

LEARNING TO RANK FROM IMPLICIT FEEDBACK

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Filip Andrzej Radliński

August 2008

© 2008 Filip Andrzej Radliński

ALL RIGHTS RESERVED

LEARNING TO RANK FROM IMPLICIT FEEDBACK

Filip Andrzej Radliński, Ph.D.

Cornell University 2008

Whenever access to information is mediated by a computer, we can easily record how users respond to the information with which they are presented. These normal interactions between users and information systems are *implicit feedback*. The key question we address is – how can we use implicit feedback to automatically improve interactive information systems, such as desktop search and Web search?

Contrasting with data collected from external experts, which is assumed as input in most previous research on optimizing interactive information systems, implicit feedback gives more accurate and up-to-date data about the needs of actual users. While another alternative is to ask users for feedback directly, implicit feedback collects data from all users, and does not require them to change how they interact with information systems. What makes learning from implicit feedback challenging, is that the behavior of people using interactive information systems is strongly biased in several ways. These biases can obscure the useful information present, and make standard machine learning approaches less effective.

This thesis shows that implicit feedback provides a tremendous amount of practical information for learning to rank, making four key contributions. First, we demonstrate that query reformulations can be interpreted to provide relevance information about documents that are presented to users. Second, we describe an experiment design that provably avoids presentation bias, which

is otherwise present when recording implicit feedback. Third, we present a Bayesian method for collecting more useful implicit feedback for learning to rank, by actively selecting rankings to show in anticipation of user responses. Fourth, we show how to learn rankings that resolve query ambiguity using multi-armed bandits. Taken together, these contributions reinforce the value of implicit feedback, and present new ways it can be exploited.

BIOGRAPHICAL SKETCH

Filip Radliński was born in Warsaw, Poland. He grew up in Canberra, Australia, where he immigrated at a young age. He completed his Bachelor of Science with Honours degree at the Australian National University in Canberra, Australia in 2002. During his Bachelor's degree, he spent a year as an international exchange student at the Pennsylvania State University in State College, Pennsylvania, United States. He received a Master of Science degree from Cornell University in Ithaca, New York, United States in 2006.

His awards include a university medal received from the Australian National University in 2002, a Fulbright Fellowship in 2002, a best student paper award at the ACM Conference on Knowledge Discovery and Data Mining in 2005 and a Microsoft Research Fellowship in 2006.

To my mother.

ACKNOWLEDGMENTS

I would like to thank my advisor, Thorsten Joachims, for his invaluable guidance, for always making time to talk, and encouraging me to tackle challenging problems. Many of the ideas explored in this thesis are the result of discussions with him. I also owe special thanks to Robert Kleinberg for his advice, always insightful observations, and his careful reading of this thesis.

I also thank all my other collaborators and co-authors, especially Susan Dumais, Vincent Crespi and Eric Loken. I have learned a lot working with them. I am also grateful to Simeon Warner, Paul Ginsparg and Paul Houle for providing me access to data, and support in running the search systems upon which much of the evaluation in this thesis rests.

I express my gratitude to the other faculty members of the department of Computer Science at Cornell University, in particular to Rich Caruana, Johannes Gehrke, Joe Halpern, Dan Huttenlocher and Dexter Kozen, for their support and advice. I also thank the administrative staff and particularly Becky Stewart, Stephanie Meik and Melissa Totman for their help whenever it was needed.

I extend special thanks to my friends and fellow graduate students nearest my research area, Eric Breck, Tom Finley, Art Munson, Alex Niculescu-Mizil and Yisong Yue. They were constant sounding boards for ideas, enormously helpful when learning new concepts, and always ready to distract me from too much work when necessary.

I am very grateful to Megan Owen for her wonderful friendship, encouragement and many kind words. I also thank Sam Arbesman, Jill Chavez, Saggi Cherem, Lauren Childs, Steve Chong, Yashoda Dadkar, Russell de Vries, David Hellier, Katy Munson, Nate Nystrom, Matt Schultz, Daria Sorokina, Kevin Walsh as well as many of the other graduate students at Cornell University, for being

great friends throughout. I thank the Computer Science and Friends ice hockey team for getting me away from my work.

I thank my family for instilling in me a passion for research, for their continual support and for their unending patience. Above all, my warm thanks to my mother, Emilka, Basia and Campbell.

Finally, I would like to acknowledge that financial support for the work presented in this thesis came from a Fulbright Fellowship, a Microsoft Research Graduate Student Fellowship, NSF CAREER Award IIS-0237381, the KD-D grant, a research gift from Google, Inc, and the Department of Computer Science at Cornell University.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgments	v
Table of Contents	vii
List of Tables	x
List of Figures	xi
List of Definitions	xiii
List of Algorithms	xiv
1 Introduction	1
1.1 Overview	1
1.2 Obtaining Relevance Information	4
1.3 Relevance Judgments from Users	8
1.3.1 Explicit Feedback	8
1.3.2 Implicit Absolute Feedback	9
1.3.3 Implicit Preference Feedback	11
1.3.4 Preferences from Query Chains	12
1.4 Clicks, Bias, Diversity and Noise	14
1.5 Presentation Bias and Learning Convergence	16
1.6 Combatting Evaluation Bias	17
1.7 Addressing User Diversity	19
1.8 Bibliographic Notes	21
2 Understanding and Interpreting Users' Decisions	23
2.1 Offline Decision Making	23
2.1.1 What influences user decisions?	24
2.1.2 Are decisions absolute or relative?	25
2.2 Online Decision Making	25
2.2.1 What can we observe of online user behavior?	26
2.2.2 Which search results do users consider?	27
2.2.3 Which results do users consider before clicking?	29
2.2.4 How can we infer judgments from clicks?	30
2.2.5 Implicit Feedback as Relative Relevance Judgments	33
2.2.6 Are these judgments valid?	35
2.3 Summary	37
3 Learning to Rank	38
3.1 Performance Metrics	38
3.1.1 Precision and Recall	38
3.1.2 Ranking Based Metrics	40
3.1.3 Performance with Relative Judgments	42
3.1.4 Comparing Two Rankings	43

3.2	Algorithms for Learning to Rank	46
3.2.1	Ranking as a Classification Problem	46
3.2.2	Ranking as Ordinal Regression	48
3.2.3	Ranking as Regression	49
3.2.4	Learning to Rank with Pairwise Data	49
3.2.5	Learning from Entire Rankings	51
3.3	Two Particular Learning to Rank Algorithms	52
3.3.1	Ranking SVMs	53
3.3.2	Tournament Participant Ranking	55
3.4	Summary	57
4	Learning from Implicit Feedback Encoded in Query Chains	58
4.1	Introduction	58
4.2	Related Work	60
4.3	Analysis of User Behavior	62
4.4	Implicit Feedback Strategies	64
4.5	Detecting Query Chains	68
4.6	Accuracy of the Feedback Strategies	71
4.7	Learning Ranking Functions	74
4.7.1	Ranking SVMs	75
4.7.2	Retrieval Function Model	76
4.8	Adding Prior Knowledge	79
4.9	Evaluation	81
4.9.1	Interleaved Evaluation	83
4.9.2	Results and Discussion	84
4.10	Summary	89
5	Avoiding Bias in Implicit Feedback	90
5.1	Introduction	90
5.2	Presentation Bias	93
5.3	Bias-Free Feedback	95
5.4	Theoretical Analysis	98
5.5	Practical Considerations	101
5.6	Learning Convergence	103
5.7	Experimental Results	105
5.7.1	Item Relevance Score	106
5.7.2	Ignored Relevance Score	108
5.7.3	FairPairs Preference Test	109
5.8	Summary	111
6	Active Methods for Optimizing Data Collection	113
6.1	Introduction	113
6.2	Formalizing the Learning Problem	115
6.2.1	Probabilistic Model	116

6.2.2	Inference	117
6.2.3	Loss Function	120
6.2.4	Estimating the Model Parameters	121
6.3	Exploration Strategies	123
6.4	Evaluation Methodology	127
6.5	Results	129
6.5.1	Synthetic Data	130
6.5.2	TREC Data	133
6.5.3	Controlling for Presentation Loss	140
6.6	Summary	144
7	Optimizing Rankings with Dependent Document Relevances	145
7.1	Introduction	145
7.2	Related Work	147
7.3	Problem Formalization	150
7.4	Learning Algorithms	152
7.4.1	Ranked Explore and Commit	152
7.4.2	Ranked Bandits Algorithm	153
7.5	Theoretical Analysis	156
7.5.1	The Offline Optimization Problem	156
7.5.2	Analysis of Ranked Bandits Algorithm	157
7.5.3	Analysis of Ranked Explore and Commit	161
7.6	Evaluation	163
7.6.1	Performance Without Click Noise	164
7.6.2	Effect of Click Noise	165
7.6.3	Practical Considerations	165
7.7	Summary	168
8	Conclusions and Open Questions	169
8.1	Thesis Conclusions	169
8.2	Open Questions and Future Directions	172
8.2.1	Personalization	172
8.2.2	Malicious Noise	177
8.2.3	Scalability and Generalizability	185
	Bibliography	187

LIST OF TABLES

2.1	Summary of the feedback strategies discussed by Joachims et al. (2005).	34
2.2	Accuracy of feedback strategies discussed by Joachims et al. (2005).	36
4.1	Features used to learn to classify query chains.	70
4.2	Accuracy of the strategies for generating pairwise preferences from clicks.	72
4.3	Evaluation results on the Cornell University library search engine.	85
4.4	The most common words to appear in queries in the training data, and the fraction of queries in which they occur.	87
4.5	Five most positive and most negative feature weights in the ranking function learned using query chains on the Cornell University Library search engine	88
5.1	Results from a user study using the Google search engine presented by Joachims et al. (2005).	94
6.1	Mean Average Precision after 3000 iterations of optimizing different loss variants using OSL.	139

LIST OF FIGURES

1.1	Example of a TREC information need.	5
2.1	Percentage of time an abstract was viewed/clicked on depending on the rank of the result.	29
2.2	Mean number of abstracts viewed above and below a clicked link depending on its rank.	30
2.3	Fraction of results looked at, and clicked on, by users presented with standard and modified Google results.	32
3.1	Example interleaving of two rankings.	45
3.2	The Glicko update equations (Glickman, 1999).	56
4.1	Percentage of time an abstract was viewed/clicked on depending on the rank of the result.	63
4.2	Two example queries and result sets.	65
4.3	Strategies used to infer relevance judgments from implicit feedback.	66
4.4	Sample query chain and the feedback that would be generated using all feedback strategies.	68
4.5	Example interleaving of two rankings.	83
5.1	Feedback strategies from Chapter 4.	91
5.2	Click probability measurements of the Item Relevance Score.	107
5.3	Click probability measurements of the Ignored Relevance Score.	108
5.4	Evaluation of the relative relevance of search results returned by the arXiv search engine.	109
5.5	Probability of user clicking only on the bottom result of a pair as a function of the pair.	110
5.6	Probability of user clicking only on the bottom result of a pair as a function of the pair for all queries generating at least one user click.	110
6.1	Example of how we can maintain estimates of the relevance of documents to a fixed query.	116
6.2	The Glicko update equations (Glickman, 1999).	122
6.3	Change in loss as a function of the number of pairwise comparisons for each exploration strategy on synthetic data.	131
6.4	Effect of the weight given to the prior on the final loss, evaluated on synthetic data.	132
6.5	Change in loss as a function of the number of pairwise comparisons for each selection algorithm on TREC-10 data.	134
6.6	Change in MAP score as a function of the number of pairwise comparisons for each selection algorithm on TREC-10 data.	136

