falls within the queried category, or 0 otherwise:

$$\mathcal{L}(\theta) = \mathbb{E}_{q,l}[r(l, q)]. \tag{4.3}$$

The policy gradient takes the form given in Eq. (4.4):

$$\nabla_\theta \mathcal{L}(\theta) = \sum_q P(q) \sum_l \nabla_\theta \big(PL_\theta(l|q) \cdot r(l, q)\big). \tag{4.4}$$

In order to compute the gradient it is necessary to sum over all possible result lists. This is not feasible because the number of possible lists is very large. We would like to estimate this gradient by sampling just some or one of the result lists. In order to achieve this we need to turn this equation into an expectation. In this way we will be able to sample some results lists, compute the gradients for them and get an unbiased estimator of the gradient. Using some simple rewriting, we can get a different form of the gradient as shown in Eq. (4.5) and Eq. (4.6):

$$\nabla_\theta \mathcal{L}(\theta) = \sum_q P(q) \sum_l \frac{PL_\theta(l|q)}{PL_\theta(l|q)} \cdot \nabla_\theta PL_\theta(l|q) \cdot r(l,q) \tag{4.5}$$

$$= \mathbb{E}_{q,l}[\nabla_\theta \log PL_\theta(l|q) \cdot r(l,q)] \tag{4.6}$$

Now the gradient also has the form of an expectation over the query distribution and ranking distribution. This is something we can approximate using sampling, for example using a single Monte Carlo sample.

**Correction for exploration distribution.** The gradient of our model takes the form of an expectation as seen in Eq. (4.6). We can approximate this expectation using Monte Carlo samples from the distribution over rankings. However, we do not sample from the real distribution over actions, but from the distribution provided by the exploration strategy in order to trade off between exploration and exploitation. To correct for this, we have to multiply each sample with a correction weight. It is easy to see that the correction weight should be as shown in Eq. (4.7):

$$\omega_{corr}(l,q,\theta) = \frac{PL_\theta(l|q)}{(1-\epsilon) \cdot PL_\theta(l|q) + \epsilon \cdot p_e(l|q)}. \tag{4.7}$$

Here, $p_e(r|q)$ is the explore distribution and $\epsilon$ is the probability that an exploratory action is taken; for example, in the case of uniform exploration this is a uniform distribution over all rankings.

**Reward transformation.** In order to further reduce the variance we transform the reward by subtracting a baseline from it. By *baseline* we mean a scalar value that approximates the expected reward of the actions taken by the model, i.e., the expected DCG of the result lists that are produced by the current model and shown to the user. We want to increase the probability of actions that are better than expected and decrease the probability of actions that are worse than expected. We achieve this by subtracting a scalar value $b_i$, which we compute as the average of the previous $n_b$ rewards which approximates the expected reward which is an optimal baseline [142] to an acceptable quality:

$$b_i = \frac{1}{n_b} \sum_{j=i-n_b}^{j=i} r(l_j|q_j). \tag{4.8}$$

When we add the correction for the exploration distribution, Eq. (4.7), and add the reward transformation, Eq. (4.8), to the gradients in Eq. (4.6) and approximate them using $i$ finite samples, we obtain the final equation for the gradients of PG-loss:

$$\nabla_\theta \mathcal{L}(\theta) = \sum_i [\nabla_\theta \log PL_\theta(l_i|q_i) \cdot (r(l_i,q_i) - b_i) \cdot \omega_{corr}(l_i,q_i,\theta)]. \tag{4.9}$$

Stepping back, in this section we have introduced the reward signal and the two loss functions that we use to train the score network. The proposed combinations of score network, reward signal and loss functions have the following three advantages:

1. DCG-loss and PG-loss are list-wise learning methods, therefore they directly optimize the target metric, DCG.
2. The shared score components of DCG-loss and PG-loss are flexible enough to be set to any neural network structure, e.g., in our task we use ResNet-50 as the score network, while we can always substitute it with other structures in the future.
3. We do not need item-level feedback, therefore our method can directly optimize both item-level and list-level metrics.

## 4.4 Experimental Setup

Our experiments are designed to address the following research questions:

**RQ3.1** How do the proposed DCG-loss and PG-loss learning methods compare to each other?

**RQ3.2** What exploration policy helps to improve the performance of the proposed DCG-loss and PG-loss methods?

**RQ3.3** How does the proposed architecture (score network + exploration policy + loss function) perform for different sizes of result lists?

### 4.4.1 Datasets

We conduct experiments on the MSCOCO image dataset created for the object recognition problem [85]. This dataset is well-suited for the image filtering task and can be used as follows. In MSCOCO, each image contains objects, where an object can be seen as a category the image belongs to or a query the image is relevant to. Each image in MSCOCO contains one or more objects, which can be interpreted as the image being relevant to one or more queries. More precisely, MSCOCO contains 2.5 million labeled objects in $328,000$ images chosen from a set of 80 objects.

We train our models on the training set of MSCOCO and test them on a left out test set.[2] Before passing images to our score network, ResNet-50, we apply the VGG-Net preprocessing script [125] to raw MSCOCO images in order to rescale them and obtain random crops so as to avoid overfitting [125].

### 4.4.2 Experimental design

Since MSCOCO is originally designed for an object recognition task, we propose the following procedure to fit this dataset to our OLTR task:

1. Given a query, a set of candidate images is selected randomly, such that at least one of the selected images is relevant to the query. This step is the same as in standard LTR systems, where a set of candidate documents is first selected using unsupervised retrieval techniques and then the selected set is ranked using LTR.

[2]Because the test set of MSCOCO does not have public labels, the test set we use is essentially the left out validation set of MSCOCO.

2. ResNet-50 produces a score for each selected image.

3. Then an exploration policy is used to decide how to produce a ranking of images, i.e., based on relevance scores, randomly or sampling and returns this ranking to a user.

4. The user provides list level feedback on the produced ranking of images. Any list level metric could be used here. Since we do not have access to real user feedback in this work, we simulate list level feedback by computing DCG of the produced image ranking.

5. The user feedback is used to update ResNet-50.

Since we focus on the image filtering scenario with a finite set of queries, the same query appears multiple times in the training and test set. On the other hand, the same image may appear in only one of the sets.

In order to avoid updating ResNet-50 frequently and to take advantage of parallelized computations, the above procedure is performed in batches. Each batch contains 100 queries (see Section 4.4.5), which are processed in parallel. We update ResNet-50 after we collect feedback for all 100 queries in a batch.

## 4.4.3 Evaluation measures

We have two types of evaluation. In the first one, we care about the final quality of an image filtering algorithm, which is measured in terms of NDCG on a test set. In the second, we care about user experience during training. It is important to measure both because aggressive exploration can often result in high final quality but may hurt user experience while the system is being trained.

When we measure the final quality, we randomly choose $500$ batches each containing $100$ queries from the test set. Then, we compute the average NDCG over these $50,000$ queries. To verify whether differences between the best performance and the others are significant, we use a two tailed Student's t-test with $p < 0.05$.

When we measure the user experience during training, we compute the discounted cumulative reward:

$$\mathcal{R}(T) = \sum_{t=1}^{T} \gamma^{t-1} r(l_t), \tag{4.10}$$

where $r(l_t)$ is a reward corresponding to an image list $l_t$ and $\gamma \in [0, 1]$ is a discounting factor. By setting different values of $\gamma$, we can measure short-term and long-term user experience. For instance, if we focus on long-term user experience, we should use large values of $\gamma$, so that all summands in Eq. (4.10), both older and newer, contribute to the cumulative reward. In contrast, if we care about short-term user experience, we should use smaller values of $\gamma$, so that only recent summands contribute to the cumulative reward. In our experiments, we report both the cumulative reward ($\gamma = 1$) and the discounted cumulative reward ($\gamma = 1 - \frac{1}{30000}$), which balances the long- and short-term user experience, where $30,000$ is the number of batches (see Section 4.4.5).

### 4.4.4 Experimental conditions

We compare several experimental conditions in terms of different combinations of loss functions and exploration policies. Below, we use subscripts to indicate the exploration policy used: $\epsilon$, $exploration$, $exploitation$, $BOLTZ$ and $BOOT$ stand for $\epsilon$-greedy, pure exploration, pure exploitation, Boltzmann exploration and bootstrapped exploration policies, respectively. In total, we compare $10$ conditions in our experiments:

- DCG-loss$_\epsilon$,
- DCG-loss$_{exploration}$,
- DCG-loss$_{exploitation}$,
- DCG-loss$_{BOLTZ}$,
- DCG-loss$_{BOOT}$,
- PG-loss$_\epsilon$,
- PG-loss$_{exploration}$,
- PG-loss$_{exploitation}$,
- PG-loss$_{BOLTZ}$, and
- PG-loss$_{BOOT}$.

### 4.4.5 Optimization, regularization and hyperparameters

There are many hyperparameters that can potentially influence the performance of our code: the learning rate, $l1$ and $l2$ regularizations, dropout rate, gradient clipping just to name a few. In practice, it is very expensive to tune them for online learning algorithms because each setting of hyperparameters would have to be evaluated online. For this reason, we do not tune the hyperparameters, but use the following predefined values. We use a learning rate of $0.0001$ together with the Adam [69] optimizer, we also use batch normalization [60] with the decay of $0.997$ and epsilon of $0.00001$ which are the default values. We use a batch size of $100$ which is close to the maximum of what can fit in the memory of the GPUs we used. We train our models on $30,000$ batches.

## 4.5 Results

In this section we report and analyse the results of the experiments described in Section 4.4. First, we compare the overall performance of DCG-loss and PG-loss in Section 4.5.1. Then we analyze the effect of the list size in Section 4.5.3. Finally, we compare different exploration policies in Section 4.5.2.

Table 4.2 and Table 4.3 show the NDCG values computed on the test set after the models were trained using DCG-loss and PG-loss on the training set. The values in these table reflect the quality of the user experience after the models finished training using DCG-loss and PG-loss.

**Table 4.2: NDCG@$k$ on the test set of MSCOCO. The best results per value of $k$ (over Table 4.2 and Table 4.3) are marked in boldface. Statistically significant losses over the best results are indicated by $^*$. We report the standard deviation in the subscripts.**

| | DCG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | $0.951_{0.02}$ | $\mathbf{0.953}_{0.02}$ | $0.952_{0.02}$ | $0.951_{0.02}$ | $0.949^*_{0.02}$ |
| $k = 5$ | $0.837_{0.05}$ | $0.816^*_{0.05}$ | $\mathbf{0.838}_{0.05}$ | $0.829^*_{0.05}$ | $0.820^*_{0.05}$ |
| $k = 10$ | $0.691_{0.08}$ | $0.660^*_{0.08}$ | $\mathbf{0.697}_{0.09}$ | $0.681^*_{0.08}$ | $0.667^*_{0.08}$ |

**Table 4.3: NDCG@$k$ on the test set of MSCOCO. The best results per value of $k$ (over Table 4.2 and Table 4.3) are marked in boldface. Statistically significant losses over the best results are indicated by $^*$. We report the standard deviation in the subscripts.**

| | PG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | $0.904^*_{0.02}$ | $0.924^*_{0.02}$ | $0.945^*_{0.02}$ | $0.909^*_{0.02}$ | $0.910^*_{0.02}$ |
| $k = 5$ | $0.638^*_{0.06}$ | $0.604^*_{0.05}$ | $0.626^*_{0.05}$ | $0.666^*_{0.05}$ | $0.514^*_{0.06}$ |
| $k = 10$ | $0.484^*_{0.06}$ | $0.467^*_{0.07}$ | $0.495^*_{0.07}$ | $0.474^*_{0.06}$ | $0.475^*_{0.07}$ |

**Table 4.4: Cumulative reward on the training set of MSCOCO. NDCG@$k$ is used as reward. The best results per value of $k$ (over Table 4.4 and Table 4.5) are marked in boldface.**

| | DCG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | **28257.32** | 24572.42 | 27895.06 | 28120.18 | 28068.68 |
| $k = 5$ | **24089.83** | 17955.74 | 23485.60 | 21429.76 | 23620.31 |
| $k = 10$ | **18878.01** | 14045.72 | 18622.58 | 16518.62 | 18140.91 |

**Table 4.5: Cumulative reward on the training set of MSCOCO. NDCG@$k$ is used as reward. The best results per value of $k$ (over Table 4.4 and Table 4.5) are marked in boldface.**

| | PG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | 27146.26 | 24568.05 | 27784.00 | 27122.83 | 27075.45 |
| $k = 5$ | 18852.77 | 17954.19 | 18432.53 | 19154.88 | 18493.73 |
| $k = 10$ | 14269.26 | 14019.09 | 13805.38 | 14345.53 | 14017.46 |

**Table 4.6: Discounted cumulative reward with $\gamma = 1 - \frac{1}{30000}$ on the training set of MSCOCO. NDCG@$k$ is used as reward. The best results per value of $k$ (over Table 4.6 and Table 4.7) are marked in boldface.**

| | DCG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | $\mathbf{17797.83}$ | 15534.25 | 17574.31 | 17753.56 | 17664.05 |
| $k = 5$ | $\mathbf{15051.26}$ | 11351.03 | 14685.07 | 13386.99 | 14730.12 |
| $k = 10$ | $\mathbf{11724.37}$ | 8880.00 | 11578.96 | 10267.38 | 11216.53 |

**Table 4.7: Discounted cumulative reward with $\gamma = 1 - \frac{1}{30000}$ on the training set of MSCOCO. NDCG@$k$ is used as reward. The best results per value of $k$ (over Table 4.6 and Table 4.7) are marked in boldface.**

| | PG-loss | | | | |
|---|---|---|---|---|---|
| | Exploitation | Exploration | $\epsilon$-greedy | BOLTZ | BOOT |
| $k = 2$ | 17138.80 | 15530.08 | 17506.00 | 17120.39 | 17076.92 |
| $k = 5$ | 11884.52 | 11350.79 | 11662.13 | 12023.88 | 11644.90 |
| $k = 10$ | 9020.68 | 8862.71 | 8770.04 | 9041.78 | 8852.13 |

Table 4.4 and Table 4.5 show the NDCG values accumulated while training the models. These values reflect the quality of the user experience while the models are being trained. Higher values indicate that the user experience is less damaged by exploration.

Similar to Table 4.4 and Table 4.5, Table 4.6 and Table 4.7 show the NDCG values accumulated while training the models but with discounting (see Eq. (4.10)). The discounting is applied to put more weight on the NDCG values accumulated early in the training history. The motivation for this is that we do not just want a model that trains well, but we also want a model that trains fast. The faster training models will have accumulated more reward early on and therefore have higher discounted cumulative performance.

Figure 4.7 presents the training curves of DCG-loss on the left and PG-loss on the right. The NDCG values are computed on the batches used for training the model and then averaged over $3,000$ batches for smoothness.

## 4.5.1 DCG-loss vs. PG-loss

Table 4.2 and Table 4.3 show that DCG-loss clearly dominates PG-loss across all metrics. Even for list size $k = 2$, where PG-loss has a comparable performance, the worst exploration strategy of DCG-loss, namely bootstrapped exploration, outperforms the best exploration strategy of PG-loss, namely $\epsilon$-greedy, with NDCG of $0.949$ vs. NDCG of $0.945$. As the list size grows, the performance of PG-loss degrades and the gap in performance between DCG-loss and PG-loss grows.

Answering **RQ3.1**, we find that DCG-loss is a better choice as the loss function for

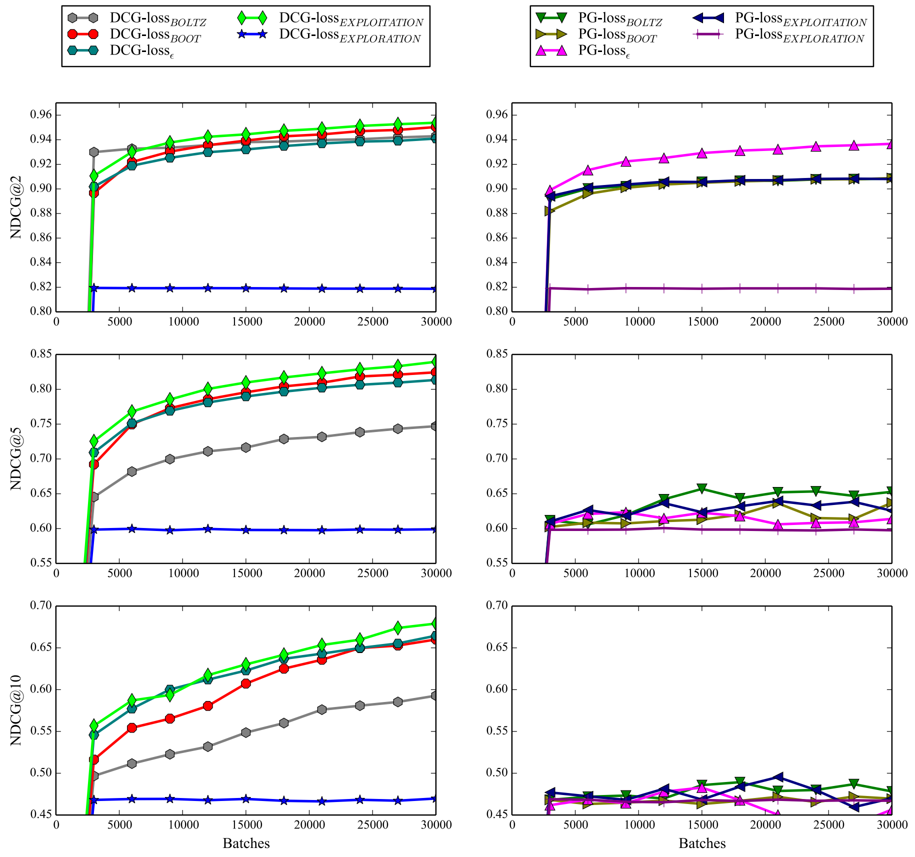


**Figure 4.7: Offline performance (NDCG) of different combination of loss functions (DCG-loss and PG-loss) and exploration policies ($\epsilon-$greedy, Boltzmann, bootstrapped, pure exploration and pure exploitation). Higher is better.**

OLTR with deep neural networks.

## 4.5.2 Exploration

The effect of exploration can be seen by comparing the top performers in Table 4.2–4.7. The best combination of loss function and exploration strategy according to Table 4.2 and Table 4.3 is DCG-loss with $\epsilon$-greedy exploration (the results for $k = 2$ and DCG-loss with pure exploration are not significantly better). However, according to Tables 4.4 and 4.6 the best combination is DCG-loss with pure exploitation. These results are not surprising because exploration is expected to have better long-term performance at the expense of short term performance. This happens because more exploration produces less biased and more diverse training data which increases the performance on the test dataset. However, sampling diverse data points is likely to hurt short term performance when compared to always taking the “best” action according to the model.

Answering **RQ3.2**, we conclude that $\epsilon$-greedy exploration provides the best trade-off between exploration and exploitation. It has good generalization performance, high scores on a heldout dataset, and it does not hurt the online performance as well because it has comparable cumulative and discounted cumulative NDCG performance. Additionally, it is possible to explicitly control the amount of exploration by choosing different values for $\epsilon$. It is also the easiest method to implement and does not add computational overheads.

### 4.5.3 Effect of list size

In order to correctly interpret the results in Tables 4.4–4.7, it is important to keep in mind that NDCG of a random model is $0.81$ for a list size $k = 2$, $0.6$ for $k = 5$ and $0.44$ for $k = 10$. This supports the intuition that increasing the list size increases the task complexity, which can also be explained as follows. First, the feedback signal becomes sparser and the same feedback has to be shared among more items. It makes it more difficult for the algorithms to figure out which items are the relevant ones and which ones are not relevant. This leads to higher variance and slower convergence of both the algorithms. Second, the number of possible rankings grows with the list size: a list with two items can either be ranked correctly or incorrectly, while a list with ten items has $3,628,800$ possible permutations, and if there is just one relevant item then there are ten possible positions on which it can appear. This makes the exploration more difficult.

Tables 4.4–4.7 show that DCG-loss has superior performance across all list sizes. The PG-loss method degrades rapidly as the list size increases. While the performance is comparable for $k = 2$, for $k = 5$ the gap in performance is already huge. However, it is worth noting that PG-loss still manages to learn: its performance is significantly better than random, even for $k = 10$.

We analyze the performance of PG-loss in more detail to understand why it performs so poorly for larger list sizes. We find that the gradients of ResNet-50 vary for different list sizes. In particular, when the list size increases, so do the average $l2$-norm of the gradients and its variance. The variance in gradients can lead to much slower convergence. We further investigate why the gradients have higher variance for larger list sizes and find that the correction weights in Eq. (4.7) take on much more extreme values. The variance of the gradients in Eq. (4.9) depends on the variance of the correction weights and therefore when it is high the variance of gradients is high as well. One of the possible remedies to this problem would be to only explore lists that would have similar values of the correction weights.

Figure 4.7 shows that increasing the list size also affects the speed of convergence. For $k = 2$ the models converge after about $5,000$ batches, while for both $k = 5$ and $k = 10$ the models continue learning even after $30,000$ batches. As discussed above, this happens because for larger list sizes the feedback is more sparse and there are more possible permutations. Answering **RQ3.3**, we find that the proposed architectures will converge slower with the increasing size of lists.

As a summary of all the experiments reported above, we conclude that DCG-loss with $\epsilon$-greedy exploration is the top performing OLTR method across metrics and list sizes.

## 4.6 Related Work

We first give a brief overview of the state-of-the-art in image retrieval in Section 4.6.1. Then we discuss existing OLTR studies in Section 4.6.2. Finally, we survey related work on deep online learning in Section 4.6.3.

### 4.6.1 Image retrieval

Early approaches to image retrieval are mostly based on text-based approaches, where text-based descriptors such as title, description, and tags are used for matching [6]. In contrast, in content-based approaches [127] some level of machine-based understanding of the content of images is used. While early content-based approaches were low-level (e.g., with features such as color histograms, RGB color space, Gabor wavelets, efficient color descriptors), in recent years, deep learning-based approaches to content-based image understanding have made tremendous progress [51, 58, 125] so that very large human interpretable vocabularies can be used for reliable image understanding.

With the increase in quality of content-based image analysis, implicit feedback and other behavioral aspects of image search have begun to draw attention from researchers in recent years. Previous work [108, 111, 136] has illustrated that user behavior on image search engine result pages differs from traditional, linear result pages. For example, image search leads to shorter queries, tends to be more exploratory, and requires more interactions compared to traditional web (text) search. Xie et al. [147] study the examination behavior of image search users and Xie et al. [148] study the intent of image searchers on the web. Hua et al. [57] and Pan et al. [107] study the potential of large-scale click data with a variety of experiments, such as building large-scale concept detectors, tag processing, search, definitive tag detection, intent analysis, etc.

As our understanding of image search user behavior increases, there is a growing number of publications on learning to rank images that somehow exploits user behavior. Jain and Varma [62] use click data as a pseudo relevance signal to train a re-ranking model and also use PCA and Gaussian Process regression to address the sparsity problem of click data in image search. Yu et al. [153] simultaneously use visual features and click features to learn a ranking model. O'Hare et al. [103] extract user behavior features such as hover-through rate and demonstrate that combining these features with content features can yield significant improvements on relevance estimation compared to purely content-based features. However, to train a learning to rank framework using these features, a manually annotated dataset is needed.

In contrast to the work listed above, we propose an online learning to rank method for image search that is based on list-level feedback. To the best of our knowledge, we are the first to do so.

### 4.6.2 Online learning to rank

Learning to Rank (LTR) is used in both offline and online scenarios. Offline LTR [86] is well-studied and widely used both in academia and industry to learn effective ranking models [14, 16, 61, 86]. However, offline LTR methods heavily depend on human-annotated data, which is expensive to produce and may not necessarily align with user satisfaction [119].

Online LTR, instead, learns a ranking model from the interactions between users and an IR system [56, 121]. Most previous studies formulate the Online LTR problem as a Multi-Armed Bandits (MAB) [74, 114, 160] problem or as a linear bandits problem [56, 121, 154].

MAB-based algorithms estimate document relevance for a given query by treating each document as an arm,[3] and the best arm is determined based on user feedback, i.e., clicks. Kveton et al. [74] propose Cascading Bandits that learn document relevance for a given query based on the cascade click model [24]. Zoghi et al. [160] generalize this idea so that document relevance is learned without any assumption about click models. The drawback of this type of algorithm is that they require us to train a separate model for each query and they need a lot of data before they can learn good estimates of document relevance for a given query. Another downside of this type of algorithms is that they require term-level feedback, e.g., clicks on specific items in SERP, an assumption that we are not making in this chapter.

In linear bandits, each ranker is treated as an arm and user feedback is utilized to learn the best ranker online. An important recent development in this area is Multileave Gradient Descent (MGD) [121]. While learning, MGD samples a number of rankers around the optimal ranker that has been learned so far and interleaves all ranked lists output by these rankers. Then MGD evaluates rankers based on user feedback and identifies winning rankers, i.e., those that beat the optimal ranker. If there exist winning rankers, MGD updates the optimal ranker in the direction of either the average or the best of the winning rankers. Importantly, linear bandits based algorithms also require term-level feedback, and, thus, we do not include this type of algorithms in this work either.

### 4.6.3 Deep online learning

Deep online learning is a rapidly growing area of research [94, 105, 106, 132].[4] One of the challenges here is how to efficiently explore the space generated by DNN. Since the space of image rankings generated by DNN is large and complex, neither frequentist exploration policies, like Upper Confidence Bound [4], nor Bayesian policies, like Thompson Sampling [135], are computationally tractable here [105]. For this reason, we focus on traditional exploration policies, namely $\epsilon$-greedy, Boltzmann exploration [67, 129] and the more recently proposed bootstrapped exploration [105].

Mnih et al. [94] use $\epsilon$-greedy in deep reinforcement leaning and propose Deep Q-Learning (DQN), which surpasses a human expert in some of the Atari games. With the $\epsilon$-greedy policy, the learning algorithm (or learner) randomly explores the space generated by DNN randomly with probability $\epsilon$. The main drawback of $\epsilon$-greedy is that

[3]In the bandit setup, an action is called an arm. The goal is to find the best arm based on the historical behavior of arms.

[4]The algorithms discussed here are developed in the area of deep Reinforcement Learning (RL). However, in LTR, online learning can be formulated as RL [55] and so we say *deep online learning* instead of *deep reinforcement learning* here.

the learner always randomly explores with probability $\epsilon$, which potentially degrades the user experience. Although we can choose a small $\epsilon$, it may take a long time for the learner to reach the best action in the space, which, in other words, means that the learner may learn slowly.

In follow-up work, Osband et al. [105] propose Bootstrapped DQN, which improves the learning speed and cumulative reward across most games in the Arcade Learning Environment, a reinforcement learning platform proposed by Bellemare et al. [8].

Boltzmann exploration is another widely used policy in online (or reinforcement) learning. With Boltzmann exploration, the learner scores every action in the space. Then the learner samples an action based on the softmax distribution over all scores. Since the learner explores based on the learned knowledge – when the best action is played more frequently, its probability of being chosen will become higher – Boltzmann exploration is a more sophisticated policy than $\epsilon$-greedy [67].

## 4.7 Conclusion

This chapter has shown a novel way to use deep neural networks for online learning to rank in the image filtering task. The two proposed loss functions, DCG-loss and PG-loss, can be used to train deep OLTR score networks, ResNet-50, with the only requirement being that we need list-wise implicit feedback. Furthermore, ten combinations of loss function and exploration policy have been extensively compared in experiments on the MSCOCO dataset. The main findings are that DCG-loss with $\epsilon$-greedy offers best final quality while DCG-loss with pure exploitation has the best users' experiences during online training. These results have shown the potential to use DCG-loss to train a DNN to learn optimal ranking of image online.

We have focused on five types of exploration policy that are classic but easily implementable. More recent development in deep exploration schemas [106, 132] might also be compared in future work. Meanwhile, since we have found that DCG-loss outperforms PG-loss, we can hypothesize whether this is caused by the policy gradient schema being worse than the Q-learning schema in ranking or by the baseline policy used in policy gradient being too weak to provide useful guidance. If the second case is true, future work should examine how to make deep OLTR doubly robust [33], where the policy gradient uses Q-learning as a baseline.