6.7	Effect of different noise levels in pairwise preferences on final MAP score, evaluated on TREC-10 data.	137
6.8	Effect of incorrect assumptions about the noise level in relevance judgments on final MAP score, evaluated on TREC-10 data using OSL.	139
6.9	MAP scores of the mode rankings and the presented rankings, as a function of number of pairwise comparisons for OSL and RANDOM.	141
6.10	MAP scores of the presented ranking after 3,000 pairwise comparisons after presenting selected pairs at different ranks.	143
6.11	MAP scores of the mode and presented rankings after 3,000 pairwise comparisons.	143
7.1	Example of users and relevant documents for some query.	151
7.2	Clickthrough rate of the learned ranking as a function of the number of times the ranking was presented to users.	164
7.3	Effect of noise in clicking behavior on the quality of the learned ranking.	166
7.4	Performance of RBA, REC and two RBA variants.	167
8.1	Evaluation of MF, MRV and MS diversity methods on a fixed query set, as well as on queries taken from users' Web browser cache.	177
8.2	User utility as a function of threshold when $\mu_B = 1$ and $\sigma_B = 2$. $\mu_A = 0, \sigma_A = 1$ and $a = 25$	183

LIST OF DEFINITIONS

1.1	Implicit Feedback	9
1.2	Clickthrough Data	10
1.3	Query Chain	13
1.4	Presentation Bias	14
1.5	Evaluation Bias	14
1.6	User Diversity	15
1.7	Click Noise	15
3.1	Precision	39
3.2	Recall	39
3.4	Mean Average Precision	41
7.1	Abandonment	151

LIST OF ALGORITHMS

5.1	FairPairs Algorithm.	96
6.1	Evaluation simulation for active ranking.	129
6.2	Model of user behavior.	129
7.1	Ranked Explore and Commit Algorithm.	153
7.2	Ranked Bandits Algorithm.	154

CHAPTER 1

INTRODUCTION

1.1 Overview

A tremendous number of interactive information ranking systems are available on the Internet, and on desktop computers everywhere. Web search engines rank Web documents in response to user queries, shopping sites rank products a user may wish to purchase, movie rental sites suggest movies, email clients rank email messages, desktop search applications assist in finding files, and many community websites rank everything from restaurants to photos to romantic matches. In general, the goal of interactive information ranking is to present a ranked list of results ordered such that the highest ranked results are those most related to a query or user profile provided.

The simplest approach of manually constructing a function that produces this ranking is difficult and time consuming, with diminishing returns in terms of improvement with increased effort. One of the main reasons for this is the sheer number of possible functions that could be used to rank results. Selecting the best parameterization, and then picking the best parameter settings, is simply too large a task to solve optimally by hand. This problem is exacerbated when we would like to use the same ranking system to serve many different people with different goals. For instance, the perfect function for ranking Web documents for academic users at Cornell University is not necessarily the same as the perfect one for teenagers in Japan. On a more fine grained level, it may well be the case that the best function for ranking Web documents for one Cornell researcher

is not best for another. In response, the machine learning community started addressing the question of how to optimize ranking functions automatically using machine learning techniques. For example, Cohen et al. (1999), Freund et al. (2003) and Burges et al. (2005), among many others, have addressed this question using a variety of approaches that will be discussed later in this thesis.

All techniques for learning to rank require two essential pieces of information: *training data*, which provides examples for the learning algorithm as to what distinguishes good results from poor results, and an *error metric* that the algorithm optimizes relative to this training data. Most previous research in learning to rank has assumed a supervised learning setting where training data is provided by some offline mechanism. Such data is often obtained by paying expert relevance judges to provide it, for instance presenting them with a sequence of recorded search queries and Web documents. The role of the judge is to *guess* the users' intentions based on the query issued, and provide an appropriate graded relevance score such as *very relevant* or *somewhat relevant* for each document assessed. However, judgments collected from users would be preferable, as they would reflect the users' true needs, and be much cheaper and faster to collect. With respect to error metrics, most algorithms optimize metrics that aggregate over the judgments made for (query, result) pairs, assessing how well the rankings produced by the learned ranking function agree with the judgments provided by the experts. Again, it would be preferable for error metrics to instead reflect the experiences of interactive information ranking system users.

This thesis extends research in learning to rank with four primary contributions. First, we demonstrate that query reformulation in Web search provides extremely informative relevance information that reflects *users'* needs. This

builds upon a technique proposed by Cohen et al. (1999) and Joachims (2002), but collects substantially more relevance judgments. Moreover, these judgments are often also more useful for learning algorithms. Second, we show that the way in which users interact with Web search systems means that standard learning algorithms trained with data collected from user interactions, using previous approaches, will never converge to a fixed ranking. We present a technique for modifying the rankings shown to users in a controlled manner, and show that it provably corrects for this problem. Third, most previous work in learning to rank has not considered that there is a natural tradeoff when learning from user interactions. While in the short run presenting the best known results provides users with the best rankings, in the long run exploration of unknown results may improve average performance. We will demonstrate that by using directed exploration, the rankings learned can improve much more rapidly. Fourth, most previous algorithms for learning to rank optimize error metrics that measure performance with respect to relevance judgments provided by experts. We describe two principled algorithms that instead optimize abandonment, a performance metric that directly reflects desirable user behavior, and present formal performance guarantees.

It is important to note, that although we use ranking for Web search as the canonical interactive information ranking task in the remainder of this thesis, ranking is also a fundamental goal of many other real world applications. Some of these were enumerated in the first paragraph of the introduction. Other applications studied in the research literature range from predicting consumer food preferences (Luaces et al, 2004) to assisting astronomers in devising schedules that optimally use limited telescope time (Branting & Broos, 1997). The machine learning community has also addressed ranking questions as diverse as ranking

people or teams based on the outcome of two-player (Herbrich & Graepel, 2006) or two-team games (Huang et al, 2004), ranking universities by various criteria (Dittrich et al, 1998) and ranking patients by their risk of developing pneumonia (Caruana et al, 1995). In all these settings, user interactions with ranking systems can implicitly provide training data and evaluation opportunities for improving the rankings produced.

Similarly, interpreting user behavior as implicit feedback is not limited to information retrieval settings. Any computer system or mobile computing platform can collect implicit feedback from users, be there a small or large number of them. While not addressed in this thesis, implicit feedback could be used in settings as varied as designing better software interfaces or measuring social phenomena.

1.2 Obtaining Relevance Information

The first step for any machine learning task is to obtain training data, to which we can then apply a particular algorithm. We will now consider how such data is usually obtained when we want to learn to rank Web documents.

In an academic setting, the largest public data collection suitable for learning to rank is derived from an annual evaluation run as part of the Text REtrieval Conference (TREC). The purpose of the evaluation is to compare the performance of competing search systems, given particular information needs. An example of the information needs provided in TREC evaluations is shown in Figure 1.1.

Topic Number	503
Title	Vikings in Scotland?
Description	What hard evidence proves that the Vikings visited or lived in Scotland?
Narrative	A document that merely states that the Vikings visited or lived in Scotland is not relevant. A relevant document must mention the source of the information, such as relics, sagas, runes or other records from those times.

Figure 1.1: Example of a TREC information need.

For each information need, each competing system must automatically transform the request into a query, and return documents from a fixed document collection. The top n results returned by any retrieval system taking part in the evaluation are then manually judged for relevance. Human judges rate each selected document as not relevant, relevant or highly relevant (Voorhees, 2004). All documents not ranked in the top n by any competing system are assumed not relevant. The dataset consisting of the information needs, documents and document judgments is then made publicly available.

However, TREC data is not representative of the data necessary for learning to rank in many interactive information ranking settings, including the ranking of Web documents. In particular, rather than a long description of an information need, the input to a Web search system is usually a short query. Yet, translating the TREC approach, the most common way that relevance judgments for training a Web search system are obtained is by providing human experts with (query, document) pairs. The experts then provide graded relevance judgments. The job of an expert is to understand each query he or she is presented with, and consider the possible relevance of the presented document to the information need that likely motivated a user to enter that query. The expert then needs to specify to what extent the document satisfies this inferred information need.

Clearly, this task is difficult and slow, requiring many experts to produce a meaningful amount of training data. For the judgments to be consistent across different experts and a wide variety of queries and documents, the experts must be trained and provided with extensive documentation as well as a detailed relevance scale.

This approach is often used by researchers at large search engine companies. For example Burges et al. (2005) from Microsoft, as well as Jones et al. (2006) from Yahoo!, assume expert labeled data is available. However, there are at least three difficulties present when learning for Web search that are absent in the controlled TREC setting. First, as obtaining judgments is time consuming, judgments can only be obtained for a minute fraction of typical queries and a handful of the billions of documents on the Internet. This brings up the the question of how the documents and queries to judge should be selected, so as to have the largest eventual effect on future users' search satisfaction. Moreover, as Web users' needs and the documents available change constantly, how should expert judgments be updated to stay representative of real search tasks?

Second, as the judged queries are usually drawn from those issued by actual users, how is an expert judge to know the intent of the users who entered the queries? This is particularly difficult because, in contrast to TREC, most Web queries are too short to unambiguously identify the users' information needs. Typical Web queries are only two or three words long (Silverstein et al, 1998; Spink et al, 2001; Zhang & Moffat, 2006). Additionally, many queries have multiple valid meanings, with the "correct" one dependent on the user who issued the query. Canonical examples of ambiguous queries include *jaguar* (which can refer to (i) a car, (ii) an animal, or (iii) an operating system), *Michael*

Jordan (which can refer to (i) a basketball player, (ii) sports clothing that bears his brand or (iii) a professor at the University of California, Berkeley) and *flash* (which can refer to (i) a photography product, (ii) a popular file format or (iii) a superhero). There is no easy way for a judge to work out the relative importance of each meaning across the population of all users. Even seemingly unambiguous queries can mean different things to different people. For instance, the clearly machine learning query *support vector machine* might be issued by researchers searching for downloadable software, for a tutorial about the algorithm or for theoretical performance bounds. These different needs are likely to be satisfied by different documents.

Third, different human judges may have different opinions concerning the relevance of a particular document to a particular query, even when the user intent is clear. For instance, one judge may not trust anything printed by a left-leaning newspaper, while another may consider blogs as much less authoritative. For these disagreements not to cause difficulties requires substantial training of judges, and long and detailed definitions of the judgment scale. In particular, it is difficult to trade off between obtaining judgments with sufficient granularity to be useful in practice, and obtaining reproducible judgments. Agreement between different judges providing relevance scores for the same (query, document) pair, usually termed *inter-judge agreement*, is often less than ideal. Although specific numbers depend on the granularity of judgments, some statistics from real search engine judges were recently provided by Carterette et al. (2008).

While there has been substantial effort in improving expert-labeled collections (for example, Reid (2000) presented a partial overview), the four key contributions in this thesis will address an alternative method to circumvent these

difficulties: obtain relevance judgments directly from users, without involving expert human judges.

1.3 Relevance Judgments from Users

Since obtaining relevance judgments from human experts is fraught with difficulties, the alternative is to obtain relevance judgments directly from users. We now look at how to get relevance information from users.

1.3.1 Explicit Feedback

One obvious approach to obtaining training data from users is to solicit data by posing users explicit questions: Ask them whether or not specific documents are relevant. While an explicit feedback approach is commonly used when learning to recommend movies, for instance by Crammer and Singer (2001), Herbrich et al. (2000) and Rajaram et al. (2003), Web users are generally not willing to provide such explicit feedback. In particular, as judging documents is onerous, users cannot be expected to provide relevance judgments for each result presented or even each result clicked on in a Web ranking setting. In fact, search engines that attempted to add relevance judgment buttons to search results have not been successful. Moreover, given the option to provide such relevant judgments, malicious users would have much more incentive to provide judgments (promoting Web pages that should not be ranked highly) than regular Web users.