In this chapter we addressed research question RQ3: How to optimize deep neural networks using implicit feedback to perform learning to rank online? We have shown that it is important which loss function one uses even if the learning signal and the exploration strategy are the same. Therefore, it is important to develop high quality differentiable estimators of the performance of interactive systems that can be computed using the observed interactions. However, what happens when one does not know how to estimate the performance of the interactive system using the observed interactions? Is it possible to optimize an interactive system without explicitly telling the algorithm how to know if the user is satisfied? These are the questions we address in the next chapter.

# 5 Optimizing Interactive Systems with Data-Driven Objectives

In this chapter we answer research question **RQ4:** Is it possible to optimize interactive systems using user interactions without explicit reward? The motivation behind the chapter is that interactive systems are growing more diverse and complex. It is difficult to explicitly define user satisfaction with many interactive systems. For example, how to model the user satisfaction with a chat bot? How to evaluate it? Right now, we either ask the user to explicitly rate it, use task completion or some similarity metric with a golden set of answers. For each new interactive system, for each new task, for each new user it may be is necessary to develop a new metric. In this chapter we attempt to optimize an interactive system without specifying a metric explicitly.

## 5.1 Introduction

Interactive systems play an important role in assisting users in a wide range of tasks. They are characterized by doing so through repeated interactions with humans. For instance, if users are looking for information, interactive systems can assist them in the form of web search engines [145], dialogue systems [82], or intelligent assistants [70]. Here, users interact with systems following the request-response schema: first the user takes an action, which could be a question or a query, then the interactive system produces a reply, which could be an answer or a search engine result page. Such interactions can continue for several iterations until the user decides to stop when he is either satisfied or frustrated with his experience. Importantly, an interactive system and its users always have a shared goal: for users to have the best experience. Thus, both a system and its users are expected to behave accordingly, e.g., the user submits the query that he expects to lead him to the desired results and the interactive system provides the search results that are most helpful to the user. But despite their shared goal, only the user can observe their own experience, leaving interactive systems unable to directly optimize their behavior.

Currently, the optimization of interactive systems relies on assumptions about user needs and frustrations [83]. Commonly, an objective function is manually designed to

This chapter was published as [41].

reflect the quality of an interactive system in terms of user satisfaction. The drawback of this approach is that it is heavily based on domain knowledge, e.g., clicks on search results [92] or the cross-entropy between generated replies and predefined answers [82]. Additionally, a handcrafted objective function is expensive to maintain and does not generalize over different domains. Moreover, it is impossible to design such functions when there is a lack of domain knowledge. Given an objective function, optimization can be done following the Reinforcement Learning (RL) paradigm; previous work does this by considering an interactive system as the agent and the stochastic environment as a user [55, 82]. However, user needs are inherently complex and depend on many different factors [72, 143]. Consequently, manually crafted objective functions rarely correspond to the actual user experience. Therefore, even an interactive system that maximizes such an objective function is not expected to provide the optimal experience.

In contrast, we propose an approach that overcomes this discrepancy by simultaneously inferring an objective function directly from data, namely user interactions with the system, and optimizing the system for this data-driven objective. Thereby, and in contrast to traditional approaches, our data-driven objectives are learned from user behavior, instead of being hand-crafted. We suppose that by incorporating a data-driven objective, interactive systems can be optimized to an objective closer to the actual user satisfaction; unlike previous methods that optimize for assumed user preferences.

We seek to answer the following **main research question**:

> *Can we optimize an interactive system for users through data-driven objectives?*

To answer this research question, we introduce a novel algorithm: Interactive System Optimizer (ISO). It provides a new principled approach by concurrently inferring data-driven objectives from their interactions, and optimizing the interactive system accordingly. Thus, ISO does not depend on any domain knowledge.

In this chapter, we start by formalizing the interaction process between a user and an interactive system as a MDP (Section 5.3). Then we make the following contributions:

- The first method that infers data-driven objectives solely from user interactions, that accurately reflect the users' needs without using any domain knowledge (Section 5.4).

- A novel algorithm, ISO, that optimizes an interactive system through data-driven objectives (Section 5.5). Our experiments with different types of simulated user behavior (Section 5.6) show that ISO has higher performance by increasing the expected state value by at least $89\%$ and up to $136\%$ (Section 5.7).

## 5.2 Background

Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) are the fundamental techniques used in the framework we propose in this chapter.

In RL an agent learns to alter its behavior through trial-and-error interactions with its environment [130]. The goal of the agent is to learn a policy that maximizes the expected return. RL algorithms have successfully been applied to areas ranging from traditional games to robotics [32, 79, 80, 95, 124, 141, 156].

The task of IRL is to extract a reward function given observed, optimal (or suboptimal) behavior of an agent over time [100]. The main motivation behind IRL is that designing an appropriate reward function for most RL problems is non-trivial; this includes animal and human behavior [1], where the reward function is generally assumed to be fixed and can only be ascertained through empirical investigation. Thus, inferring the reward function from historical behavior generated by an agent's policy can be an effective approach. Another motivation comes from imitation learning, where the aim is to teach an agent to behave like an *expert* agent. Instead of directly learning the agent's policy, other work first recovers the expert's reward function and then uses it to generate a policy that maximizes the expected accrued reward [100].

Since the inception of IRL by Russell [118], several IRL algorithms have been proposed. Generally, these methods assume that the environment can be modelled as an MDP. Many IRL methods [87, 159] model the reward functions as linear combinations of hand selected state features; these linear functions are then chosen so that they can explain the behavior of the agent. However, linear functions are rarely capable of explaining complex behavior in real environments. Thus, other work has introduced methods for non-linear reward functions, such as margin-based methods [5, 77, 116], that recover non-linear reward functions through feature construction while assuming the demonstrated behavior is optimal. For suboptimal behavior Levine et al. [78] combine probabilistic reasoning about stochastic expert behavior with non-linear reward functions, outperforming prior methods in suboptimal settings. To avoid their reliance on handcrafted state features, recent methods have exploited the representational capacity of neural networks to approximate complex reward functions without meticulous feature engineering [146]. Alternatively, Finn et al. [39] explore how Inverse Optimal Control can learn behaviors and recover reward functions from demonstrations in high-dimensional robotics settings. Another branch of IRL uses evaluated suboptimal demonstrations where the agent's trajectories are scored by an expert. By changing the expert's role from a demonstrator to a judge, El Asri et al. [38] learn reward functions from the scores even when the transition functions are unknown. Burchfiel et al. [13] show that this IRL method is robust to labelling errors in scored trajectories; a disadvantage of these approaches is that obtaining scores is often very costly.

In this chapter we propose ISO. The critical difference with previous work is that interactive systems are optimized while the objectives are inferred from recovered user reward functions.

## 5.3 Modeling User-System Interactions

In this section we explain how we model user interactions (Section 5.3.1) and define different types of user behavior in interactive systems (Section 5.3.2).

### 5.3.1 Modeling user behavior in interactive systems

We assume that the agent is a user who interacts with the interactive system with the goal of maximizing his expected rewards. This process is modelled using a finite Markov

Decision Process $(S, A, \tau, r, \gamma)$, in the following way:

1. $S$ is a finite set of states that represent responses from the interactive system to the user.

2. $A$ is a finite set of actions that the user can perform on the system to move between states.

3. $\tau$ is a transition probability table and $\tau(s, a, s')$ is the probability of transitioning from state $s$ to state $s'$ under action $a$ at time $t$:
$$\tau(s' \mid s, a) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a). \tag{5.1}$$
The set of all possible $\tau$ is $T$.

4. $r(s, a, s')$ is the expected immediate reward after transitioning from $s$ to $s'$ by taking action $a$. We compute the expected rewards for (state, action, next state) triples as:
$$r(s, a, s') = \mathbb{E}[R_t \mid S_t = s, A_t = a, S_{t+1} = s'], \tag{5.2}$$
where $R_t$ is reward at time $t$.

   For similarity in exposition, we write rewards as $r(s)$ rather than $r(s, a, s')$ in our setting; the extension is trivial [100].

5. $\gamma \in [0, 1]$ is a discount factor.

We write $P$ to denote the set of interactive systems, i.e., triples of the form $(S, A, \tau)$. System designers have control over the sets $S$, $A$ and the transition probability table, $\tau$, and $\tau$ can be changed to optimize an interactive system.

The *user behavior strategy* is represented by a policy, which is a mapping, $\pi \in \Pi$, from states, $s \in S$, and actions, $a \in A$, to $\pi(a|s)$, which is the probability of performing action $A_t = a$ by the user when in state $S_t = s$:
$$\pi(a|s) = \mathbb{P}(A_t = a \mid S_t = s). \tag{5.3}$$

The observed history of interactions between the user and the interactive system, $H$, is represented as a set of trajectories, $\{\zeta_i\}_{i=1}^{n}$, drawn from a distribution $Z$, which is brought about by $\tau$, $\pi$, and $D_0$, where $D_0$ is the initial distribution of states. A *trajectory* is a sequence of state-action pairs:
$$\zeta_i = S_0, A_0, S_1, A_1, \ldots, S_t, A_t, \ldots. \tag{5.4}$$

Next we introduce different ways of generating $\zeta_i$.

### 5.3.2 Defining different types of user behaviors

There are two aspects to characterize the type of user behaviors that influence the shape of $\zeta_i \in H$:

1. **What are the underlying principles that govern how users make decisions while interacting?**
   (a) *Randomly.* Users have no prior information about an interactive system and behave randomly.
   (b) *Optimally.* Users know how to behave optimally in an interactive system to satisfy their needs.
   (c) *Suboptimally.* The behavior is suboptimal, which is better than random but not as good as optimal.
2. **Are users giving explicit feedback about the quality of an interactive system?**
   (a) *Yes.* Users provide us with feedback about the quality of the interactive system by labelling $\zeta_i \in H$.
   (b) *No.* Users do not give us any feedback and the $\zeta_i \in H$ are unlabelled.

To summarize, we have described the basic principles of modeling interactions between users and an interactive system. Next, we detail how to define data-driven objectives which are used to optimize an interactive system.

## 5.4 Defining Data-driven Objectives

In this section we present a way to convert user needs to interactive system objectives (Section 5.4.1) and explain how these objectives can be estimated (Section 5.4.2).

### 5.4.1 Defining interactive system objectives

We define the *quality* of an interactive system as the expected quality of trajectories under optimal user policy. The quality of the $i$-th trajectory, $\zeta_i$, is the discounted sum of the rewards of each state in the trajectory: $\sum_{t=0}^{\infty} \gamma^t R_{t+1}$. The expected quality of the $i$-th trajectory, $\zeta_i$, is the value of its starting state, $S_0$, in interactive system under user policy $\pi$:

$$v_\pi(S_0) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \right], \tag{5.5}$$

where the expectation $\mathbb{E}_\pi[\cdot]$ is taken with respect to sequences of states $S_0, S_1, \ldots, S_t, \ldots$ drawn from the user policy $\pi$ and transition probability table $\tau$. The quality of the interactive system under user policy $\pi$ is:

$$\mathbb{E}_{S_0 \sim D_0}[v_\pi(S_0)], \tag{5.6}$$

where $D_0$ is the initial distribution of states. In the proposed setting, the user goal is to find the best policy such that $\mathbb{E}_{S_0 \sim D_0}[v_\pi(S_0)]$ is maximized. And $v_*(S_0)$ defines the maximum possible value of $v_\pi(S_0)$ as follows:

$$v_*(S_0) = \max_{\pi \in \Pi} v_\pi(S_0), \tag{5.7}$$

where $\Pi$ is the set of possible user policies. We formulate the problem of finding the optimal interactive system's transition probability table, denoted $\tau^*$, in the following terms:

$$\tau^* = \arg\max_{\tau \in T} \mathbb{E}_{S_0 \sim D_0}[\upsilon_*(S_0)]. \tag{5.8}$$

Therefore, Eq. (5.8) represents the objective that we use to optimize an interactive system in order to improve the user experience. To estimate these interactive system objectives we first need to recover $R_t$, which we will discuss next.

### 5.4.2 Recovering user rewards

We assume that continued user interactions with the system indicate a certain level of user satisfaction, which can be reflected by experienced rewards. In contrast with $\zeta_i \in H$ presented in Eq. (5.4), the complete history of interactions, $\hat{H}$, consists of trajectories $\hat{\zeta}_i \sim \hat{Z}$, which include the user reward $R_t$:

$$\hat{\zeta}_i = S_0, A_0, R_1, S_1, A_1, R_2 \ldots, R_t, S_t, A_t, \ldots. \tag{5.9}$$

The problem is that the true reward function is hidden and we need to recover it from the collected incomplete user trajectories, $H$, shown in Eq. (5.4). To address this challenge we apply IRL methods (Section 5.2), which are proposed to recover the rewards of different states, $r(s)$, for trajectories $\zeta_i \in H$.

**Assumptions about user reward function.** We make the following assumptions about the user reward function:

- There is a state feature function, $\phi : S_t \to \mathbb{R}^k$, which can describe a state with a $k$-dimensional feature vector.
- There exist unknown true reward weights $\theta \in \mathbb{R}^k$ which linearly map the state features, $\phi : S_t$, to a reward value, $r(s) = \theta^T \phi(s)$, which represent the satisfaction of a user for this state.

We adopt two types of IRL method, as we have assumed two scenarios for user feedback (Section 5.3.2).

**Unlabeled trajectories.** We use Maximum Entropy Inverse Reinforcement Learning (MaxEnt-IRL) [159] if users give no feedback about their experience with an interactive system. The core idea of this method is that trajectories with equivalent rewards have equal probabilities to be selected and trajectories with higher rewards are exponentially more preferred, which can be formulated as:

$$\mathbb{P}(\zeta_i \mid \theta) = \frac{1}{\Omega(\theta)} e^{\theta^T \phi(\zeta_i)} = \frac{1}{\Omega(\theta)} e^{\sum_{t=0}^{|\zeta_i|-1} \theta^T \phi(S_t)}, \tag{5.10}$$

where $\Omega(\theta)$ is the partition function. MaxEnt-IRL maximizes the likelihood of the observed data under the maximum entropy (exponential family) distribution. Its task can be seen as a classification problem where each trajectory represents one class. MaxEnt-IRL employs gradient descent to update the reward weights $\theta$.