1.3.2 Implicit Absolute Feedback

Instead of explicitly asking users for relevance feedback, we could alternatively track *normal* user interactions with a interactive information ranking system. Consider that, when using an online interface, users usually perform actions as a result of the rankings they are presented with. These actions can be used to implicitly infer relevance judgments. In a movie task, this might be done by observing which movies users search for and then watch or perhaps buy. In the Web search setting, this can be done by extracting implicit relevance feedback from search engine log files, or by recording actions users perform in their Web browsers.

We define implicit feedback as follows:

Definition 1.1. *Implicit feedback is information that can be obtained by analyzing the normal interactions of a user with an online system. These interactions include any input the user provides, the information that is shown to the user in response, and all the user's online actions in response to being presented with this information.*

For example, in a Web search setting, implicit feedback may include the user query, any results clicked on, the timing of the clicks, further input provided by the user to the search system, the choice to no longer use the search system, or even bookmarking or printing a website. Most importantly, implicit feedback reflects the judgments of all the users of the search engine rather than a select group of paid judges. In addition, due to the scale at which search engines operate, this usually provides as much data as can be practically exploited, and does so at almost no cost.

Implicit feedback that simply records clicks on Web search results (also commonly called *clickthrough data*) is most easily observed by Web retrieval systems. Search engines typically collect clickthrough data by incorporating a redirect into all links on the results page presented to users¹. While a number of researchers, including Kelly and Teevan (2003), Fox et al. (2005) and White et al. (2005), have considered data describing from other behavioral cues, such as bookmarking or scrolling behavior, we focus on clickthrough data. In particular, it effectively captures user intent while being most easily collected.

Definition 1.2. Clickthrough data is *implicit feedback obtained by recording the queries users run on a search engine as well as the results they click on.*

The interpretation of clickthrough data as relevance judgments that would most closely mirror relevance judgments collected from experts is in terms of absolute statements about the relevance of particular documents to particular queries. Indeed, early work in learning to rank took such an approach, for instance by Wong et al. (1988) and Bartell et al. (1994). Many researchers followed in this interpretation of *absolute relevance judgments* implicitly collected from clickthrough data, including Boyan et al. (1996); Cohen et al. (1999); Kemp and Ramamohanarao (2002); Cui et al. (2002); Tan et al. (2004); Dou et al. (2007). Such work usually assumes that documents clicked on in search results are highly likely to be relevant. For example, Kemp and Ramamohanarao (2002) assume results clicked on are relevant to the query and append the query to these documents to make them more likely to be ranked highly in the future. Similarly, Dou et al. (2007) propose reordering search results based on the frequency with which returned results are clicked on.

¹Alternative methods also exist, for instance using JavaScript event actions or a browser extension or add-on that some users install.

Yet, as we will see in Chapter 2, a user clicking on a result does not always indicate that the result is relevant. For example, Boyan et al. (1996) constructed three different ranking functions with very different performance, yet saw that the average rank at which users click did not differ meaningfully between the better and worse rankings. Given that the worse ranking functions ranked relevant results lower, we would have expected the rank of the first click to be lower for the poorer ranking functions. One approach to correct for biases in clicking behavior would be to model user clicking behavior and compensate for clicks on non-relevant documents using methods proposed by Dupret et al. (2007) or Carterette and Jones (2007). However, we will see that there is a simpler process by which reliable relevance judgments can be collected: interpreting clicks as *relative relevance judgments*.

1.3.3 Implicit Preference Feedback

Cohen et al. (1999) suggested an alternative interpretation of clickthrough data, that instead of inferring absolute judgments from implicit feedback, we can interpret a click as a relative preference. Specifically, they suggested that clicked on documents are likely better than higher ranked documents that were not clicked on. Joachims (2002) formalized and evaluated this idea and found it to work well, learning an improved search engine ranking function specifically for a small number of German machine learning researchers using their clicking behavior.

In fact, Joachims et al. (2005; 2007) demonstrated in a laboratory study that clickthrough data is strongly biased by the position at which results are presented.

They showed that if a search result is moved higher in a presented result set, this immediately increases the expected number of clicks it will receive, even if the result is not relevant to the query. Moreover, if the top ten documents retrieved by a search engine are presented in reverse order, non-relevant documents are clicked more often. This is at odds with an absolute interpretation of implicit feedback, as presentation effects will strongly influence the relevance judgments obtained. On the other hand, their study confirmed that judgments can be validly interpreted as relative statements of the form that one document is more relevant than another based on clicking behavior. We will discuss these studies in depth in Chapter 2.

1.3.4 Preferences from Query Chains

While implicit feedback interpreted as preferences gives rise to reliable relevance judgments that reflect user needs, previous work using this interpretation is still limited in the relevance feedback received. It is commonly known that search engine users predominantly click only on top ranked results. Granka et al. (2004) partially explained this effect through an eye tracking study, observing that most users do not even look at documents below the top few. Now, consider a user who enters a query for which the search engine performs particularly poorly, not retrieving any relevant documents in the top few positions. It is very unlikely that the user will scroll down to a truly relevant document and click on it. Rather, almost all users will either not click, or click on a highly ranked yet irrelevant document. Thus, any learning algorithm would have difficulty learning an improved ranking function, given that the training data is unlikely to provide any preferences identifying relevant documents.

However, many search users will *reformulate* poorly performing queries. The first of the four key contributions of this thesis is to study how to generate relevance judgments from implicit feedback collected over sequences of multiple queries reflecting the same information need. Such sequences are termed *query chains*.

Definition 1.3. *A query chain is a sequence of queries issued by a user over a short period of time with a constant information need in mind.*

We will show in Chapter 4 that collecting relevance judgments by considering query chains leads to significant improvements in search engine performance. Moreover, the reliability of the relevance judgments obtained from query chains is on par with the reliability of explicit judgments from expert judges.

Outside of a Web search setting, Furnas (1985) first proposed the use of learning from reformulations. In particular, he considered the task of learning new command names on a command line system by recording which commands users tried then reformulated. He interpreted two commands being seen in sequence to mean that the intended result of the first command is the same as the intended result of the second command. Similarly, Cucerzan and Brill (2004) looked at using reformulations to learn spelling corrections for Web queries. However previous work has not considered learning general functions using information inferred from reformulations, instead focussing on learning specific corrections that can be made.

1.4 Clicks, Bias, Diversity and Noise

As this thesis considers using records of user behavior to infer relevance judgments, we are limited by the noise and bias that is always present in real world data. The signal in clickthrough data is inherently masked in at least four ways, which must all be considered when relying on data from search engine log files. We now introduce these effects, and will periodically return to them throughout this thesis.

Definition 1.4. Presentation bias *is manifested when users preferentially click on higher ranked results, irrespective of relevance.*

Presentation bias is usually seen when particular search results move up or down an otherwise fixed ranking, and in consequence receive a vastly different number of clicks. As described in the previous section, one effect of presentation bias is that absolute relevance judgments tend to be difficult to collect from clickthrough data. While inferring relative relevance judgments from clickthrough data avoids this difficulty, there is also bias in the preferences that can possibly be collected, as described in Section 1.5 below.

Definition 1.5. Evaluation bias *is the bias exhibited by users to preferentially look at and evaluate highly ranked documents. The effect is that clickthrough data obtained from search engine logs predominantly describes the relevance of documents already ranked highly by a search engine.*

There is an important albeit subtle distinction between evaluation bias and presentation bias. To see this, consider that it could well be that some users consider only highly ranked results but also only click on documents that are

relevant, thus exhibiting evaluation bias while not exhibiting presentation bias. Evaluation bias is partly combatted by the concept of query chains, as they allow relevance judgments to be collected about documents at low rank for the original query but ranked highly by reformulations. However, we will also present a further method for combatting evaluation bias in Chapter 6.

Definition 1.6. *User diversity is the property that different users have different concepts of relevance given the same query.*

The presence of user diversity was, for instance, shown by Teevan et al. (2005a; 2007). It adds noise to relevance judgments collected from users. In particular, due to user diversity contradicting preferences may be collected for only the reason that the same query (such as *jaguar*) indicates that one user is looking for information on one topic (such as big cats) while another user is looking for different information (such as computer operating systems).

Finally, *Click noise* is present whenever logs of online user behavior are used.

Definition 1.7. *Click noise is noise in clickthrough data caused by users accidentally clicking on search results, or clicking without thinking.*

The effect of click noise is to obscure the signal in clickthrough data with random noise. Methods for collecting relevance judgments from clickthrough data need to be robust to the presence of this noise.

1.5 Presentation Bias and Learning Convergence

As seen earlier, presentation bias can be combatted by interpreting user clicks as relative relevance judgments rather than absolute judgments. However, this causes a further difficulty: the only relative statements that can be obtained using the methods proposed by Cohen et al. (1999) and Joachims (2002) are of the form that a lower ranked document is preferred to a higher ranked document. The same applies to preference judgments collected from query chains, as described in Chapter 4. This means that the relevance judgments collected in any dataset always oppose the order in which results are presented to users. In particular, if an original ranking were reversed, all the preferences based on that ranking would be satisfied. The effect (further described in Chapter 5) is that any ranking function trained using such implicit feedback will never converge to a fixed ranking.

The second key contribution of this thesis, presented in Chapter 5, is to describe an algorithm for randomly modifying the results shown to users to compensate for presentation bias. In particular, the algorithm presented is proved, under reasonable assumptions, to allow a learned ranking function to eventually converge to the optimal ranking function.

Previous convergence results of algorithms for learning to rank apply only if training data is assumed to come from distributions that do not suffer from such bias effects. For instance, Cohen et al. (1999) and Freund et al. (2003) assumed preferences are drawn according to a model that does not allow for preferences one way (i.e. opposing the original presentation order order) to be much more likely than preferences the other way. Most other theoretical convergence results

for learning to rank address learning from absolute relevance judgments (for example, Herbrich et al. (2000); Crammer and Singer (2001); Chu and Keerthi (2005)).

1.6 Combatting Evaluation Bias

Although we have now seen that user behavior provides implicit relevance judgments, even after considering query chains, evaluation bias still limits which results are assessed by users. In particular, previous work in learning to rank using clickthrough data has only considered clickthrough data that is collected anyway. Specifically, it assumes that when collecting clickthrough data, users are simply presented with the documents as ranked by the current ranking function. As such, these documents are shown without regard for what data may be collected or how this data may impact the rankings presented in the future. However, Granka et al. (2004) showed that users very rarely even look at results beyond the top few. Hence the data obtained by restricting the rankings shown to those generated by a pre-existing ranking function is strongly biased toward results already ranked highly. This means that highly relevant documents that are not initially ranked highly, due to the ranking function being suboptimal, may very rarely be observed and evaluated. This can lead to the learned ranking converging to an optimal ranking only very slowly.

To avoid evaluation bias, the third key contribution of this thesis is to propose a method to select the rankings presented to users. We show that this method obtains more useful training data, while also presenting high quality rankings. As an illustration, a naïve possibility for obtaining more useful training data and

guarantee eventual convergence to an optimal ranking would be to intentionally present unevaluated results in the top few positions of rankings, aiming to collect more feedback about them. However, such an ad-hoc approach is unlikely to be useful in the long run due to the sheer number of documents in a typical collection (for instance, the Web consists of at least tens of billions of documents), and will likely strongly hurt user satisfaction. In Chapter 6, we will introduce a principled approach to modify the rankings presented to users such that ranking quality improves quickly while limiting any temporary reduction in quality as new documents are explored. In particular, this approach consists of maintaining a probability distribution over document relevance. This contrasts it with other probabilistic approaches for ranking that estimate document relevance while not explicitly modeling uncertainty (for example, (Chu & Ghahramani, 2005c)).

Given uncertainty information, it is possible to compute the probability of any particular ranking being optimal, and also compute what the loss from presenting a different, suboptimal, ranking would be. By integrating over all possible rankings, we will show how to compute the expected loss of any given ranking. The algorithm will then select rankings to show such that this expected loss is minimized, given the implicit feedback we expect to collect from users. In particular, minimizing such a user-centric loss contrasts with previous approaches extending probabilistic models to actively select pairs of documents to evaluate. While Chu and Ghahramani (2005a) previously chose to ask for labels over pairs of documents where the entropy of the predicted outcome of a comparison would decrease most, the approach we present minimizes a user-centric loss measure. By minimizing the future expected loss as a function of the training data that is likely to be collected, we will show that the quality of

the ranking presented can improve much more rapidly than naively presenting documents in terms of decreasing estimated relevance.

1.7 Addressing User Diversity

Until now, this introduction has presented training data collection for Web search in terms of the relevance of individual documents, either as absolute or relative (pairwise) judgments. This implicitly assumes that each document has some real valued relevance to a particular query. As such, with a complete set of relevance scores, should it be possible to collect them for all documents, it would be sufficient to rank the documents by these scores. More formally, given judgments assessing the relevance of documents to a query, the standard approach is to learn the parameters of a scoring function. Given a new query, this function can be called upon to compute the score for each document *independently*, and rank documents by decreasing score.

The theoretical model that justifies ranking documents in this way is the probabilistic ranking principle (Robertson, 1977). It suggests that documents should be ranked independently by their probability of relevance to the query. However, the optimality of this process relies on the assumption that there are no statistical dependencies between the probabilities of relevance among documents – an assumption that is clearly violated in practice. For example, if one document about jaguar cars is not relevant to a user who issues the query *jaguar*, other car pages are also now less likely to be relevant. As users are often satisfied with finding a small number of, or even just one, relevant document, the usefulness and relevance of a document does depend on other documents ranked higher.

In fact most search engines today attempt to eliminate redundant results and produce *diverse* rankings that include documents that are potentially relevant to the query for different reasons.

Most previous work in obtaining diverse rankings suggests to diversify the top ranked documents given a non-diverse ranking. Perhaps the most common technique is Maximal Marginal Relevance (MMR), proposed by Carbonell and Goldstein (1998). Given a similarity (relevance) measure between a document and a query, as well as a similarity measure between two documents, MMR iteratively selects the most relevant documents that are also least similar to any other documents already selected. As such, MMR requires the relevance of a document to a query and the similarity of two documents to be known. It is usual to obtain these using standard algorithms for learning to rank, which will be discussed in Chapter 3. The goal of MMR is to *rerank* an already learned ranking to improve diversity.

In contrast, the fourth key contribution of this thesis is to present an algorithm that learns rankings that directly maximize a proxy for the fraction of users who find at least one relevant search result. This algorithm produces a diverse ranking of results in a principled and provably optimal manner. Specifically, the Ranked Bandits Algorithm that we present in Chapter 7 obtains the best achievable polynomial time approximation to maximizing the fraction of users who click on at least one search result. In addition, the algorithm does not assume a relevance or document similarity measure is known or provided a priori, instead learning directly what ranking of documents is best to present.

It is also worth noting that implicit feedback is particularly useful when learning to produce diverse rankings in a principled manner. Learning to produce

optimally diverse rankings using expert judgments would require a document collection with document relevance obtained for all possible meanings of a query. While the TREC interactive track² provides some documents labeled in this way for a small number of queries, such document collections are even more difficult to create than standard expert labeled collections. Moreover, the judgments would need to establish the relative importance of the different meanings of each query to optimally satisfy the user population.

1.8 Bibliographic Notes

The research presented in this thesis was performed and subsequently published jointly with Professor Thorsten Joachims and partly jointly with Professor Robert Kleinberg at Cornell University. While Chapters 1, 2 and 3 present background material, the key contributions of this thesis have been published as follows. Chapter 4, which presents and evaluates the concept of query chains was published in (Radlinski & Joachims, 2005b). Chapter 5, which presents an algorithm to combat presentation bias, was published in (Radlinski & Joachims, 2006). Chapter 6, which describes an approach to avoid evaluation bias, was published in (Radlinski & Joachims, 2007). Finally, Chapter 7, which addresses user diversity, was published in (Radlinski, Kleinberg and Joachims, 2008b).

During my degree, I have also contributed other research not presented as a chapter in this thesis. In particular, I have published research on personalized Web search (Radlinski & Dumais, 2006), identifying related documents using implicit feedback (Pohl, Radlinski and Joachims, 2007), machine learning algorithms

²http://trec.nist.gov/data/t11_interactive/t11i.html

for optimizing ranking performance metrics (Yue, Finley, Radlinski and Joachims, 2007), ranking online advertisements (Radlinski, Broder, Ciccolo, Gabrilovich, Josifovski and Riedel, 2008a), studying the tolerance of learning from implicit feedback to random and malicious noise (Radlinski & Joachims, 2005a; Radlinski, 2007) and studying how high school students prepare for high stakes exams (Loken, Radlinski, Crespi, Cushing and Millet, 2004; Loken, Radlinski, Crespi and Millet, 2005). I also contributed to (Joachims, Granka, Pan, Hembrooke, Radlinski and Gay, 2007). Finally, a less technical overview describing a number of the key ideas in this thesis was published in (Joachims & Radlinski, 2007).

CHAPTER 2

UNDERSTANDING AND INTERPRETING USERS' DECISIONS

This chapter presents an overview of previous research that considers users' decision making processes, and how user actions can be interpreted. In particular, we will see how users' decisions are affected by the way users are presented with choices to make. We start by considering the decisions people make in the offline world, from a marketing and economics perspective, as well as when applied to document relevance judgments. Following this, the majority of this chapter will consider user behavior in a Web search setting, asking a number of questions that will determine how implicit feedback can be interpreted.

2.1 Offline Decision Making

The process by which people make decisions offline has been studied extensively, particularly motivated by marketing applications. For instance, Eliashberg (1980) presented approaches for estimating consumer utility functions and Currim and Sarin (1984) studied job preferences using a related utility formulation. Numerous related behavioral models have been proposed, including bounded rationality (Herbert, 1957), aspiration adaptation theory (Selten, 1998) and prospect theory (Tversky & Kahneman, 1981).

2.1.1 What influences user decisions?

Of more direct relevance to this thesis, a number of recent studies have explored factors that impact the reliability of preference choices made by people. Mantell and Kardes (1999) describe how the order in which choices are presented affects the preferences shown. In particular, they found that when comparisons are made between products based on specific attributes, order effects play a larger role than when comparisons are made in terms of overall (attitude-based) evaluations. Moreover, when comparing pairs of products, Moore (1999) showed that if a superficially attractive option is presented first, consumers are willing to pay more for both options than when the superficially less attractive option is presented first. In addition, Coupey et al. (1998) and Zhang and Markman (2001) studied the effect of familiarity and motivation on preference decisions respectively. They found that the preferences people make are influenced by these factors, suggesting that consumer decision making is indeed a complex process.

In our context of interactive information ranking systems, these studies should be taken as indicative of the complexity of the behavior we will observe from online users. It is unlikely that simple behavioral models will suffice to explain why people express the preferences they express. Rather the studies motivate careful analysis of online user behavior to identify complicating factors, to allow us to find ways to avoid them when collecting implicit feedback. For instance, these studies suggest that the order users are presented with online search results will affect their judgments of the relevance of the results.

2.1.2 Are decisions absolute or relative?

In the context of document relevance judgments, a particularly important question to ask in the offline world regards which sort of judgments people are able to make most reliably. In particular, when human experts are trained to make judgments about the relevance of documents to queries, what sort of judgments can we expect? Recently, Carterette et al. (2008) studied how relevance judgment quality is affected by the specific question asked of expert human judges. The standard approach to obtain relevance judgments from expert judges is for judges to provide a relevance score for each (query, document) pair on a two to six point scale. These relevance scores can then be used to construct a set of relative statements. Carterette et al. compared the relative judgments obtained in this way to those obtained by asking the experts for relative judgments directly. They found that relevance judges tend to agree more with each other when asked to directly provide relative judgments about pairs of documents. Moreover, it took the judges substantially less time to make a relative judgment than it took them to make an absolute judgment on a five point scale. This shows us that people (at least those trained to make relevance judgments) appear to have an easier time providing reliable and consistent relative judgments than absolute judgments.

2.2 Online Decision Making

We can now turn to the question of what decisions regular Web users make when searching for information online, and how we can identify the extent to which document relevance affects the decisions.

2.2.1 What can we observe of online user behavior?

Given the online medium, the first question we must ask is how much of user behavior can we practically observe? In principle, it is possible to record all user interactions with a Web browser. In particular, Kelly and Teevan (2003) provided a detailed survey of the many user actions that can be observed including browsing, bookmarking, printing, and so forth. Somewhat more recently, White et al. (2005), Fox et al. (2005) and Kellar and Watters (2006) also studied what can be observed of user behavior if users are asked to use specially instrumented Web browsers. Each study addressed a different prediction problem in terms of user satisfaction or user task. However, key to this thesis, to obtain such a complete picture of what users are doing online requires the users to use Web browsers with specific add-ons that record all these interactions. Understandably, a relatively small fraction of real users install such tools, both due to the inconvenience of additional software installation and due to privacy concerns. Hence, the number of users for whom complete data could be collected is limited.

A simpler alternative is to record the entire stream of network traffic issued by users. For instance Kammenhuber et al. (2006) installed a monitor at the Internet gateway of a major German university, using the data collected to build a model of user behavior. However, this approach requires the cooperation of Internet service providers, with similar privacy concerns and logistical difficulties as when users are asked to install browser add-ons.

However, as we are concerned particularly with interactive information ranking systems, a sufficiently complete picture of user behavior can be obtained by recording the user interactions with just the system of interest. For instance, in

a Web search setting, user queries can be recorded by the search engine directly. If the search results link to addresses on a search engine server, that then redirect users to the actual result pages, we can also observe which search results users click on when they click.

Although we will study in detail what information is provided by logs of queries and clicks, as an example that clicks encode substantial information, consider a recent result obtained by Pohl et al. (2007). In their work, Pohl et al. found that documents in an academic search engine that are co-clicked on by search engine users tend to be co-cited in the future. This suggests that potentially useful information is present in clickthrough logs.

2.2.2 Which search results do users consider?

Before we can interpret user behavior in Web search, we must first establish which results users actually look at and hence can possibly be making judgments about. We do so by drawing on the results of groundbreaking eye tracking studies reported by Granka, Joachims and collaborators (2004; 2005; 2007). In particular, the authors study user behavior on the Google search engine³, considering both what users look at, and what users click on. While other researchers had previously used eye tracking studies to observe how users behave when using a Web browser (for instance, Brumby and Howes (2004)), they did not specifically focus on behavior on the results page of a Web search engine.