**Labeled trajectories.** We employ Distance Minimization Inverse Reinforcement Learning (DM-IRL) [13] for scenarios when users do give us feedback. For DM-IRL, it is not essential for the trajectories to be optimal because trajectories, $\zeta_i \in H$, are labeled. Therefore, DM-IRL directly attempts to regress the user's actual reward function that explains the given labels. DM-IRL uses discounted accrued features to represent the trajectory:

$$\psi(\zeta_i) = \sum_{t=0}^{|\zeta_i|-1} \gamma^t \phi(S_t), \tag{5.11}$$

where $\gamma$ is the discount factor. The score of a trajectory $\zeta_i$ is assumed to be:

$$\text{score}_{\zeta_i} = \theta^T \psi(\zeta_i). \tag{5.12}$$

Since the score for each trajectory is supplied, the task reduces to a normal regression problem.

Once we have recovered the reward function $r(s)$, we can proceed to the optimization objectives presented in Eq. (5.8).

## 5.5 Optimizing Interactive Systems with Data-driven Objectives

In this section, we aim to find the best interactive system for an optimally behaving user. This is equivalent to finding the optimal transition probability table $\tau^*$, defined in Eq. (5.8).

We start by explaining how to maximize the quality of an interactive system for a user behaving according to a fixed stationary policy $\pi$:

$$\tau^*_\pi = \arg\max_{\tau \in T} \mathbb{E}_{S_0 \sim D_0}[\upsilon_\pi(S_0)]. \tag{5.13}$$

This problem is equivalent to finding the optimal policy in a new MDP$^+$($S^+$, $A^+$, $\tau^+$, $r^+$, $\gamma^+$), where the agent is an interactive system and the stochastic environment is a user. In MDP$^+$, the state $S^+_t$ is represented by a combination of the state $S_t$ the user is and the action $A_t$ the user takes at time step $t$ from the original MDP; the action $A^+_t$ is the original state $S_{t+1}$. The interactive system observes the current state $S^+_t$ and picks an action $A^+_t$ under the interactive system policy $\pi^+(A^+_t|S^+_t)$. Then the user returns the next state $S^+_{t+1}$ according to the transition probability $\tau^+(S^+_{t+1}|S^+_t, A^+_t)$ conditioned on the policy $\pi(A_{t+1}|S_{t+1})$ and transition probability $\tau(S_{t+1}|S_t, A_t)$ from the original MDP.

Therefore, finding the optimal $\tau^*_\pi$ from Eq. (5.13) is equivalent to finding the optimal $\pi^+_*$ for MDP$^+$ as follows:

$$\pi^+_* = \arg\max_{\pi^+ \in \Pi^+} \mathbb{E}_{S^+_0 \sim D^+_0}[\upsilon_{\pi^+}(S^+_0)], \tag{5.14}$$

which can be done using an appropriate RL method such as Q-learning or Policy Gradient. $D^+_0$ is the initial distribution of states in MDP$^+$. After we have demonstrated

**Algorithm 4** Interactive System Optimizer (ISO)

```
1: Input: Original system $(S, A, \tau)$, $r$, $\gamma$, $D_0$.
2: Output: Optimized system $(S, A, \tau^*)$
3: Construct original MDP$(S, A, \tau, r, \gamma)$
4: $\pi_*(a|s) = RL(S, A, \tau, r, \gamma)$
5: Transform MDP to MDP$^+(S^+, A^+, \tau^+, r^+, \gamma^+)$:
     • $S_t^+ = (S_t, A_t)$
     • $A_t^+ = S_{t+1}$
     • $\tau^+(S_{t+1}^+|S_t^+, A_t^+) =$
       $\tau(S_{t+1}|S_t, A_t) \cdot \pi_*(A_{t+1}|S_{t+1})$
     • $r(S_t^+)^+ = r(S_t)$
     • $\gamma^+ = \gamma$
6: $D_0^+ \sim (S_0 \sim D_0, A_0 \sim \pi_*(a|S_0))$
7: $\pi^+(A_t^+|S_t^+) = \tau(S_{t+1}|S_t, A_t)$
8: $\pi_*^+(a^+|s^+) = RL(S^+, A^+, \tau^+, r^+, \gamma^+)$
9: $\tau^*(S_{t+1}|S_t, A_t) = \pi_*^+(A_t^+|S_t^+)$
```

how to optimize the interactive system for a given stationary policy, we return to the original problem of optimizing the interactive system for an optimal policy $\pi_*$

We propose a procedure ISO that is presented in Algorithm 4 and has the following main steps:

- *Line 1:* We assume that we have an estimate of the reward function $r(s)$ using one of the IRL methods described in Section 5.3, so we have as input the original system $(S, A, \tau)$, reward function $r$, discount factor $\gamma$ and initial distribution of states $D_0$.

- *Line 2:* ISO outputs the optimized interactive system $(S, A, \tau^*)$.

- *Line 3:* ISO formulates the original system as MDP$(S, A, \tau, r, \gamma)$.

- *Line 4:* ISO uses an appropriate RL algorithm to find the optimal user policy $\pi_*(a|s)$.

- *Line 5:* ISO transforms the original MDP$(S, A, \tau, r, \gamma)$ into the new MDP$^+(S^+$, $A^+$, $\tau^+$, $r^+, \gamma^+)$. In our setting, $S_t^+$ has the same reward value as $S_t$. The discount factor $\gamma^+$ remains the same.

- *Line 6:* ISO transforms $D_0$ to $D_0^+$ to match the distribution of first state-action pairs in the original MDP.

- *Line 7:* The equivalence $\pi^+(A_t^+|S_t^+) = \tau(S_{t+1}|A_t, S_t)$ means that finding the optimal $\pi_*^+$ according to Eq. (5.14) is equivalent to finding the optimal $\tau_\pi^*$ according to Eq. (5.13).

- *Line 8:* We can use an appropriate RL algorithm to find $\pi_*^+(A_t^+|S_t^+)$.

- *Line 9:* ISO extracts $\tau^*(S_{t+1}|S_t, A_t)$ from the optimal system policy $\pi_*^+(A_t^+|S_t^+)$. The extraction process is trivial: $\tau^*(S_{t+1}|S_t, A_t) = \pi^+(A_t^+|S_t^+)$. Therefore, ISO terminates by returning the *optimized* interactive system.

Once ISO has delivered the optimized system $(S, A, \tau^*)$, we expose it to users so they can interact with it. Hence, it is natural to assume that users adjust their policy towards $\tau^*$. After enough iterations the user policy will converge to the optimal one. The iteration between optimizing the interactive system for the current policy and updating the user policy for the current interactive system continues until both converge.

In summary, we have presented the Interactive System Optimizer (ISO). It optimizes an interactive system using data-driven objectives. It works by transforming the original MDP, solving it and using its solution to yield the optimal transition probability table in the original MDP.

## 5.6 Experimental Setup

In this section we explain our experimental setup to test the performance of the Interactive System Optimizer (ISO).

### 5.6.1 Designing an interactive system

To design an interactive system we need a finite set of states $S$, a finite set of actions $A$ and a transition probability table $\tau$. Features of a state $\phi(s)$ are fixed. We use GridWorld as an example of an interactive system. In our setting, GridWorld is an $N \times N$ grid of states, where $N = 6$ ($|S| = 36$). It supports four possible actions per state ($|A| = 4$) that represent the four directions in which a user can move. In standard settings of GridWorld, from any state a user can only jump to neighboring ones, so the transition probability table, $\tau$, is static. For our experimental setup, we design a more complex environment where a user can move between any two states and the transition probability is changeable. For an initial interactive system, $D_0$ is randomly sampled as well as $\tau$. At each iteration ISO delivers $\tau^*$, which substitutes the initial $\tau$.

### 5.6.2 Modeling user behavior

To model user behavior we require a true reward function $r(s)$, and an optimal user policy $\pi_*$. We utilize a linear reward function $r(s)$ by randomly assigning $25\%$ of the states with reward 1 while others with 0. As we use one-hot features for each state, $r(s)$ is guaranteed to be linear. We use $25\%$ because we see quantitatively the same performance when the proportion of the rewarded states changes. We hypothesize that the complexity of the problem is proportional to the entropy in the reward function because it leads to higher entropy in the observed trajectories and higher variance in the reward estimate. We use the value iteration method [158] to obtain the optimal user policy $\pi^*$. There are two main aspects of user behavior, as introduced in Section 5.3.2:

***Types of user trajectory:*** Suboptimality in user behavior influences the quality of the recovered reward functions, which in turn can affect the performance of ISO as it relies on $r(s)$ to optimize an interactive system. To simulate optimal user behavior, we use $\pi_*$ trained with the real reward function. To model suboptimal user behavior we use two user policies: (1) an optimal user policy $\pi_*$; and (2) an adversarial policy $(1-\pi_*)$. We included an adversarial policy instead of a random one because it is the hardest case as users behave opposite of what we expect. The final dataset $H$ is a mix of trajectories generated by two policies. The noise factor (NF) $\in [0.0, 1.0]$[1] determines the proportion of trajectories in $H$ that has been generated by the adversarial policy. Hence, user trajectories are modelled as follows: (1) adversarial (adv) when $NF = 1.0$; (2) optimal (opt) – $NF = 0.0$; and (3) suboptimal (sub) – $NF = 0.4$.

***Types of user feedback:*** The generated history of user interactions $H$ represents the case of *unlabelled trajectories*. To generate a dataset with *labelled trajectories* $\hat{H}$ we calculate the score using $r(s)$ as shown in Eq. (5.12).

At each iteration, we sample six datasets reflecting different types of histories of user interactions: $\hat{H}_{adv}$, $\hat{H}_{opt}$, $\hat{H}_{sub}$, $H_{adv}$, $H_{opt}$, $H_{sub}$, each of size $10,000$ and $|\zeta_i| \in [20, 30]$.

### 5.6.3 Evaluation process

To evaluate the performance of ISO, we report the expected state value under optimal policy (Eq. (5.6)) for an *initial* interactive system and an *optimised* one, which we derive after 200 iterations. A higher expected state value means users are more satisfied while interacting with the interactive system. We randomly initialize 100 reward functions and report the overall performance. Also, relative improvements are computed. We use a t-test to show statistical significance ($p < 0.01$) of derived relative improvements. We separately show the quality of the selected IRL methods for different types of user behavior.

In summary, we have described our experimental setup, which includes the design of an interactive system, user behavior simulation, and evaluation metrics. Next, we present and discuss our experimental results.

## 5.7 Results and Discussion

In this section, we present the experimental results and analyze the performance of ISO and its robustness.

### 5.7.1 Performance of ISO

Table 5.1 displays the expected state values of the *initial* and *optimized* interactive system and the relative improvement (Impr) that ISO achieves using labelled and unlabeled trajectories of the adversarial, optimal, and suboptimal user behavior. ISO manages to improve the interactive system in all cases but one – when there is no feedback and the

[1]For example, $NF = 0.1$ means that 10% of the trajectories are noisy and generated with the adversarial policy.

**Table 5.1: The performance of ISO, measured as relative improvement (*Impr*) in expected state value over the *Initial* interactive system of the *Optimized* version (after 200 iterations) for different types of user behavior (Section 5.3.2): (1) trajectory generation principles: (a) *Adversarial*, (b) *Optimal*, (c) *Suboptimal*; (2) with and without user feedback. * indicates statistically significant changes ($p < 0.01$) using a paired t-test.**

| (1) Trajectories / (2) Feedback | (a) Adversarial | | | (b) Optimal | | | (c) Suboptimal | | |
|---|---|---|---|---|---|---|---|---|---|
| | Initial | Optimized | Impr | Initial | Optimized | Impr | Initial | Optimized | Impr |
| Explicit feedback | 1.78 | 4.21 | 136%* | 1.78 | 4.21 | 136%* | 1.78 | 4.21 | 136%* |
| No feedback | 1.78 | 1.66 | −6% | 1.78 | 3.95 | 122%* | 1.78 | 3.36 | 89%* |

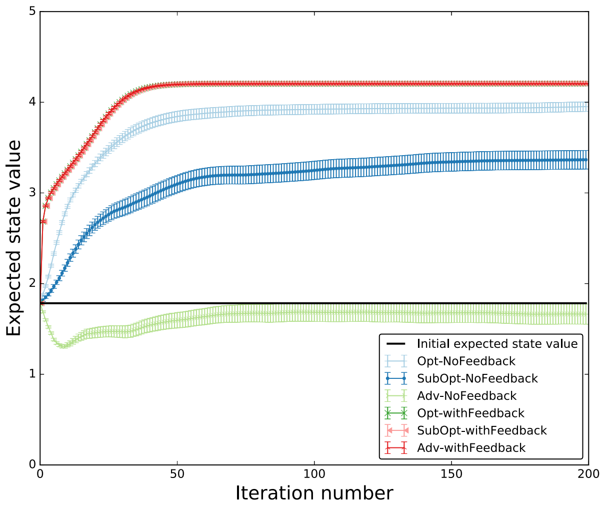


**Figure 5.1: Performance of ISO. Expected state value over 100 random functions with standard error. Note that the curves for Opt-withFeedback, SubOpt-withFeedback and Adv-withFeedback have almost the same shape.**

user behavior is adversarial. As expected, when the user gives feedback about the quality of the trajectories, the task is simpler and ISO manages to get higher improvements than when the labels are not provided. While working with labeled trajectories, ISO is also completely insensitive to the optimality of the user behavior. However, the picture changes when we hide the labels from the trajectories. Without labels, ISO relies on the optimality of user behavior to recover the reward function. As the optimality decreases so does the behavior of ISO, and the performance decays.

### 5.7.2 Improving interactive systems with ISO

Figure 5.1 shows how the quality of the interactive system increases with each iteration of ISO. ISO converges quite fast – as we can see in Figure 5.1, after 50 iterations the expected state value begins to plateau. Most improvements happen in the first several iterations. Also, ISO improves consistently – each iteration is an improvement over the previous one. User trajectories range from optimal (Opt) to suboptimal (SubOpt) to adversarial (Adv) – as long as there is user feedback, ISO is able to improve the initial expected state value. As expected, with respect to adversarial user trajectories without feedback, ISO fails to optimize the interactive system and the expected state value decreases. Thus, ISO works with accurately labelled trajectories, but usually obtaining high-quality labels is intractable and expensive in a real interactive system because the real rewards are invisible.

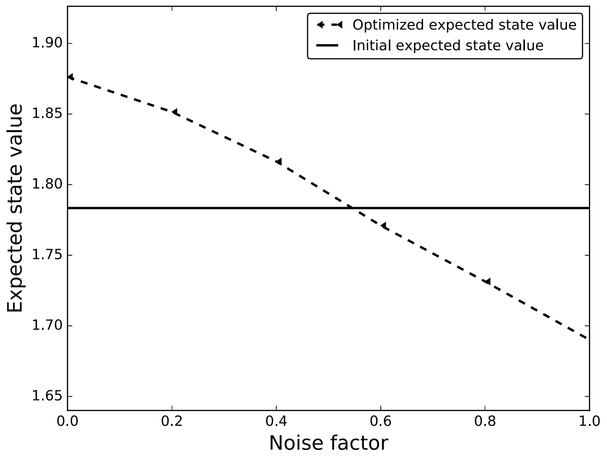


**Figure 5.2: Performance of ISO without feedback under different noise factors after the first iteration.**

### 5.7.3 Suboptimal trajectories in the absence of feedback

In Figure 5.2 we analyze further what happens after one iteration when the user trajectories are increasingly suboptimal and there is no feedback. The suboptimality of user behavior only matters in the absence of labels, so we only plot the performance of ISO without labels across different levels of noise in Figure 5.2.

Recall that the noise factor is the proportion of trajectories generated by the adversarial policy compared to the optimal user policy. The tipping point for our algorithm is around 0.5 – when more than half of the trajectories come from the adversarial policy, ISO starts to deteriorate. However, as long as the noise factor is below 0.5, ISO manages to optimize the interactive system already after the first iteration. With more iterations, the robustness of ISO gets even stronger.

Thus, while ISO is able to deal with unlabelled trajectories, we have to assume that the majority of users behaves optimality; its performance degrades when this assumption is violated. With a noise factor of 0.4, ISO manages to get $89\%$ improvement, with $136\%$ being the maximal improvement (Table 5.1: suboptimal trajectories).

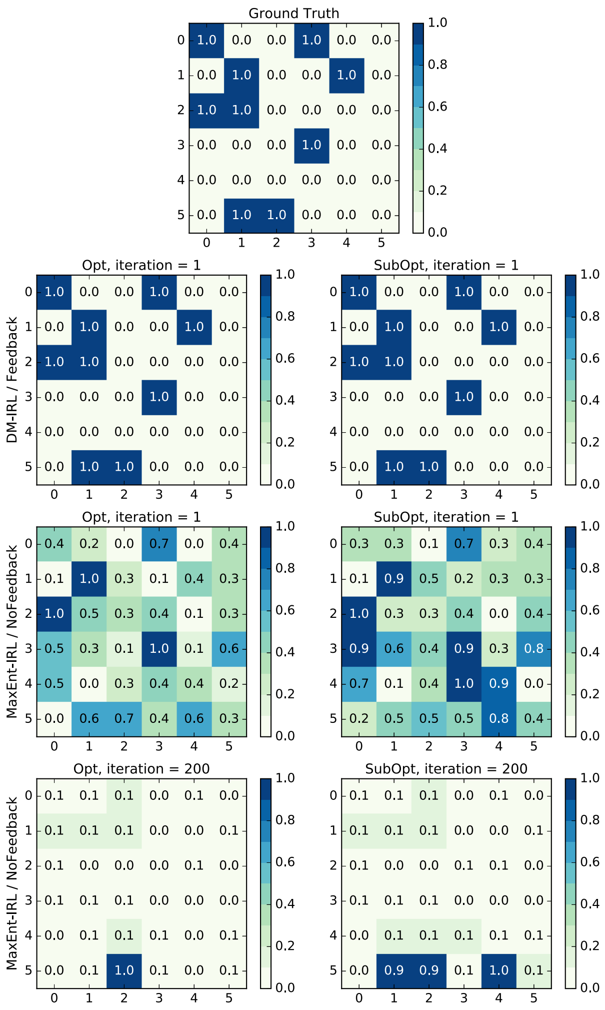


**Figure 5.3: The quality of IRL methods (from top to bottom): *Line 1:* the true $r(s)$; *Line 2:* the recovered $r(s)$ by DM-IRL; *Line 3:* the recovered $r(s)$ by MaxEnt-IRL after the first iteration; and *Line 4:* the recovered $r(s)$ by MaxEnt-IRL after the 200-th iteration. Left column was produced using data from optimal trajectories, right column was produced using data from suboptimal trajectories.**

### 5.7.4 Impact of ISO components

The performance of ISO depends on its two components: (1) RL methods used to optimize the user policy $\pi$ for the original MDP and system policy $\pi^+$ for transformed MDP$^+$; and (2) IRL methods – to recover the true reward function. The dependence on RL methods is obvious – the end result will only be as good as the quality of the final optimization, so an appropriate method should be used. The performance of ISO can be influenced by the quality of the recovered reward functions, $r(s)$, which we analyze for the following types of user behavior: $\hat{H}_{opt}$, $\hat{H}_{sub}$, $H_{opt}$, $H_{sub}$. For the case of labeled trajectories, we can see that values of $r(s)$ recovered by DM-IRL are identical to the ground truth in Figure 5.3 (2nd line).[2] For the case of unlabelled trajectories, the quality of MaxEnt-IRL is worse as shown in Figure 5.3 (3rd and 4th lines). However, MaxEnt-IRL can still give a general overview of $r(s)$ if user trajectories are optimal as presented in Figure 5.3 (3rd line, left). With each iteration of running ISO the shape of the sampled trajectories becomes more similar, which means most trajectories pass by the same states and the diversity of trajectories decreases. This makes it even more difficult to recover $r(s)$ so the MaxEnt-IRL quality deteriorates with the number of iterations. Hence, improving the performance of IRL methods is likely to significantly boost the performance of ISO.

In summary, we have presented the experimental results and analyzed the performance of ISO with and without labels and across different types of user trajectory. We can conclude that ISO works well in the presence of user feedback. In case of unlabelled trajectories, the performance of ISO depends on the optimality of user interactions.

## 5.8 Related Work

Relevant work for this chapter comes in two broad strands: how to optimize interactive systems and what reward signal can be used for optimization.

### 5.8.1 Optimizing interactive systems

Interactive systems can be optimized by direct and indirect optimization. Direct optimization aims at maximizing the user satisfaction directly, in contrast, indirect optimization solves a related problem while hoping that its solution also maximizes user satisfaction [30]. Direct optimization can be performed using supervised learning or RL [96].

Many applications of RL to optimizing interactive systems come from IR, recommender systems, and dialogue systems. Hofmann et al. [53, 56] apply RL to optimize IR systems; they use RL for online learning to rank and use interleaving to infer user preferences [55]. Later work on RL in IR predefines reward functions as the number of satisfied clicks in session search [91, 92].

Shani et al. [122] describe an early MDP-based recommender system and report on its live deployment. Li et al. [82] apply RL to optimize dialogue systems, in particular they optimize the hand crafted reward signals such as: ease of answering, information flow, and semantic coherence.

[2]We sampled 100 different reward functions to run ISO, we report the quality of one $r(s)$, the results of the rest are similar.

### 5.8.2 Rewards for interactive systems

When applying RL to the problem of optimizing interactive systems, we need to have rewards for at least some state-action pairs. Previous work typically handcrafts those, using, e.g., NDCG [102], clicks [73] before the optimization or the evaluation of the algorithm. Instead of handcrafting rewards, we recover them from observed interactions between the user and the interactive system using IRL. Ziebart et al. [157] use IRL for predicting the desired target of a partial pointing motion in graphical user interfaces. Monfort et al. [97] use IRL to predict human motion when interacting with environment. IRL has also been applied to dialogues to extract the reward function and model the user [109]. Typically, IRL is used to model user behavior in order to make predictions about it. We use IRL as a way to recover the rewards from user behavior instead of handcrafting them and optimize an interactive system using these recovered rewards. The work that is closest to ours in spirit is by Lowe et al. [88], who learn a function to evaluate dialogue responses. However, the authors stop at evaluation and do not actually optimize an interactive system.

Thus, the key distinctions between our work and previous studies are that we first use recovered rewards from observed user interactions to reflect user needs and define interactive system objectives, subsequently the interactive system can be optimized according to the defined data-driven objectives to improve the user experience.

## 5.9 Conclusions and Future work

In this chapter, we have recognized that previous work on interactive systems has relied on numerous assumptions about user preferences. As a result, interactive systems have been optimized on manually designed objectives that do not necessarily align with the true user preferences and cannot be generalized across different domains. To overcome this discrepancy, we have investigated the following main research question: *Can we optimize an interactive system for users through data-driven objectives?* As an answer, we have proposed a novel algorithm, the Interactive System Optimizer (ISO), that both infers the user objective from their interactions, and optimizes the interactive system according to this inferred objective.

Firstly, we model user interactions using MDPs, where the agent is the user, and the stochastic environment is the interactive system. Users display one of three types of behavior: random, suboptimal, optimal. Each of these types of behavior reflects different levels of familiarity with the interactive system, i.e., an unexperienced user will display random behavior, whereas an experienced user will maximize their experience by displaying optimal behavior. User satisfaction is modelled by rewards received from certain interactions, and the user interaction history is represented by a set of trajectories. Thus, if a user is not displaying random behavior, their trajectories will be somewhat indicative of their preferences. Optionally, users can also give explicit feedback on the quality of the interactive system by labelling their trajectories.

Secondly, we infer user needs from observed interactions in the form of a data-driven objective. Since the user goal is to find the optimal policy that maximizes his gain from the system, their interactions will indirectly indicate their satisfaction. Making use of this property, we use Inverse Reinforcement Learning (IRL) to recover the user reward function from the observed user behavior. We experiment with two IRL methods, one that works with explicit feedback, the other without. Importantly, these methods work without any domain knowledge, and are thus even applicable when prior knowledge is absent.

Thirdly, ISO optimizes the interactive system to match the inferred objective. The interactive system chooses how to respond to user actions, and from the user perspective these responses are state transitions. However, the interactive system is in control of the transitions, thus these are the actions it chooses from. We optimize the system behavior by using a transformed MDP that represents the system perspective. Using the recovered reward signal the system changes its behavior, and thus how it responds to the user interactions. In response, the user is expected to change his behavior as well, to adopt the new system policy. ISO iterates between optimizing the interactive system for the current inferred objective; and letting the user adapt to the new system behavior. This process repeats until both the user and system policies converge. In the end, both the behavior of the system and the user have been optimized according to the user satisfaction. Our experimental results show that ISO robustly improves the user experience across different types of user behavior.

In conclusion, we have proposed a new approach to infer objectives from user interactions that uses IRL methods. Furthermore, we have invented the principled algorithm ISO that simultaneously infers objectives from interactions, while optimizing a system for these inferred user preferences. Since optimizing an interactive system based on data-driven objectives is novel, many promising directions for future work are possible. For instance, while ISO performs well for users with a singular goal, this approach could be extended for settings with multiple goals. Similarly, extensions considering more personalized goals could benefit the overall user experience. Finally, investigating the scalability and real world applicability of ISO could open many research possibilities.

With this chapter we have reached the end of the fourth and final research chapter in the thesis. Next, we zoom out and formulate our overall conclusions.

# 6 Conclusions

In this thesis we have studied how to interactively optimize web search engines using interactions. Our approach to designing web search engines is different from more traditional ones, which rely on the system designer having enough domain knowledge to construct a full-information supervised dataset. Often, it is too expensive or difficult to provide the web search engines with the knowledge which action it has to take in each context. Instead of using a full-information supervised dataset, we have proposed to use interactions to optimize web search engines. This choice poses a number of challenges. In this thesis we attempted to answer the following RQs:

**RQ1** Which is the best click model to model interactions with a search engine result page?

**RQ2** Does quantifying uncertainty in click models help to evaluate search engine rankers?

**RQ3** How to optimize deep neural networks using implicit feedback to perform learning to rank online?

**RQ4** Is it possible to optimize interactive systems using user interactions without explicit reward?

In this, the concluding chapter, we look back at the studies on which we reported, on the results and answers obtained, and identify the broader implications. We also describe the limitations of our studies and indicate promising directions for follow-up work.

## 6.1 Main Findings

In this section we summarize what we have done in the thesis. The four research chapters of this thesis addressed the challenges of interactively optimizing web search engines using interactions as follows.

### 6.1.1 **RQ1:** Which is the best click model to model interactions with a search engine result page?

In Chapter 2 we answered RQ1. We modeled clicks using click models and analyzed different aspects of their performance. Based on our experimental results we concluded that the performance of different click models depend on the task. In particular, we discovered that the UBM click model [35] has the lowest log-likelihood and that the DBN click model [19] has the lowest perplexity, the SDBN [19] and the DCM [46] click models predict clicks the best as measured by RMSE, and the UBM [35] and the PBM click models predict annotated relevance the best, the CCM [45] click model produces the best ranking feature. We also found that high query frequency and low click entropy make the click model predictions more reliable.

Overall, we concluded that there is no single "best" click model and that click models and annotated relevance, while being correlated, they are vastly different signals. This further served as a motivation for using interactions such as clicks instead of annotations for optimization of interactive systems.

### 6.1.2 **RQ2:** Does quantifying uncertainty in click models help to evaluate search engine rankers?

In Chapter 3 we answered RQ2. We used click models to evaluate search engine rankers using historical clicks and a predefined click based metric – the expected effort metric defined in [23]. In particular, we addressed the switching problem that is described in Chapter 3: is the collected click data enough to confidently conclude that a candidate ranker is better than the production one? We argued that this is an important problem because it is highly practical and it also explores the key issues of interactions with SERPs in the context of web search – namely position bias and presentation bias [65]. We took a Bayesian stance and extended the DBN click model to compute full posteriors of its parameters. We used these posteriors to compute the probability that one ranker is better than another ranker.

We showed that the resulting algorithm BARACO could confidently decide if the candidate ranker is better than the production one and it could better estimate the quality of the ranker than the Expectation Maximization algorithm previously proposed in the literature.