In each phase of a two-phase study, Granka et al. recruited undergraduate student volunteers to search for the answers to specific questions, while having

³<http://www.google.com/>

an eye tracker record where on the screen they looked during the process. The subjects were asked to start from the Google search page and find the answers to ten questions. Five of the questions asked were navigational (for example, “*find the homepage of Emeril, the chef who has a television cooking program*”) while the other five were informational (for example, “*what is the name of the researcher who discovered the first modern antibiotic?*”) (Broder, 2002). The questions asked varied in difficulty and topic content. There were no restrictions on what queries the users may choose, how they may continue after entering the first query, or which links to follow. Users were told that the goal of the study was to observe how people search the Web, but were not told of the specific interest in their behavior on the results page of Google. All clicks, the results returned by Google, and the pages connected to the results were recorded by an HTTP proxy. A detailed presentation of the experimental setup is provided by Joachims et al. (2007).

The light bars in Figure 2.1 (Figure 1 from Joachims et al. (2007)) show the percentage of result pages for which users looked at each of the top 10 search result abstracts (the short text that describes each search result) for a query. The dark bars show the fraction of the time that a user’s first click was at a particular rank. The striking result is that while most users looked at least at the top two result abstracts, the fraction of searches after which users even *look* at lower ranked results decays very rapidly with result rank.

These eye tracking results are consistent with other studies that have also seen that users tend to pay much more attention to top ranked documents in a Web search engine. For instance, Agichtein et al. (2006) presented a summary distribution of the relative click frequency on Web search results for a large commercial search engine as a function of rank for 120,000 searches for 3,500

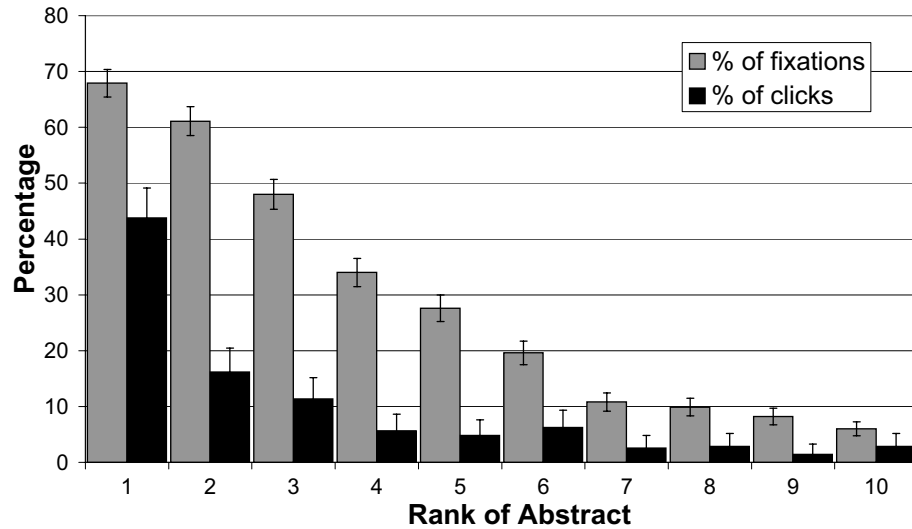


Figure 2.1: Percentage of time an abstract was viewed/clicked on depending on the rank of the result (from Joachims et al. (2007)).

distinct queries. They show that the relative number of clicks rapidly drops with the rank – compared with the top ranked result, Agichtein et al. observed approximately 60% as many clicks on the second result, 50% as many clicks on the third, and 30% as many clicks on the fourth.

2.2.3 Which results do users consider before clicking?

Given that the interactions of users with the Web search engine that we record are clicks on search results, we next ask which results have been observed before users click.

Granka et al. found that users in their study considered search results in order from top to bottom. Figure 2.2 (Figure 3 from Joachims et al. (2007)) shows the number of abstracts viewed above and below any result that was clicked on. We see that the lower the rank of the clicked document, the more previous abstracts

the user is likely to have looked at first. While users do not tend to look at all earlier abstracts, this figure suggests that users do generally scan the results in order from top to bottom. We also see that users usually look at one abstract below any they click on. Further analysis by Joachims et al. (2007) showed that this is usually the abstract immediately below the one clicked on. We can conclude that users typically look at most of the results from the first to the one below the last one clicked on.

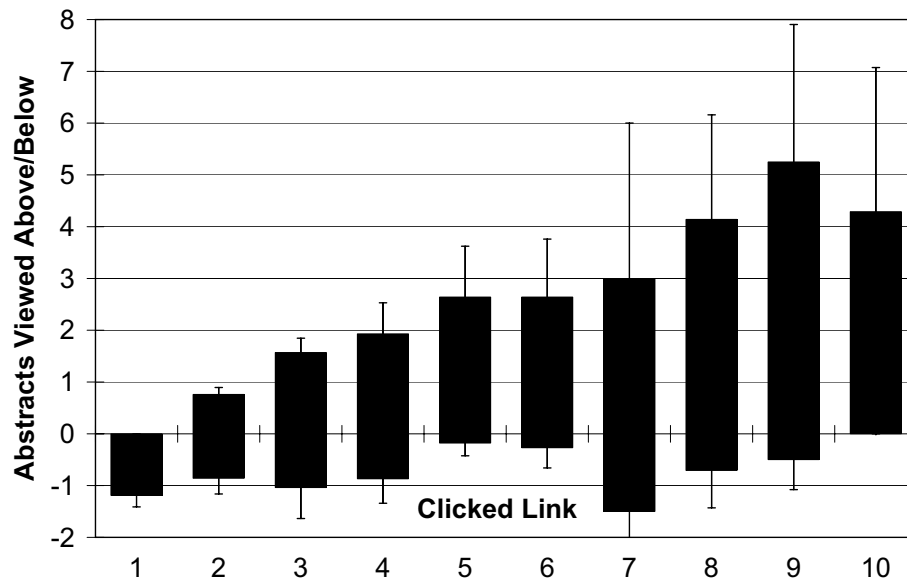


Figure 2.2: Mean number of abstracts viewed above and below a clicked link depending on its rank (from Joachims et al. (2007)).

2.2.4 How can we infer judgments from clicks?

Now that we can infer which results users looked at before clicking, we can ask the question: What affects a user's decision to click? To study these influences, Joachims et al. evaluated how users' behavior is affected by the quality of the results shown. In the second phase of the eye tracking study, some users were presented with the original Google results, while some were presented with the

top two Google results swapped, and others were presented with the top ten results in reverse order. None of the users in the study suspected such a change was being made. Figure 2.3 (Figure 4 from Joachims et al. (2007)) shows how these changes affected which results users looked at, and which ones they clicked on. It shows that that reducing the relevance by modifying the order of results impacted how users behaved, both increasing the fraction of users who looked at lower ranked results, and increasing the number of clicks at low rank.

This may lead us hypothesize that users simply click on relevant results, as is assumed by much previous work (for example, by Boyan et al. (1996); Kemp and Ramamohanarao (2002); Fox et al. (2005)). We will now see that users' decisions to click are more complex. Following the eye tracking study, Joachims et al. asked human evaluators to assess the relative relevance of all the results seen by each study participant. The evaluators were asked to rank all the short text abstracts presented to participants on the Google search results page in terms of relevance to the questions asked of the participants (evaluators were allowed to judge two abstracts as equally relevant). For the documents returned in the second part of the study, the evaluators were also asked to rank the documents linked to by the search results.

If users simply click on relevant documents, unaffected by the presentation order, we would expect that the frequency with which users click on the top two results only depends on the relevance of these results. However, Joachims et al. found a significant change in clicking behavior when the top results were presented in reverse order. They show that two effects are in play: first, a *trust bias* as users appear to trust that documents presented high by the search engine are most relevant, and second a *quality-of-context bias* where users are more likely

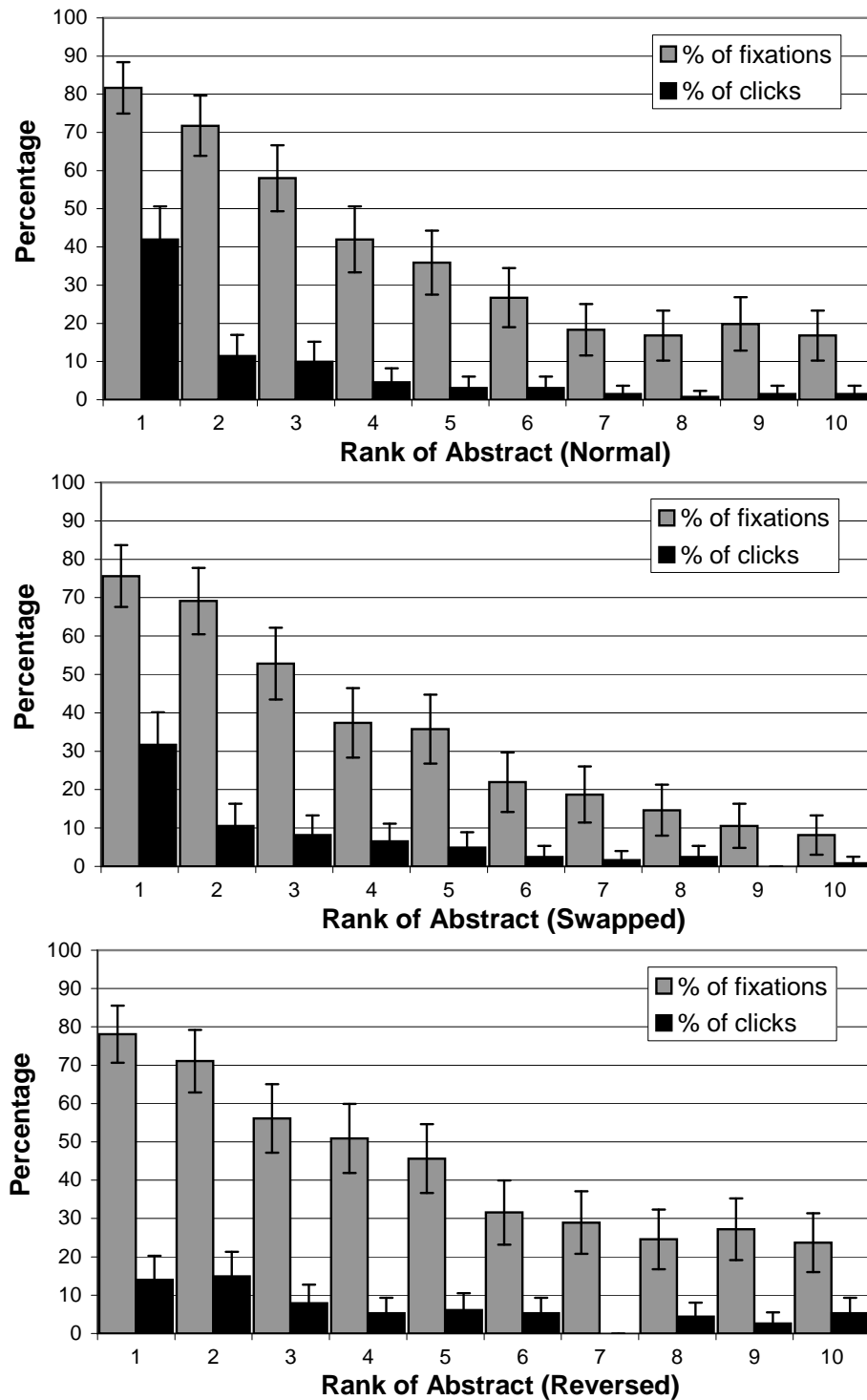


Figure 2.3: Fraction of results looked at, and clicked on, by users presented with standard Google results (top), top two results reversed (middle) and all top ten results reversed (bottom) (from Joachims et al. (2007)).

to click on less relevant documents if they are embedded in a poorer quality ranking.

Due to these effects, interpreting clicks as judgments about the relevance of the clicked documents appears difficult. However, Cohen et al. (1999) and Joachims (2002) previously suggested an alternative interpretation of user clicks: as *relative* relevance judgments. Joachims et al. found evidence supporting such an interpretation of clickthrough data in the eye tracking study. We now discuss this approach.

2.2.5 Implicit Feedback as Relative Relevance Judgments

Intuitively, selection of Web search results may be much like the selections people make when purchasing products, where they must select among the available options. Perhaps users choose to click on a search result not because it is relevant, but because it is *more likely* to be relevant than any other option they are aware of. This idea is known as revealed preferences in economics (Samuelson, 1948; Varian, 1992). We now consider how clicking behavior encodes such preferences.

Cohen et al. (1999) and Joachims (2002) proposed an interpretation to clickthrough data in terms of relative relevance judgments. They hypothesized that when a user clicks on a document, the user is indicating that he or she considers the selected document more relevant than other documents he or she has already considered but not clicked on. Including this strategy, Joachims et al. (2005) discuss five strategies for obtaining relative relevance feedback from users clicking behavior along similar lines. These strategies are summarized in Table 2.1

Table 2.1: Summary of the feedback strategies discussed by Joachims et al. (2005).

Strategy	Description
CLICK > SKIP ABOVE	A clicked-on document is likely more relevant to the query than one presented higher but not clicked on.
LAST CLICK > SKIP ABOVE	The last clicked-on document is likely more relevant than any documents presented higher but not clicked on.
CLICK > EARLIER CLICK	A clicked-on document is likely more relevant than any documents presented higher that were clicked on.
CLICK > SKIP PREVIOUS	A clicked-on document is likely more relevant than the preceding document, if that document was not clicked on.
CLICK > NO-CLICK NEXT	A clicked-on document is likely more relevant than the next document, if that document was not clicked on.

To illustrate these strategies and the relative relevance judgments they would generate, consider for example a user who submitted query q and was presented with the ranked results (d_1, d_2, d_3, d_4) , then clicked on d_2 and d_4 . Note that in this case, since the user clicked on the fourth result, the results from the eye tracking study suggest that the user probably also considered results d_1 and d_3 despite deciding not to click on them. CLICK > SKIP ABOVE would generate three preferences:

$$d_2 \succ_q d_1; d_4 \succ_q d_1; d_4 \succ_q d_3$$

where $d_i \succ_q d_j$ should be taken to mean not that d_i is relevant to q , but rather that d_i is *more likely* to be relevant than d_j , with respect to the query q . LAST CLICK > SKIP ABOVE would generate just the preferences:

$$d_4 \succ_q d_1; d_4 \succ_q d_3,$$

making the assumption that perhaps d_2 is not actually more relevant to the user, as he or she chose to come back to the search results after observing that document. CLICK > EARLIER CLICK would generate the preference

$$d_4 \succ_q d_2,$$

again taking the view that perhaps the last clicked document is most likely to satisfy the user's needs. CLICK > SKIP PREVIOUS would generate the preferences

$$d_2 \succ_q d_1; \quad d_4 \succ_q d_3,$$

being more conservative in generating long-distance preferences since the result most reliably evaluated before a click is the one directly preceding it. Finally, CLICK > NO-CLICK NEXT would generate the preference

$$d_2 \succ_q d_3,$$

since users usually consider the *next* result before clicking.

2.2.6 Are these judgments valid?

While the feedback strategies described above are intuitively appealing, a quantitative evaluation is necessary to establish their degree of validity. We now present a summary of an evaluation of the above strategies reported by Joachims et al. (2005).

Joachims et al. evaluated the accuracy of each of these feedback strategies by asking human judges to rate the relevance of every result returned to the volunteers during the eye tracking study. The judges could know the ground truth

Table 2.2: Accuracy of the implicit relevance judgments obtained when using each of the strategies presented by Joachims et al. (2005) in Table 2.1. The Phase II numbers include the average performance of the strategies across data collected in all three experimental conditions described earlier: normal, swapped and reversed.

Strategy	Abstracts		Pages
	Phase I	Phase II	Phase II
<i>Inter-Judge Agreement</i>	89.5	82.5	86.4
CLICK > SKIP ABOVE	80.8 ± 3.6	83.1 ± 4.4	78.2 ± 5.6
LAST CLICK > SKIP ABOVE	83.1 ± 3.8	83.8 ± 4.6	80.9 ± 5.1
CLICK > EARLIER CLICK	67.2 ± 12.3	46.9 ± 13.9	64.3 ± 15.4
CLICK > SKIP PREVIOUS	82.3 ± 7.3	81.6 ± 9.5	80.7 ± 9.6
CLICK > NO CLICK NEXT	84.1 ± 4.9	70.4 ± 8.0	67.4 ± 8.2

relevance scores as they were given the questions provided to the study participants. The judges were asked to produce a partial ordering of all abstracts shown, as well as the Web pages the results point to, for each query issued. By presenting some of the sets of documents to two different judges, Joachims et al. measured inter-judge agreement, seeing agreement rates between 82.5 and 89.5% in the human relevance judgments. This means that when two human judges judged that one document in a pair was more relevant, they agreed on which document was more relevant this fraction of the time. The remaining 10.5 to 17.5% of the time, the judges disagreed as to which document was more relevant. This measure is important as it provides an upper bound the accuracy that could be expected from any relevance judgments obtained implicitly or explicitly from users.

Using these judgments, the authors assessed the preferences generated by the strategies in Table 2.1, obtaining the results shown in Table 2.2.

The results show that all the strategies that generate preferences of the form that a clicked document is likely more relevant than a higher ranked non-clicked

document are very reliable. In particular, the accuracy of these strategies is very close to the inter-judge agreement, which is an upper bound on the accuracy that could ever be expected. Additionally, it is apparent that despite users clicking after only having seen a short abstract extracted from each ranked document, the clicks reflect the true relevance of the documents. All together, CLICK > SKIP ABOVE provides the largest number of relevance judgments, and these judgments are equally reliable to those obtained from the more conservative strategies.

2.3 Summary

In this chapter, we have considered previous work that studied how people, and in particular Web search users, actually make decisions. Following this, we presented a number of strategies for how online search behavior can be used as implicit relevance feedback. In particular, Web search users typically read results from top to bottom. Observing user clicks and making inferences that a clicked on document is more relevant than higher ranked skipped documents has been shown to be valid.

CHAPTER 3

LEARNING TO RANK

Thus far, we have seen how previous research obtained relevance judgments for training personalized information ranking, and particularly Web search, systems. Now, we turn to the question of using these relevance judgments to learn a ranking function. We start by presenting an overview of standard performance metrics used for evaluating the performance of ranking algorithms. Following this, we will summarize some of the major algorithms for learning to rank, grouping them based on the type of training data that they require. The last part of this chapter will provide a more detailed description of two key algorithms that will be used later in this thesis.

3.1 Performance Metrics

Suppose we provide some query q to a Web search engine, and obtain a ranked list of results \mathcal{D}_q . How do we evaluate the quality of \mathcal{D}_q ? Many different metrics have been proposed for evaluating the quality of rankings. We now provide an overview of those most commonly used. An in-depth discussion of these metrics is presented by Manning et al. (2008).

3.1.1 Precision and Recall

We start with two of the simplest ranking performance measures, precision and recall.

Suppose that we consider each document d_i in some document collection \mathcal{C} as either relevant, or not relevant, to a query q (in other words, relevance is binary). Let $rel_q(d_i) \in \{0, 1\}$ denote this relevance. Further, let $\mathcal{D}_q = (d_1, \dots, d_n)$ denote the ranking returned for this query by the search engine being evaluated.

Two basic questions that can be asked about \mathcal{D}_q are: What fraction of the documents in \mathcal{D}_q are relevant? What fraction of all the available relevant documents in \mathcal{C} were found in \mathcal{D}_q ? These two measures are called *precision* and *recall*. We can write them as:

$$Precision(\mathcal{D}_q) = \frac{|\{d \in \mathcal{D}_q \mid rel_q(d) = 1\}|}{|\mathcal{D}_q|} \quad (3.1)$$

$$Recall(\mathcal{D}_q) = \frac{|\{d \in \mathcal{D}_q \mid rel_q(d) = 1\}|}{|\{d \in \mathcal{C} \mid rel_q(d) = 1\}|} \quad (3.2)$$

Precision and recall measure two fundamentally different things. Precision measures whether documents returned are indeed relevant, while recall measures how many relevant documents were missed. Both may be appealing in different cases. For instance, if we need just one document to answer the query (for instance, to find the definition of some term), we would prefer high precision so that we don't need to read through many irrelevant documents. On the other hand, if we want to make sure we find all relevant documents (for instance, preparing for a legal trial), having high recall may be more important. To compare systems that trade off precision and recall differently, a combination score that is commonly used is the *F-1 score*:

$$F1(\mathcal{D}_q) = \frac{2 \times Precision(\mathcal{D}_q) \times Recall(\mathcal{D}_q)}{Precision(\mathcal{D}_q) + Recall(\mathcal{D}_q)} \quad (3.3)$$

3.1.2 Ranking Based Metrics

While precision and recall are intuitive, they ignore the order in which documents are returned. Ignoring this order ignores important information about rankings, because as we have seen in the previous chapter, users of search engines tend to read through returned items in a consistent order. Indeed, if a search engine returns 100,000 results for some query, it is of no use to a user if all the relevant documents are at the end of that list.

One approach, initially proposed by Cooper (1968), takes the view that a good ranking should minimize the amount of “wasted effort” users must exert before finding a sufficient number of relevant documents. As a special case, we can consider the number of documents a user must look at before finding the first relevant document. This performance measure is called the *mean reciprocal rank* (*MRR*) and is usually taken as the inverse of the rank of the first relevant document in \mathcal{D}_q .

Another common way to evaluate how well results are ordered is using a variant of precision, *precision at k* ($P@k$). This metric measures the precision if we only consider the first k results in \mathcal{D}_q . It is otherwise identical to regular precision. In effect, this caps the effort a user would put in (that is, we assume that the user would only consider at most k results) and $P@k$ tells us what fraction of those documents would be relevant. Common values for k are between 1 and 10.

But how do we choose the right value of k for evaluating systems? In particular, different queries may have different numbers of relevant documents. If k is larger than the number of relevant documents for some query, the highest possible precision at k may be much smaller than 1. One option is to find the k

where the precision and recall are equal, and report this precision. It is commonly called the *precision-recall break-even point*.

However, recall is usually less interesting in a Web search setting than precision. *Average precision* is a measure that solves the difficulty with $P@k$ while also placing substantially more weight on the top ranked documents. It measures the average of the precision at the rank of every relevant document (sometimes truncating D_q first, and considering all documents below some cutoff rank as “not found”):

$$AP(\mathcal{D}_q) = \frac{1}{|\{d \in \mathcal{C} \mid rel_q(d) = 1\}|} \sum_{i=1}^{|\mathcal{D}_q|} rel_q(d_i) \times Precision@i(\mathcal{D}_q). \quad (3.4)$$

One important property of averaging over all the relevant documents is that average precision is much more sensitive to the relevance of the first few documents than those further down. To evaluate the performance of a ranking system, typically a large number of queries is selected and the mean of the average precision scores across all queries is measured. This measure is called the *mean average precision* (MAP) and is one of the most common metrics for measuring the performance of ranking functions.