### 6.1.3 **RQ3:** How to optimize deep neural networks using implicit feedback to perform learning to rank online

In Chapter 4 we answered RQ3. We optimized deep image ranking in the context of image filtering – ranking images for a known finite set of information needs. We assumed that we can observe a list-wise user satisfaction metric, a setting that is more general than the setting in the previous chapter that required an item-wise metric and item-wise observations. We proposed and compared two loss functions: DCG-loss and PG-loss and several exploration methods.

We found that DCG-loss with $\epsilon$-greedy exploration has the best overall performance. PG-loss was found not to be able to optimize SERPs that have more than two elements efficiently no matter which exploration strategy is used, while DCG-loss can optimize long result lists.

### 6.1.4 **RQ4:** Is it possible to optimize interactive systems using user interactions without explicit reward?

In Chapter 5 we answered RQ4. We tackled the most complex setting – optimizing an interactive system without predefined user satisfaction metric. We modeled a user interacting with the system as a Markov Decision Process. We looked at scenarios where the user performs optimally or sub-optimally and gives feedback about the quality of the interactions or does not give feedback about the quality of the interactions. First, we recovered user rewards to estimate data-driven objectives using Inverse Reinforcement Learning (IRL) [100], namely we used two IRL methods: Maximum Entropy [146] in the unsupervised case and Distance Based Optimization [13] for the case where the user has left feedback. Then, we optimized interactive systems by optimizing the state transition probabilities of the Markov Decision Process. We did so by transforming the original Markov Decision Process to a new one, where the user and the system are swapped. Then we applied Value Interaction to solve the transformed Markov Decision Process and transform it back to get the state transition probabilities of the newly optimized environment.

We found that we can get even better performance if we deploy the optimized system, collect new user behavior and repeat the optimization. This happens because the users start to explore more rewarding states in the optimized system and the new trajectories lead to better reward estimates. We also found that the method is robust to some degree of suboptimality of the user performance in the unsupervised case without user feedback.

The goal of this thesis has been to develop a better understanding of how to interactively optimize interactive systems using interactions and to develop and evaluate new algorithms for this task. As described above, we advanced towards this goal in several ways, in particular we compared and analyzed click models in the setting of search engines, proposed a novel ranker comparison algorithm, proposed a series of algorithms for image filtering, and an algorithm to optimize interactive systems using data driven objectives. This thesis is intended mainly for industrial practitioners, people who develop interactive systems such as search engines, recommender systems, e-commerce platforms and computational advertising. This thesis may also be interesting for academic researchers in the field of information retrieval and artificial intelligence.

## 6.2 Looking Forward

In this section we start by discussing the limitations of the work on which we report in the thesis by chapter and ways to overcome them. Then we discuss more general directions for future work.

In Chapter 2 we analyze click models. The main limitations of the studied click models is that they only address position and presentation biases of clicks and nothing else. While clicks are a strong signal of user satisfaction, there are many other signals such as dwell time, transactions and others that these click models ignore. Also, often user satisfaction happens without clicks – for instance, the user may be satisfied by a result snippet without clicking it. Complex SERPs such as the ones with many verticals pose additional challenges for click models. The studied click models also assume that the user queries in a session are independent, while this is not true – query reformulations are also an important satisfaction signal. Finally, the click models examined have a tabular representation for document query pairs and therefore do not generalize to new document query pairs. In order to address these limitations more complex click models can be used such as click models that take into account query document features [155].

In Chapter 3 we present BARACO. The main limitations of BARACO are inherited from the limitations of click models – namely tabular representation, no generalization and only click signals. BARACO also has its own limitations – the metric has to be item-wise decomposable and defined on observed clicks or the parameters of the click model. BARACO is computationally expensive – its memory usage is linear in the click log size and it requires sampling from complex posteriors using Metropolis Hasting algorithm, which is quite slow. The size of the posterior in memory can be addressed by using click models that have a conjugate prior and the computation speed can be improved by using a posterior that allows easier sampling such as a Gaussian posterior.

In Chapter 4 the main limitation of the work is that the set of user information needs is finite and known in advance – while this is fine for the image filtering task, ad-hoc search would require generalization across user information needs. This limitation can be addressed by using a neural network architecture where the query is first projected into a dense space and then its cosine similarity with the image embedding is computed [138, 139]. Second, while the algorithm only requires the list-wise metric to be observed, the metric itself should be decomposable as a weighted sum of independent item-wise contributions. This is necessary in order to factor the loss function. This means that metrics with item interactions like diversity aware metrics cannot be optimized by the proposed algorithm. This limitation can be addressed by using a context sensitive scoring method [104].

In Chapter 5 the main limitation of the work is that we either need the user to perform optimally or provide annotations of his or her experience. A second limitation is that the current version of the algorithm would not work in big state action spaces. A third limitation is that we do not model individual user preferences and assume that all users have the same goals and that the reward signal does not change with time. Additionally, the proposed algorithm was evaluated only using a toy environment of Grid World. While the first limitation is fundamental, the other limitations can be addressed by using function approximation in order to handle large state action spaces, by explicitly modeling personal or evolving goals and evaluating the approach on a more realistic task.

Let us turn the table now and discuss potential future directions of research that will mitigate the limitations that we have just listed. This thesis has provided answers to some questions, but it has raised a lot of new ones too. The key challenges of developing interactive systems are their *evaluation*, the amount of *data* they require, and the *algorithms*.

### 6.2.1 Evaluation

While we have high quality offline evaluation techniques for some types of interactive systems, such as search engines and recommender systems, evaluating other interactive systems is very difficult. In particular, evaluating interactive systems is difficult when interactions take multiple turns and only the entire multi-turn trajectory can be evaluated. This happens in search engines when one considers evaluating search sessions instead of individual SERPs and it is particularly pronounced in dialog systems. In dialogue systems this problem happens twice: at the utterance level and at the dialogue level – it is not possible to annotate all responses to a phrase, and it is even more difficult to annotate all sequences of utterance exchanges between the agent and the user. Future work along this line should address these limitations by developing better generalization techniques that would require less annotated data, better active learning techniques to decide which data to annotate and better ways to learn from unannotated user behaviors.

### 6.2.2 Data

In this thesis we have optimized interactive systems using interactions. However, often one needs a lot of interactions to be able to optimize an interactive system. For instance, in Chapter 4 we start to get decent performance only after observing tens of thousands interactions, and similarly in Chapter 5 a big chunk of the state space has to be covered by observed trajectories. This poses a real limitation in the real world, especially for problems with many states and actions. In order to overcome this problem one could leverage unsupervised data that may be is available in abundance. However, it is not clear how to do it. Currently, most approaches train a model on one task – which can be unsupervised and then fine-tune it on the main task. These attempts often result in higher performance. However, this method is not principled as it provides no theoretical guarantees. Future work along this line should utilize massive amounts of unsupervised data and various tasks while guaranteeing the improvement of the performance of the main task.

### 6.2.3 Better algorithms

Optimizing interactive systems is difficult from an algorithmic perspective. One can use many different Reinforcement Learning algorithms in combination with other techniques. In particular, current approaches suffer from high variance and bias. The problem is particularly noticeable in sequential problems with large discrete combinatorial state-action spaces such as dialogue systems. These problems are very difficult and require too much data to be solved using current techniques. Currently, we can only hope to solve these problems if we have access to an oracle that can evaluate any possible interaction trajectory. This is only possible for the cases when the entire agent-environment system can be simulated reliably. However, most real world problems are not currently amendable to simulation. This calls for new algorithms that are able to learn good policies from few available observations or new algorithms that would be able to simulate more complex real world problems.

# Appendices

# A Acronyms

| | |
|---|---|
| **AUC** | Area Under the ROC Curve |
| **BARACO** | Bayesian Ranker Comparison |
| **CM** | Cascade Model |
| **CCM** | Click Chain Model |
| **CTR** | Click-Through Rate |
| **DBN** | Dynamic Bayesian Network |
| **DCG** | Discounted Cumulative Gain |
| **DCM** | Dependent Click Model |
| **DM-IRL** | Distance Minimization Inverse Reinforcement Learning |
| **DNN** | Deep Neural Networks |
| **DRL** | Deep Reinforcement Learning |
| **DCTR** | Document CTR |
| **EBU** | Expected Browsing Utility |
| **EM** | Expectation Maximization |
| **ERR** | Expected Reciprocal Rank |
| **GCTR** | Global CTR |
| **IR** | Information Retrieval |
| **IRL** | Inverse Reinforcement Learning |
| **ISO** | Interactive System Optimizer |
| **LTR** | Learning to Rank |

| | |
|---|---|
| **MAB** | Multi-Armed Bandits |
| **MAP** | Mean Average Precision |
| **MaxEnt-IRL** | Maximum Entropy Inverse Reinforcement Learning |
| **MDP** | Markov Decision Process |
| **MGD** | Multileave Gradient Descent |
| **MLE** | Maximum Likelihood Estimation |
| **MSCOCO** | Microsoft Common Objects in COntext |
| **NDCG** | Normalized Discounted Cumulative Gain |
| **OLTR** | Online Learning to Rank |
| **PBM** | Position-Based Model |
| **RCTR** | Rank CTR |
| **RL** | Reinforcement Learning |
| **RMSE** | Root Mean Squared Error |
| **SERP** | search engine result page |
| **SDBN** | Simplified DBN |
| **UBM** | User Browsing Model |

# B Code

The contributions of this thesis consist of answers to the research questions and of implementations of the algorithms made available to the research community. Providing code is important for three reasons. First, it makes research reproducible and dramatically reduces the effort to reproduce the results. Second, it serves as a reference, often important details can be omitted in the paper that presents the core algorithmic idea, but nothing can be omitted in the code. Finally, having the code for an algorithm makes improving this algorithm a lot easier.

We provide code for the algorithms described in Chapters 3, 4, and 5. The code implementing the algorithms can be found at the following locations

- `https://bitbucket.org/agrotov/lerotexperimental/src`
- `https://bitbucket.org/agrotov/meta-rl-rank/src/master/`
- `https://bitbucket.org/ZimingLi/irl-icml-code/src/master/`

## B.1 Code for Bayesian Ranker Comparison Based on Historical User Interactions

In this section we present the code for Chapter 3. Bayesian Ranker Comparison (BARACO) is a framework to compare search engine rankers given a click log. It takes as input two ranking functions and a dataset with observed clicks and returns the probability that the first ranker is better than the second one. The probability is computed by doing Bayesian inference on the Dynamic Bayesian Network click model. In order to run BARACO one must specify the path to the click log in Yandex WSDM 2014 Web search personalization challenge format. The compared ranking functions are generated randomly by perturbing the rankings in the click log. BARACO can be run using the following command:

```
compare_rankers.py --dataDir <click_data>
      --outputFile <outputFile>
```

## B.2 Code for Deep Online Learning to Rank for Image Filtering with Implicit Feedback

In this section we present the code for Chapter 4, entitled DeepOLTR. DeepOLTR learns to rank images against a finite set of queries interactively. It learns from the observed user reward. At each time step a set of images and a query are sampled. DeepOLTR decides on the ranking of the images and the nDCG for this list is computed and presented to the algorithm. The algorithm then updates the weights of the ResNet-50 network, which it uses for scoring the images. The process then repeats for a set number of time steps. The algorithm is evaluated on final nDCG on a held-out set of images and both cumulative and discounted cumulative nDCG accumulated during learning.

DeepOLTR is a modular framework based on Python and Tensorflow. It implements two loss functions: DCG-loss and PG-loss. It also implements the following exploration strategies: $\epsilon$-greedy, Boltzmann and Bootstrapped exploration.

DeepOLTR uses full-information to bandit conversion. In order to run the experiment we use the MSCOCO dataset [85] originally designed for the object recognition task.

We use the following procedure to fit this dataset to our OLTR task:

1. Given a query, a set of candidate images is selected randomly, such that at least one of the selected images is relevant to the query.
2. ResNet-50 produces a score for each selected image.
3. Then an exploration policy is used to decide how to produce a ranking of images, i.e., based on relevance scores, randomly or sampling and returns this ranking to a user.
4. The user provides list level feedback on the produced ranking of images. Any list level metric could be used here. Since we do not assume to have access to real user feedback in this work, we simulate list level feedback by computing DCG of the produced image ranking.
5. The user feedback is used to update ResNet-50.

Since we focus on the image filtering scenario with a finite set of queries, the same query appears multiple times in the training and test set. On the other hand, the same image may appear in only one of the sets.

The above procedure is performed in batches. Each batch contains 100 queries (see Section 4.4.5), which are processed in parallel. We update ResNet-50 after we collect feedback for all 100 queries in a batch.

Below we describe how to preprocess the data, specify the desired choices of loss function and exploration and run DeepOLTR.

### B.2.1 Data preprocessing

First, the MSCOCO dataset has to be preprocessed in order to convert it to the format used by DeepOLTR. The format is based on Google's protobuf which encodes Tenserflow records. Each record contains the pixel values of the image in three channels and the set of objects present in the image. In order to preprocess the MSCOCO dataset, the following command needs to be run, where the dataDir is the path to the dataset and outputFile is the path where you want to save the preprocessed data:

```
preprocess_coco.py --dataDir <coco_data>
      --outputFile <outputFile>
```

### B.2.2 Algorithm modules

DeepOLTR is modular so before running it you need to specify the loss function and the exploration strategy that you want to use. In order to specify the loss function use the loss function parameter with two possible values, DCGloss and PGloss, for DCG-loss and PG-loss, respectively. In order to choose the exploration type, use the exploration parameter with one of the following values: epsilon, boltz, boot. You can also set epsilon with epsilon parameter and a numerical value between zero and one.

### B.2.3 Putting it together

Summarizing, you can run DeepOLTR with the following command:

```
train_coco.py --serpSize <serpSize>
      --lossFunction <pg|dcg>
      --exploration_type <exploration_type>
```

## B.3 Code for Interactive System Optimizer

In this section we present the code for the main algorithm introduced in Chapter 5, the Interactive System Optimizer (ISO). ISO optimizes the user satisfaction with an interactive system without a predefined satisfaction metric. It does this by collecting a set of trajectories, then recovering the metric using Inverse Reinforcement Learning, and then optimizing the interactive system for this metric. The algorithm proceeds in two steps: first the trajectories are generated, then the algorithm uses them for recovering the metric and optimization.