Various extensions or replacements for MAP have also been proposed, in particular addressing the limitation that documents relevance is binary. For instance, Järvelin and Kekäläinen (2000) introduced discounted cumulative gain (DCG). It is often used when documents have non-binary relevance scores. The DCG of a ranking is defined as

$$DCG(\mathcal{D}_q) = \sum_{i=1}^k \frac{2^{rel(d_i)} - 1}{\log(i + 1)}, \quad (3.5)$$

considering just the first k documents in \mathcal{D}_q . While DCG does not average over just the most relevant documents, the log factor in the denominator means that the relevance of highly ranked documents is given much more weight than that of lower ranked documents. A common variant of DCG also introduced by Järvelin and Kekäläinen (2000) is normalized DCG (NDCG). It simply normalizes the DCG to always output a value between 0 and 1. A related performance metric used with binary relevance judgments when not all documents have been judged is $bpref$, and was introduced by Buckley and Voorhees (2004). An extension to graded relevance judgments called $rpref$ was described by De Beer and Moens (2006).

3.1.3 Performance with Relative Judgments

Unfortunately, to evaluate the precision, recall, or related measures of ranking quality presented above, we need absolute relevance judgments of the documents. As seen in Chapter 2, these can be much more difficult to obtain than relative relevance judgments. What sort of performance measures can be used only given relative relevance judgments?

One straight-forward performance measure that only needs relative relevance judgments is the fraction of misordered pairs of documents in a ranking:

$$MisorderedPairs(\mathcal{D}_q) = \frac{2}{|\mathcal{D}_q|(|\mathcal{D}_q| - 1)} \sum_{i=2}^{|\mathcal{D}_q|} \left| \{d \in (d_1, \dots, d_{i-1}) \mid d \prec_q d_i\} \right| \quad (3.6)$$

Strongly related ranking performance measures include Kendall's Tau (Kendall & Gibbons, 1990; Fagin et al, 2003), Guttman's Point Alienation (Bartell

& Cottrell, 1995), the Spearman Footrule (Diaconis & Graham, 1977) and ROC Area (Provost & Fawcett, 1997; Joachims, 2005).

Yet these measures also have a difficulty: They assume that for all pairs of documents, we know if the documents are misordered. If pairwise relevance judgments are collected from users, it is likely that this information will not be available for many pairs of documents. Although *bpref* (Buckley & Voorhees, 2004) can be used if we assume many relevant documents have been assessed in pairwise comparisons, there are no standard performance metrics that can compute an absolute performance score in the general case.

3.1.4 Comparing Two Rankings

In addition to the difficulty in obtaining labeled data, results presented by Hersh et al. (2000), Turpin and Hersh (2001), Allan et al. (2005) and Turpin and Scholer (2006), among others, have shown that many of the measures described above do not always correlate strongly with user satisfaction. For example, Turpin and Scholer constructed rankings with widely differing mean average precisions and measured the time it took users to find information in controlled tests. They saw that the correlation between MAP and the speed with which users find information is weak. This suggests the following question: How can the performance of a ranking function be evaluated in a way that is practical and reflects user needs?

One option is to build a model of user behavior then estimate user satisfaction based on how users perform relative to the model, as proposed for instance by Dupret et al. (2007). However, the model they proposed requires a number of

assumptions about user behavior that are difficult to test, and also requires the estimation of many parameters.

In contrast, Joachims (2003) proposed a simple method to determine which of *two* ranking functions is *better*. Unlike the metrics described above, this does not provide an absolute performance score of either ranking, but rather provides a relative preference for one of the input ranking functions. Specifically, given two ranking functions, he suggested an interleaving of the results generated be presented to users. This interleaving is constructed such that irrespective of how many results a user may consider, he or she will have observed an equal number of documents from each ranking in expectation.

Figure 3.1 shows two example rankings, r and r' , from two different retrieval functions as well as a valid combination, $combined(r, r')$. Let $seen(n, r)$ and $seen(n, r')$ be the number of results the user has seen from rankings r and r' respectively after looking at the top n results from the combined ranking. $seen(n, r)$ and $seen(n, r')$ are defined as the smallest number of results that we have to combine from r and r' to produce the top n results of the combined ranking. The combined ranking is generated such that for any n , $seen(n, r) \geq seen(n, r') \geq seen(n, r) - 1$. In the above example, if the user looks at the top three results in the combined ranking, this is satisfied because $seen(3, r) = 2$ and $seen(3, r') = 2$. If the user looks at the top five results, $seen(5, r) = 4$ and $seen(5, r') = 3$. To compensate for a bias toward the results of r ($seen(n, r)$ is sometimes one bigger than $seen(n, r')$), r and r' are randomly switched half the time. This means that in expectation $seen(n, r) = seen(n, r')$.

Once this combined ranking is presented to users, we can evaluate which of the two rankings is preferred. We first determine which results the user

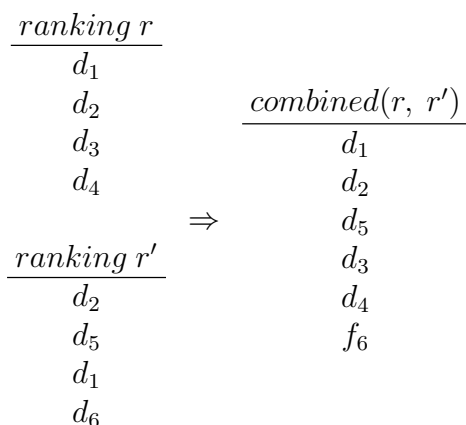


Figure 3.1: Two example rankings with four results each, and the combined outputs we would generate by starting with the top ranked document from ranking r .

looked at by taking the lowest ranked clicked-on document as where the user stopped scanning the results (a conservative estimate). If the two rankings are equally good, we would expect the user to click on just as many results from each ranking, given that he or she has seen an equal number from each (in expectation). Let $clicks(r)$ be the number of documents clicked on that are in the top $seen(n, r)$ results of r , and similarly for $clicks(r')$. For example, say the user clicked on d_1 and d_5 in the combined ranking shown in Figure 3.1. This method would infer the user looked at the top 3 results. From before, we have $seen(3, r) = seen(3, r') = 2$. Therefore, $clicks(r) = 1 (d_1)$ and $clicks(r') = 1 (d_5)$. This credit assignment algorithm is presented more formally by Joachims (2003).

If in expectation $clicks(r) > clicks(r')$, this approach concludes that the user prefers the ranking r over r' . By summing over many rankings shown to many users, this interleaving method counts how often $clicks(r) > clicks(r')$, and how often $clicks(r) < clicks(r')$, to obtain a relative comparison. A binomial sign test can then be used to assess the significance of any preference found.

3.2 Algorithms for Learning to Rank

We have now seen how the performance of a learned ranking function can be assessed in different situations. Yet how are these ranking functions obtained? We now provide a brief summary of some of the more influential algorithms for learning to rank. We categorize them by the specific form of training data that they assume as input.

3.2.1 Ranking as a Classification Problem

The simplest formulation of learning to rank is as a binary classification problem. For training examples x_1, \dots, x_n , the algorithm is provided with binary scores y_1, \dots, y_n where $y_i \in \{0, 1\}$. Given this data, the goal is to learn the parameters of a function that scores examples with a real number. New examples can then be ranked by decreasing score. In the case of Web search, x_i is a (document, query) pair, y_i is the relevance of the document to the query. The predicted score can be interpreted as the estimated level of relevance of documents to the query.

Many approaches generally fitting into this framework have been proposed. For instance, Fuhr (1989) proposed to learn the ranking function parameters by fixing the functional form and explicitly minimizing the squared error in the relevance predictions obtained. Boyan et al. (1996) used simulated annealing to learn the function parameters: Starting with an initial set of parameter values, the weights are repeatedly randomly perturbed, with parameter values resulting in improved performance kept. Learning to maximize the mean average precision of the rankings produced, Metzler and Croft (2005) use hill climbing to select

optimal parameter values while Joachims (2005) and Yue et al. (2007) perform global optimization of an upper bound on the error rate to optimize ROC Area and MAP respectively. While these last formulations treat learning to rank as predicting an entire ranking rather than the relevance of individual documents, they derive the correct rankings from relevance judgments on individual documents.

A second way to implement classification based ranking takes advantage of the relevance assignments being binary. Specifically, a ranking function can essentially memorize which documents are relevant to which queries. For example, Kemp and Ramamohanarao (2002) maintain modified Web documents by adding the text of relevant queries to the documents. Similarly, Scholer and Williams (2002) maintain a list of related queries for each document, while Xue et al. (2004) learn from clicking behavior which documents are related to each other and hence relevant to similar queries.

Finally, a third classification approach to a problem related to general ranking was recently proposed by Agichtein and Zheng (2006). They address the task of identifying when a query has a clearly best *single* document. For such queries, Agichtein and Zheng propose to learn to identify these “best bet” documents from training data. They postulate that this task is simpler than general ranking, but can have a substantial impact on ranking quality. Given a new query, if their classifier finds a “best bet” document, that document is ranked first. The lower ranked documents are obtained using a different ranking function.

3.2.2 Ranking as Ordinal Regression

Ordinal regression extends the binary classification setting by allowing the labels of examples to be integers in some range $y_i \in \{1, 2, \dots, k\}$. In a Web search setting, this corresponds to being provided with (query, document) pairs where each pair is labeled with one of a small number of scores to indicate the relevance of the document to the query.

Herbrich et al. (2000) proposed to address this setting using a modified classification support vector machine (see Cristianini and Shawe-Taylor (2000)). The modified support vector machine learned to score (document, query) pairs, but also learned which range of scores corresponds to which relevance levels. The technique was then further improved by Shashua and Levin (2002), Rajaram et al. (2003) and Chu and Keerthi (2005).

An alternative class of algorithms for the ordinal regression task is inspired by the perceptron (see Mitchell (1997)). A perceptron maintains the parameters of a scoring function, and iteratively considers training examples. Whenever an example that is misclassified is seen, the model is updated. Crammer and Singer (2001) proposed the *PRank* algorithm, which learns a model that consists of a linear scoring function and a number of thresholds. These thresholds allow each ranking score to correspond to a specific rank. As with a perceptron, the *PRank* algorithm iteratively considers the training examples, updating the model and thresholds when misclassified examples are seen. The *PRank* algorithm has also been improved and generalized by Harrington (2003) and Basilico and Hofmann (2004).

3.2.3 Ranking as Regression

For some ranking tasks, real-valued labels for the items being ranked are available. In this case, $y_i \in \mathbb{R}$. For instance, Caruana et al. (1995) considered the problem of ranking people by their risk of developing pneumonia when receiving medical treatment. By considering groups of people as training data, each group has a real-valued empirically observed rate of developing pneumonia. Caruana et al. are then able to apply a neural network to the regression problem, allowing future patients to be scored and thus ranked by their risk.

More recently, Sun and Giles (2007) suggest to order Web document by the frequency with which the documents were previously clicked on when displayed by a search engine. This also provides real-valued training labels that can be used to learn to rank the documents.

Finally, by using a probabilistic model to smooth ordinal human relevance judgments, Taylor et al. (2008) show how to use regression techniques to maximize the NDCG of a ranking function by gradient descent.

3.2.4 Learning to Rank with Pairwise Data

Given the difficulty in obtaining reliable absolute relevance labels, many researchers have also studied algorithms that learn from training data of the form of pairs of examples (x_1, x_2) indicating that x_1 should be ranked above x_2 .