The algorithm can function with unannotated and annotated trajectories. This is controlled by the `-irl` parameter with values MaxEnt-IRL being the unannotated case and DM-IRL being the annotated one. The trajectories can be generated from an optimal or suboptimal policy, in order to choose, specify `policy_noise` parameter with a numerical value between 0 suboptimal and 1 being optimal.

In order to run Interactive System Optimizer, you need to specify if you want to use annotations using annotated parameter with boolean values.

The command for running Interactive System Optimizer is:

```
pipeline-iteration-test.py -irl <MaxEnt-IRL|DM-IRL>
      -policy_noise <pg|dcg>
```

# Bibliography

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, pages 1–8. ACM, 2004. (Cited on page 73.)

[2] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. In *SIGIR Forum*, volume 42, pages 9–15. ACM, 2008. (Cited on page 2.)

[3] E. Amigó, J. Carrillo-de Albornoz, I. Chugur, A. Corujo, J. Gonzalo, E. Meij, M. de Rijke, and D. Spina. Overview of RepLab 2014: Author profiling and reputation dimensions for online reputation management. In *Information Access Evaluation. Multilinguality, Multimodality, and Interaction*, pages 307–322. Springer, 2014. (Cited on page 50.)

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. (Cited on page 68.)

[5] J. Bagnell, J. Chestnutt, D. M. Bradley, and N. D. Ratliff. Boosting structured prediction for imitation learning. In *NIPS*, pages 1153–1160, 2007. (Cited on page 73.)

[6] J. M. Barrios, D. Diaz-Espinoza, and B. Bustos. Text-based and content-based image retrieval on flickr. In *SISAP*, pages 156–157, 2009. (Cited on page 67.)

[7] N. J. Belkin and W. B. Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of ACM*, 35(12):29–38, 1992. (Cited on pages 50 and 53.)

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. (Cited on page 69.)

[9] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited on pages 28, 33, and 36.)

[10] M. Boddy and T. L. Dean. *Solving time-dependent planning problems*. Brown University, Department of Computer Science, 1989. (Cited on page 3.)

[11] B. J. Boom, J. He, S. Palazzo, P. X. Huang, C. Beyan, H.-M. Chou, F.-P. Lin, C. Spampinato, and R. B. Fisher. A research tool for long-term and continuous analysis of fish assemblage in coral-reefs using underwater camera footage. *Ecological Informatics*, 23:83–97, 2014. (Cited on page 50.)

[12] P. Borlund. The concept of relevance in IR. *Journal of the Association for Information Science and Technology*, 54(10):913–925, 2003. (Cited on page 1.)

[13] B. Burchfiel, C. Tomasi, and R. Parr. Distance minimization for reward learning from scored trajectories. In *AAAI*, pages 3330–3336. AAAI Press, 2016. (Cited on pages 73, 77, and 89.)

[14] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010. (Cited on page 68.)

[15] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005. (Cited on page 1.)

[16] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136. ACM, 2007. (Cited on pages 1 and 68.)

[17] B. Carterette. System effectiveness, user models, and user utility: a conceptual framework for investigation. In *SIGIR*, pages 903–912. ACM, 2011. (Cited on pages 28 and 30.)

[18] 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 page 2.)

[19] O. Chapelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *WWW*. ACM, 2009. (Cited on pages 3, 6, 11, 12, 14, 15, 26, 28, 29, 30, 31, 37, 40, 41, 43, and 88.)

[20] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *CIKM*, pages 621–630. ACM, 2009. (Cited on pages 1, 5, 11, 28, 30, and 37.)

[21] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems*, 30(1):6, 2012. (Cited on page 26.)

[22] A. Chuklin, A. Schuth, K. Hofmann, P. Serdyukov, and M. de Rijke. Evaluating aggregated search using interleaving. In *CIKM*, pages 669–678. ACM, 2013. (Cited on pages 26, 37, 42, 43, and 44.)

[23] A. Chuklin, P. Serdyukov, and M. de Rijke. Click model-based information retrieval metrics. In *SIGIR*, page 493. ACM, 2013. (Cited on pages 11, 25, 26, 28, 29, 30, 31, 40, and 88.)

[24] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool Publishers, 2015. (Cited on pages 3, 11, 14, and 68.)

[25] M. Ciaramita, V. Murdock, and V. Plachouras. Online learning from click data for sponsored search.

In *WWW*. ACM, 2008. (Cited on page 5.)

[26] C. W. Cleverdon, J. Mills, and E. Keen. Factors determining the performance of indexing systems (Volume 1: Design). *Cranfield: College of Aeronautics*, 1966. (Cited on page 27.)

[27] 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 3, 6, 12, and 13.)

[28] N. Dai, M. Shokouhi, and B. D. Davison. Learning to rank for freshness and relevance. In *SIGIR*, pages 95–104. ACM, 2011. (Cited on page 1.)

[29] N. DeClaris, D. Harman, C. Faloutsos, S. Dumais, and D. Oard. Information filtering and retrieval: overview, issues and directions. In *IEEE Engineering in Medicine and Biology Society*, volume 1, pages A42–A49, 1994. (Cited on pages 50 and 53.)

[30] 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 page 84.)

[31] Z. Dou, R. Song, J.-R. Wen, and X. Yuan. Evaluating the effectiveness of personalized web search. *IEEE TKDE*, 21(8):1178–1190, 2008. (Cited on page 16.)

[32] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. $RL^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016. (Cited on page 73.)

[33] M. Dudík, J. Langford, and L. Li. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*, 2011. (Cited on page 69.)

[34] G. Dupret and C. Liao. A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In *WSDM*, pages 181–190, ACM, 2010. (Cited on page 11.)

[35] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR*, pages 331–338. ACM, 2008. (Cited on pages 3, 6, 12, 13, 14, 28, and 88.)

[36] Edelman. 2018 Edelman Trust Barometer Global Report. `http://cms.edelman.com/sites/default/files/2018-02/2018_Edelman_Trust_Barometer_Global_Report_FEB.pdf`, 2018. (Cited on page 1.)

[37] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC press, 1994. (Cited on page 56.)

[38] L. El Asri, R. Laroche, and O. Pietquin. Reward shaping for statistical optimisation of dialogue management. In *SLSP*, pages 93–101. Springer, 2013. (Cited on page 73.)

[39] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, pages 49–58. ACM, 2016. (Cited on page 73.)

[40] A. Grotov and M. de Rijke. Online learning to rank for information retrieval: SIGIR 2016 tutorial. In *SIGIR*, pages 1215–1218. ACM, July 2016. (Cited on pages 5 and 49.)

[41] A. Grotov, A. Chuklin, I. Markov, L. Stout, F. Xumara, and M. de Rijke. A comparative study of click models for web search. In *CLEF*. Springer, September 2015. (Cited on pages 3, 6, and 71.)

[42] A. Grotov, S. Whiteson, and M. de Rijke. Bayesian ranker comparison based on historical user interactions. In *SIGIR*, pages 273–282. ACM, August 2015. (Cited on page 25.)

[43] F. Guo and Y.-m. Wang. Efficient multiple-click models in web search. In *WSDM*. ACM, 2009. (Cited on pages 26 and 28.)

[44] F. Guo, L. Li, and C. Faloutsos. Tailoring click models to user goals. In *Proceedings of the 2009 workshop on Web Search Click Data*, pages 88–92. ACM, 2009. (Cited on page 39.)

[45] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW*, pages 11–20. ACM, 2009. (Cited on pages 3, 6, 12, 14, and 88.)

[46] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, pages 124–131, 2009. (Cited on pages 3, 6, 12, 13, and 88.)

[47] A. Hassan and R. W. White. Personalized models of search satisfaction. In *CIKM*, pages 2009–2018. ACM, 2013. (Cited on page 5.)

[48] A. Hassan, R. Jones, and K. L. Klinkner. Beyond DCG: User behavior as a predictor of a successful search. In *WSDM*, pages 221–230, 2010.

[49] A. Hassan, X. Shi, N. Craswell, and B. Ramsey. Beyond clicks: query reformulation as a predictor of search satisfaction. In *CIKM*, pages 2019–2028. ACM, 2013. (Cited on page 5.)

[50] J. He, C. Zhai, and X. Li. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM*, pages 2029–2032. ACM, 2009. (Cited on page 26.)

[51] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE, 2016. (Cited on pages 8, 50, 54, and 67.)

[52] 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 11.)

[53] 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 5, 49, and 84.)

[54] K. Hofmann, F. Behr, and F. Radlinski. On caption bias in interleaving experiments. In *CIKM*, pages 115–124. ACM, 2012. (Cited on page 26.)

[55] 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 5, 11, 26, 28, 49, 68, 72, and 84.)

[56] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval Journal*, 16(1):63–90, 2013. (Cited on pages 5, 8, 68, and 84.)

[57] X.-S. Hua, L. Yang, J. Wang, J. Wang, M. Ye, K. Wang, Y. Rui, and J. Li. Clickage: Towards bridging semantic and intent gaps via mining click logs of search engines. In *MM*, pages 243–252. ACM, 2013. (Cited on page 67.)

[58] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. (Cited on pages 54 and 67.)

[59] Internet Live Stats. Google search statistics. `http://www.internetlivestats.com/google-search-statistics/`, 2018. (Cited on page 1.)

[60] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. (Cited on page 62.)

[61] R. Jagerman, J. Kiseleva, and M. de Rijke. Modeling label ambiguity for listwise neural learning to rank. In *Neu-IR*, August 2017. (Cited on pages 58 and 68.)

[62] V. Jain and M. Varma. Learning to re-rank: query-dependent image re-ranking using click data. In *WWW*, pages 277–286. ACM, 2011. (Cited on page 67.)

[63] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002. (Cited on pages 15 and 58.)

[64] J. Jiang, A. H. Awadallah, X. Shi, and R. W. White. Understanding and predicting graded search satisfaction. In *WSDM*, pages 57–66. ACM, 2015. (Cited on page 5.)

[65] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142. ACM, 2002. (Cited on pages 2, 26, 49, and 88.)

[66] T. Joachims. Evaluating retrieval performance using clickthrough data. In J. Franke, G. Nakhaeizadeh, and I. Renz, editors, *Text Mining*. Physica Verlag, 2003. (Cited on page 2.)

[67] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. (Cited on pages 8, 50, 51, 55, 68, and 69.)

[68] M. Karimzadehgan, W. Li, R. Zhang, and J. Mao. A stochastic learning-to-rank algorithm and its application to contextual advertising. In *WWW*, pages 377–386. ACM, 2011. (Cited on page 3.)

[69] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 62.)

[70] J. Kiseleva, K. Williams, A. H. Awadallah, I. Zitouni, A. Crook, and T. Anastasakos. Predicting user satisfaction with intelligent assistants. In *SIGIR*, pages 45–54. ACM, 2016. (Cited on pages 5 and 71.)

[71] R. Kohavi, R. Longbotham, D. Sommerfield, and R. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18:140–181, 2009. (Cited on pages 26, 27, and 28.)

[72] M. Kosinski, D. Stillwell, and T. Graepe. Private traits and attributes are predictable from digital records of human behavior. *PNAS*, 110:5802–5805, 2013. (Cited on page 72.)

[73] M. Kutlu, V. Khetan, and M. Lease. Correlation and prediction of evaluation metrics in information retrieval. *arXiv preprint arXiv:1802.00323*, 2018. (Cited on page 85.)

[74] B. Kveton, C. Szepesvari, Z. Wen, and A. Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML*, pages 767–776, 2015. (Cited on pages 5, 50, and 68.)

[75] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2011. (Cited on page 1.)

[76] LETOR: Learning to Rank for Information Retrieval. Letor: Learning to rank for information retrieval, 2009. URL `https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/`. (Cited on page 2.)

[77] S. Levine, Z. Popovic, and V. Koltun. Feature construction for inverse reinforcement learning. In

*NIPS*, pages 1342–1350, 2010. (Cited on page 73.)

[78] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *NIPS*, pages 19–27, 2011. (Cited on page 73.)

[79] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. (Cited on page 73.)

[80] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, pages 173–184, 2016. (Cited on page 73.)

[81] C. Li, A. Grotov, B. Eikema, I. Markov, and M. de Rijke. Deep online learning to rank for image filtering with implicit feedback. In *Submitted*, 2018. (Cited on page 49.)

[82] J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao. Deep reinforcement learning for dialogue generation. In *EMNLP*, pages 1192–1202. ACL, 2016. (Cited on pages 71, 72, and 84.)

[83] Z. Li, J. Kiseleva, M. de Rijke, and A. Grotov. Towards learning reward functions from user interactions. In *ICTIR*, pages 941–944. ACM, 2017. (Cited on page 71.)

[84] Z. Li, A. Grotov, J. Kiseleva, M. de Rijke, and H. Oosterhuis. Optimizing interactive systems with data-driven objectives. *arXiv preprint arXiv:1802.06306*, February 2018. (Cited on page 11.)

[85] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014. (Cited on pages 5, 51, 60, and 98.)

[86] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. (Cited on pages 67 and 68.)

[87] M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *ECML/PKDD*, pages 31–46. Springer, 2009. (Cited on page 73.)

[88] R. Lowe, M. Noseworthy, I. V. Serban, N. Angelard-Gontier, Y. Bengio, and J. Pineau. Towards an automatic Turing test: Learning to evaluate dialogue responses. In *ACL*, pages 1116–1126, 2017. (Cited on page 85.)

[89] R. D. Luce. *Individual Choice Behavior a Theoretical Analysis*. John Wiley and sons, 1959. (Cited on page 57.)

[90] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *SIGIR*, pages 587–596. ACM, 2014. (Cited on page 3.)

[91] J. Luo, X. Dong, and H. Yang. Learning to reinforce search effectiveness. In *ICTIR*, pages 271–280. ACM, 2015. (Cited on page 84.)

[92] J. Luo, X. Dong, and H. Yang. Session search by direct policy learning. In *ICTIR*, pages 261–270. ACM, 2015. (Cited on pages 72 and 84.)

[93] Y. Lv, T. Moon, P. Kolari, Z. Zheng, X. Wang, and Y. Chang. Learning to model relatedness for news recommendation. In *WWW*, pages 57–66. ACM, 2011. (Cited on page 3.)

[94] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. (Cited on pages 8, 51, and 68.)

[95] V. Mnih, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015. (Cited on page 73.)

[96] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012. (Cited on page 84.)

[97] M. Monfort, A. Liu, and B. Ziebart. Intent prediction and trajectory forecasting via predictive inverse linear-quadratic regulation. In *AAAI*, pages 3672–3678. AAAI Press, 2015. (Cited on page 85.)

[98] Movielens. Movielens dataset. `https://grouplens.org/datasets/movielens/`, 2018. (Cited on page 2.)

[99] Netflix. Netflix prize data. `https://www.kaggle.com/netflix-inc/netflix-prize-data`, 2017. (Cited on page 2.)

[100] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670. Morgan Kaufmann, 2000. (Cited on pages 73, 74, and 89.)

[101] Y. Norouzzadeh Ravari, I. Markov, A. Grotov, M. Clements, and M. de Rijke. User behavior in location search on mobile devices. In *ECIR*, pages 728–733. Springer, April 2015.

[102] D. Odijk, E. Meij, I. Sijaranamual, and M. de Rijke. Dynamic query modeling for related content finding. In *SIGIR*, pages 33–42. ACM, 2015. (Cited on page 85.)

[103] N. O'Hare, P. de Juan, R. Schifanella, Y. He, D. Yin, and Y. Chang. Leveraging user interaction signals for web image search. In *SIGIR*, pages 559–568. ACM, 2016. (Cited on pages 51 and 67.)

[104] H. Oosterhuis and M. de Rijke. Ranking for relevance and display preferences in complex presentation layouts. *arXiv preprint arXiv:1805.02404*, 2018. (Cited on pages 49 and 90.)

[105] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped DQN. In *NIPS*, pages 4026–4034, 2016. (Cited on pages 8, 51, 55, 56, 68, and 69.)

[106] I. Osband, D. Russo, Z. Wen, and B. Van Roy. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017. (Cited on pages 68 and 69.)

[107] Y. Pan, T. Yao, T. Mei, H. Li, C.-W. Ngo, and Y. Rui. Click-through-based cross-view learning for image search. In *SIGIR*, pages 717–726. ACM, 2014. (Cited on page 67.)

[108] J. Y. Park, N. O'Hare, R. Schifanella, A. Jaimes, and C.-W. Chung. A large-scale study of user image search behavior on the web. In *CHI*, pages 985–994. ACM, 2015. (Cited on page 67.)

[109] O. Pietquin. Inverse reinforcement learning for interactive systems. In *Workshop on Machine Learning for Interactive Systems*, pages 71–75. ACM, 2013. (Cited on page 85.)

[110] R. L. Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975. (Cited on page 57.)

[111] H.-T. Pu. A comparative analysis of web image and textual queries. *Online Information Review*, 29(5): 457–467, 2005. (Cited on page 67.)

[112] 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. (Cited on pages 1, 8, 37, and 38.)

[113] F. Radlinski and N. Craswell. Comparing the sensitivity of information retrieval metrics. In *SIGIR*, pages 667–674. ACM, 2010. (Cited on page 26.)

[114] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, pages 784–791. ACM, 2008. (Cited on page 68.)

[115] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM*, pages 43–52. ACM, 2008. (Cited on pages 26, 28, and 37.)

[116] N. D. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009. (Cited on page 73.)

[117] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. (Cited on page 1.)

[118] S. Russell. Learning agents for uncertain environments. In *COLT*, pages 101–103. ACM, 1998. (Cited on page 73.)

[119] M. Sanderson. Test collection based evaluation of information retrieval systems. *Foundations and Trends in Information Retrieval*, 4(4):247–375, 2010. (Cited on pages 1, 26, 27, and 68.)

[120] 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 page 26.)

[121] 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, February 2016. (Cited on pages 5, 49, 50, and 68.)

[122] G. Shani, D. Heckerman, and R. I. Brafman. An MDP-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005. (Cited on page 84.)

[123] A. Shishkin, P. Zhinalieva, and K. Nikolaev. Quality-biased ranking for queries with commercial intent. In *WWW*, pages 1145–1148. ACM, 2013. (Cited on page 3.)

[124] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. (Cited on page 73.)

[125] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv preprint arXiv:1409.1556*, 2014. (Cited on pages 54, 60, and 67.)

[126] S. P. Singh, M. J. Kearns, D. J. Litman, and M. A. Walker. Reinforcement learning for spoken dialogue systems. In *NIPS*, pages 956–962, 2000. (Cited on page 3.)

[127] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at

the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12): 1349–1380, 2000. (Cited on page 67.)

[128] Statistica. Market capitalization of the biggest internet companies worldwide as of may 2017. https://www.statista.com/statistics/277483/market-value-of-the-largest-internet-companies-worldwide/, 2018. (Cited on page 1.)

[129] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, pages 216–224, 1990. (Cited on pages 8, 51, 55, and 68.)

[130] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998. (Cited on page 72.)

[131] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 2000. (Cited on page 58.)

[132] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, pages 2750–2759, 2017. (Cited on pages 68 and 69.)

[133] H. Tavani. Search engines and ethics, 2012. URL https://plato.stanford.edu/entries/ethics-search/. (Cited on page 1.)

[134] J. Teevan, S. T. Dumais, and E. Horvitz. Characterizing the value of personalizing search. In *SIGIR*, pages 757–758. ACM, 2007. (Cited on page 2.)

[135] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933. (Cited on pages 56 and 68.)

[136] D. Tjondronegoro, A. Spink, and B. J. Jansen. A study and comparison of multimedia web searching: 1997–2006. *Journal of the Association for Information Science and Technology*, 60(9):1756–1768, 2009. (Cited on page 67.)

[137] A. Turpin and F. Scholer. User performance versus precision measures for simple search tasks. In *SIGIR*, pages 11–18. ACM, 2006. (Cited on page 2.)

[138] 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 90.)

[139] C. Van Gysel, M. de Rijke, and M. Worring. Unsupervised, efficient and semantic expertise retrieval. In *WWW*, pages 1069–1079. International World Wide Web Conferences Steering Committee, 2016. (Cited on page 90.)

[140] E. M. Voorhees and D. K. Harman, editors. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005. (Cited on pages 2 and 50.)

[141] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016. (Cited on page 73.)

[142] L. Weaver and N. Tao. The optimal reward baseline for gradient-based reinforcement learning. In *UAI*, pages 538–545. Morgan Kaufmann Publishers Inc., 2001. (Cited on page 59.)

[143] H. Wei, F. Zhang, N. J. Yuan, C. Cao, H. Fu, X. Xie, Y. Rui, and W.-Y. Ma. Beyond the words: Predicting user personality from heterogeneous information. In *WSDM*, pages 305–314. ACM, 2017. (Cited on page 72.)

[144] 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 page 56.)

[145] K. Williams, J. Kiseleva, A. C. Crook, I. Zitouni, A. H. Awadallah, and M. Khabsa. Detecting good abandonment in mobile search. In *WWW*, pages 495–505, 2016. (Cited on page 71.)

[146] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. (Cited on pages 73 and 89.)

[147] X. Xie, Y. Liu, X. Wang, M. Wang, Z. Wu, Y. Wu, M. Zhang, and S. Ma. Investigating examination behavior of image search users. In *SIGIR*, pages 275–284. ACM, 2017. (Cited on page 67.)

[148] X. Xie, Y. Liu, M. de Rijke, J. He, M. Zhang, and S. Ma. Why people search for images using web search engines. In *WSDM*, pages 655–663. ACM, February 2018. (Cited on page 67.)

[149] J. Xu and H. Li. Adarank: A boosting algorithm for information retrieval. In *SIGIR*, pages 391–398, New York, NY, USA, 2007. ACM. (Cited on page 38.)

[150] Yandex. Relevance prediction using user behaviour. https://academy.yandex.ru/events/data_analysis/relpred2011/, 2011. (Cited on page 2.)

[151] E. Yilmaz, M. Shokouhi, N. Craswell, and S. Robertson. Expected browsing utility for web search

evaluation. In *CIKM*, pages 1561–1564. ACM, 2010. (Cited on pages 1, 5, 28, and 30.)

[152] E. Yilmaz, M. Verma, N. Craswell, F. Radlinski, and P. Bailey. Relevance and effort: an analysis of document utility. In *CIKM*, pages 91–100. ACM, 2014. (Cited on page 37.)

[153] J. Yu, D. Tao, M. Wang, and Y. Rui. Learning to rank using user clicks and visual features for image retrieval. *IEEE Transactions on Cybernetics*, 45(4):767–779, April 2015. (Cited on page 67.)

[154] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*, pages 1201–1208. ACM, 2009. (Cited on pages 39, 47, 50, and 68.)

[155] Y. Zhang, D. Wang, G. Wang, W. Chen, Z. Zhang, B. Hu, and L. Zhang. Learning click models via probit Bayesian inference. In *CIKM*, pages 439–448. ACM, 2010. (Cited on page 90.)

[156] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, pages 3357–3364. IEEE, 2017. (Cited on page 73.)

[157] B. Ziebart, A. Dey, and J. A. Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *IUI*, pages 1–10. ACM, 2012. (Cited on page 85.)

[158] B. D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, 2010. (Cited on page 79.)

[159] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438. AAAI Press, 2008. (Cited on pages 73 and 76.)

[160] M. Zoghi, T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, and Z. Wen. Online learning to rank in stochastic click models. In *ICML*, pages 4199–4208, 2017. (Cited on pages 5 and 68.)

# Summary

Web search engines are used by many people everyday, serving as one of the primary gateways to information stored online. Optimizing search engines is challenging because it requires large datasets annotated by human judges. Such datasets are expensive to create and are often not reliable because there can be a mismatch between what human judges and real users find relevant. This motivates using observed interactions between the user and the web search engine. These interactions are readily available in massive quantities and have been shown to correlate with user satisfaction with the search engine.

In this thesis we study how to use these interactions to optimize web search engines. In the first research chapter we look at modeling clicks on a search engine result page using several click models; we compare the ability of these click models to explain and predict clicks, predict annotated relevance and improve rankings. In the second research chapter we derive a Bayesian inference method for the DBN click model and use it to compare search engine rankers to each other using a fixed interaction log. We compare the proposed Bayesian approach with the traditional Expectation Maximization inference. We find that the confidence estimates in the Bayesian approach help to decide if the logged interactions are enough to conclude that one ranker is better than another one. In the third research chapter we use deep neural networks to perform online learning to rank. We propose and compare two loss functions and compare several exploration strategies on the image filtering task. We find that while both loss functions are theoretically sound only one of them works in practice. In the final research chapter we address the scenario when we do not know what metric we want to optimize. We propose a new algorithm that first recovers the user satisfaction metric using Inverse Reinforcement Learning and then optimizes it using Reinforcement Learning. We find that it successfully recovers user rewards and can optimize the system resulting in a better user experience.

We have found that interactions are useful for optimizing search engines. User interactions must be used cautiously because they are biased. They are biased because users prefer to interact with the results that rank high on the result page and because the search engine displays some results more often than others. However, when used correctly they can substitute annotated datasets annotated by human judges. The observed user interactions can also be used to automatically create optimization objectives for interactive systems. This thesis advanced our understanding of how to use interactions to optimize web search engines and resulted in development of several algorithms that can be used in practice to optimize real world systems.

# Samenvatting

Internetzoekmachines worden iedere dag gebruikt door vele mensen. Ze dienen als één van de belangrijkste toegangspoorten tot online informatie. Het is lastig om deze zoekmachines te optimaliseren, aangezien er grote datasets, geannoteerd door mensen, voor nodig zijn. Het is duur om dit soort datasets te maken en bovendien zijn de uiteindelijke datasets vaak onbetrouwbaar aangezien er een verschil kan zijn tussen wat de annotatoren belangrijk vinden en wat echte gebruikers belangrijk vinden.

Het gebruik van geobserveerde interacties tussen gebruikers en zoekmachines zou kunnen helpen. Deze interacties zijn grootschalig beschikbaar en correleren met de tevredenheid van gebruikers over de zoekmachines.

In dit proefschrift onderzoeken we hoe we deze interacties kunnen gebruiken om internetzoekmachines te optimaliseren. In het eerste hoofdstuk kijken we naar het modelleren van kliks op de resultatenpagina van een zoekmachine. Hiervoor gebruiken we verschillende klikmodellen. We vergelijken hoe deze klikmodellen kliks kunnen uitleggen en voorspellen, hoe ze geannoteerde relevantie kunnen voorspellen en hoe ze rankings kunnen verbeteren.

In het tweede onderzoekshoofdstuk leiden we een Bayesiaanse inferentie methode af voor het DBN klikmodel en we gebruiken deze methode om rankings van zoekmachines te vergelijken met behulp van een gegeven log met interacties. We vergelijken de voorgestelde Bayesiaanse aanpak met de traditionele Expectation Maximization inference. We vinden dat de betrouwbaarheidsschattingen in de Bayesiaanse aanpak helpen om te beslissen of de gelogde interacties genoeg zijn om te concluderen dat de ene ranker beter is dan de andere.

In het derde onderzoekshoofdstuk gebruiken we diepe neurale netwerken om de *online learning to rank* taak uit te voeren. We stellen twee *loss* functies voor en vergelijken deze twee. Ook vergelijken we verschillende exploratie strategieën voor het filteren van afbeeldingen. Hoewel allebei de *loss* functies theoretisch correct zijn, vinden we dat er in de praktijk slechts één werkt.

In het laatste onderzoekshoofdstuk bekijken we het scenario waarin we niet weten welke metriek we willen optimaliseren. We stellen een nieuw algoritme voor dat eerst de gebruikerstevredenheid metriek construeert. Hiervoor gebruiken we *inverse reinforcement learning*. Vervolgens optimaliseert het algoritme de metriek met *reinforcement learning*. We vinden dat het algoritme de gebruikers beloningen correct weet te construeren en dat het algoritme het systeem kan optimaliseren. Dit resulteert in een betere gebruikerstevredenheid.

Dit proefschrift heeft bijgedragen aan ons begrip van hoe interacties gebruikt kunnen worden om internetzoekmachines te optimaliseren en heeft geresulteerd in de ontwikkeling van verschillende algoritmes die gebruikt kunnen worden om systemen uit de praktijk te optimaliseren.