In one of the earlier pairwise algorithms, Wong et al. (1988) proposes to use a perceptron-like algorithm to learn rankings that satisfy a given set of pairwise

constraints. In a more general setting, Cohen et al. (1999) proposed an improved iterative algorithm that learns a combination of ranking functions such that the rankings generated minimize any performance metric provided over rankings. Freund et al. (2003) proposes to learn a sequence of ranking functions that, when averaged, minimize the number of violated preferences.

One common general machine learning approach used for classification and regression tasks is gradient descent over an error metric. However, given pairwise training data, natural ranking metrics such as the number of misordered pairs or MAP are not smooth. Many gradient descent algorithms have been proposed that learn by finding a smoother surrogate measure, or using iterative parameter updates to obtain a similar effect. For instance, assuming training data of the form of triplets $(x_1, x_2, P(x_1 \succ x_2))$, where the third term indicates the probability that x_1 is actually preferred over x_2 , Burges et al. (2005) proposed the RankNet algorithm. This algorithm performs gradient descent over a cost function that encodes the error in estimating this probability given for the particular pair of items being ranked. Other gradient descent or weight update algorithms were developed by Bartell et al. (1994); Bartell and Cottrell (1995); Dekel et al. (2003); Rudin et al. (2005); Burges et al. (2006).

An alternative approach that smoothes the performance metric is to define a convex upper bound to the error present in a ranking. Joachims (2002) proposed a method that finds the globally optimal ranking function that minimizes such an upper bound on the number of misordered pairs. This algorithm, called the Ranking SVM, extended the approach used by Herbrich et al. (2000) for the ordinal regression formulation of learning to rank. Joachims showed that the Ranking SVM formulation reduces to a very similar quadratic optimization

problem to that arrived at by Herbrich et al. We will return to this algorithm in more detail later in this chapter.

Finally, learning from pairwise data has also been addressed as learning the parameters of a model so as to maximize the probability of having observed the preferences seen as training data. One of the most widely known approximate methods for this interpretation is the Glicko ranking algorithm (Glickman, 1999). This probabilistic approach is often used to rank chess players. It assumes player abilities are normally distributed but change with time, and provides a method to estimate the ability and uncertainty in the ability for each player based on the outcomes of (two player) chess games. The algorithm is built on top of the Bradley-Terry model (Bradley & Terry, 1952; Bradley, 1976), which estimates the probability of a pairwise outcome given the abilities of two players. Various extensions to the Bradley-Terry model that allow for ties and order effects have been developed by Dittrich et al. (1998); Huang et al. (2004); Davidson and Beaver (1977); Lancaster and Quade (1983). A related algorithm with very similar goals to that presented by Glickman is TrueSkill, developed by Herbrich and Graepel (2006). A second approach to building probabilistic models, using Gaussian Processes, was also presented by Chu and Ghahramani (2005b; 2005c).

3.2.5 Learning from Entire Rankings

A number of researchers have considered learning to rank given training data that encodes preferences over more than two items. Specifically, generalizing upon pairwise preferences, in this setting training data is provided as k-tuples

(x_1, \dots, x_k) , which define either a total order or a partial order over the items in the tuple.

In the simplest non-learning case, Fagin et al. (2003) considered the problem of finding the best “average” ranking given a two or more complete rankings. In a learning setting, Lebanon and Lafferty (2002) and Lebanon (2007) studied the problem of estimating the probability of total orders given tuples defining partial rankings. Similarly, Kazawa et al. (2005) presented an extension to SVMs for learning given ranked sets of items.

3.3 Two Particular Learning to Rank Algorithms

Of the vast number of algorithms for learning to rank that have been studied, we will use two again later in this thesis. We will therefore now describe those two algorithms in some more detail. First, we briefly describe Ranking SVMs (Joachims, 2002), which generalize ordinal regression SVMs developed by Herbrich et al. (2000). For further details about SVMs in general, we refer readers to Cristianini and Shawe-Taylor (2000). Second, we will briefly present the Glicko algorithm developed by Glickman (1999) for learning to rank chess players. However, in our presentation of the Glicko algorithm, we translate to the terminology of documents and relevance more suitable for learning to rank Web documents.

3.3.1 Ranking SVMs

Given an interactive information ranking system, there exists some joint probability distribution over queries and correct rankings of documents, $P(q, r)$. We would like to observe queries and correct rankings to learn this distribution, and be able to predict the correct ranking given a query q . However, correct rankings can also be described by the relative order of pairs of the ranked documents. The correct relative ordering of any two documents d_i and d_j for a particular query q can be written as

$$d_i \succ_q d_j \tag{3.7}$$

Such a constraint indicates that given query q , document d_i should be ranked above document d_j . Now, suppose that the relevance of any document d_i to query q can be described by a linear function,

$$rel(d_i, q) = w \cdot \Phi(d_i, q), \tag{3.8}$$

where $\Phi(d_i, q)$ maps documents and queries to a feature vector. Intuitively, this feature vector can be thought of as measuring the quality of the match between a document d_i and the query q along any number of dimensions. Elements of this vector may, for example, include features measuring whether the query words occur in d_i or the rank of d_i in the results returned for q by some other search engine. w is a real valued weight vector that assigns weights to each of the features in Φ . The output of this relevance function is thus a real number, where a higher score indicates a document d_i is estimated to be more relevant to the query q . The task of learning a ranking function becomes one of learning an optimal w .

We start by rewriting Equation 3.7 as

$$w \cdot \Phi(d_i, q) > w \cdot \Phi(d_j, q). \quad (3.9)$$

In accordance with a standard large-margin approach, we would like this preference (or *constraint*) to be satisfied by at least a fixed margin. However, given a large number of pairwise constraints as training data, it may be the case that no w exists that satisfies all the constraints. Hence we also add a non-negative slack variable ξ_{ij} that can be non-zero to satisfy the constraint, as is done with standard classification SVMs. This yields a preference constraint over w .

$$w \cdot \Phi(d_i, q) \geq w \cdot \Phi(d_j, q) + 1 - \xi_{ij} \quad (3.10)$$

Given a large number of such constraints for many different pairs of documents and queries, it is NP hard to find the vector w that minimizes the number of constraints that are not satisfied when $\xi_{ij} = 0$. However, we can minimize an upper bound on the number of violated constraints, $\sum \xi_{ij}$. Simultaneously regularizing to avoid overfitting leads to the following convex quadratic optimization problem:

$$\begin{aligned} & \min_{w, \xi_{ij}} \frac{1}{2} w \cdot w + C \sum_{ij} \xi_{ij} \\ & \text{subject to} \\ & \forall (q, i, j) : w \cdot \Phi(d_i, q) \geq w \cdot \Phi(d_j, q) + 1 - \xi_{ij} \\ & \forall i, j : \xi_{ij} \geq 0 \end{aligned} \quad (3.11)$$

This optimization problem is known as a ranking support vector machine. Solving it produces a weight vector w that can then be used to score any (document, query) pair. Given a query, sorting all documents by this score produces a ranking.

3.3.2 Tournament Participant Ranking

Consider again the setting that we obtain relative judgments about documents of the form of preferences similar to

$$d_i \succ_q d_j, \quad (3.12)$$

which indicate that d_i is more relevant than d_j to query q . In general, the preferences we collect may be noisy: some may be correct while others will be incorrect. A standard approach to modeling noise in pairwise comparisons is to assume that the probability of observing a particular preference is determined by the Bradley-Terry model (Bradley & Terry, 1952):

$$P(d_i \succ_q d_j) = \frac{rel(d_i, q)}{rel(d_i, q) + rel(d_j, q)}, \quad (3.13)$$

where $rel(d_i, q)$ is a measure of the relevance of d_i to query q . The Bradley-Terry model can be reparameterized setting $rel(d_i, q) = 10^{\mu_i^q/\sigma}$ where σ is a known, global and fixed parameter. Further, suppose that each document d_i has some true relevance μ_i^q to the query q . We write the set of document relevance values for all documents in a collection \mathcal{C} as

$$M = (\mu_1^q, \dots, \mu_{|\mathcal{C}|}^q). \quad (3.14)$$

Assuming the pairwise judgments are independent, we can define the probability of observing some set \mathcal{D} of relative relevance statements as

$$P(\mathcal{D} | M = (\mu_1, \dots, \mu_{|\mathcal{C}|})) = \prod_{d_i \succ_q d_j \in \mathcal{D}} P(d_i \succ_q d_j | \mu_i, \mu_j) \quad (3.15)$$

$$= \prod_{d_i \succ_q d_j \in \mathcal{D}} \frac{1}{1 + 10^{-(\mu_i - \mu_j)/\sigma}} \quad (3.16)$$

$$\nu_i \leftarrow \nu_i + \frac{q}{\frac{1}{\sigma_i^2} + \frac{1}{\delta^2}} g(\sigma_j^2) (s_i - E(s|\nu_i, \nu_j, \sigma_j^2)) \quad (3.17)$$

$$\sigma_i^2 \leftarrow \left(\frac{1}{\sigma_i^2} + \frac{1}{\delta^2} \right)^{-1} \quad (3.18)$$

where

$$q = \frac{\log 10}{400}$$

$$g(\sigma^2) = \frac{1}{\sqrt{1 + 3q^2\sigma^2/\pi^2}}$$

$$E(s|\nu_i, \nu_j, \sigma_j^2) = \frac{1}{1 + 10^{-g(\sigma_j^2)(\nu_i - \nu_j)/400}}$$

$$\delta^2 = \frac{1}{q^2 g(\sigma_j^2)^2} \times \frac{1}{E(s|\nu_i, \nu_j, \sigma_j^2)(1 - E(s|\nu_i, \nu_j, \sigma_j^2))}$$

Figure 3.2: The Glicko update equations, which describe how the estimated relevance ν_i and estimated variance σ_i^2 for document d_i should be updated following a comparison to document d_j .

Given this likelihood model, and assuming a Gaussian prior over the values μ_i^q , Glickman showed that the values μ_i^q can be estimated iteratively from a set of preferences provided as training data. Given an initial estimate of document relevance (player ability in the context of chess) ν_i and error in the estimate σ_i , this algorithm provides a set of approximate online update equations for maintaining the estimated relevance and error as pairwise preferences are collected. The update to the estimates for d_i following a single comparison to d_j (where s_i is 1 if d_i wins and 0 otherwise) is presented in Figure 3.2.

3.4 Summary

This chapter has described typical performance measures used to evaluate ranking systems, and described five classes of learning algorithms commonly used to learn them. In particular, we have seen that there are many different measures used for evaluating ranking performance. However, for most it is unclear how well they relate to ranking performance from a user perspective. This suggests an open question in what measures should in fact be optimized. Further, we also discussed how relevance data in different forms can be used to learn to rank. We saw that there are many algorithms that are amenable to learning from pairwise preference judgments as can be collected from implicit feedback. Finally, two algorithms were presented in some more detail, namely the Ranking SVM and Glicko chess rating system.

CHAPTER 4

LEARNING FROM IMPLICIT FEEDBACK ENCODED IN QUERY CHAINS

In this chapter, we consider further the question of obtaining relevance judgments from users of a search system. We will show how to extend previous approaches for learning from clickthrough data to obtain substantially more useful training data. In addition to evaluating the validity of the data collected in this new way, we will show that it can then be used to learn an improved ranking function in a real search engine. The work presented in this chapter was originally published in (Radlinski & Joachims, 2005b).

4.1 Introduction

As we have seen in earlier chapters, there has been substantial work in learning to rank documents. This is largely motivated by the difficulty in manually constructing well performing ranking functions. For this learning task, training data can be collected in at least two ways. The first approach, most commonly used in the past, has relied on asking expert judges to rate the relevance of documents to queries. However, as we have seen earlier in this thesis, such data is expensive and slow to collect, with numerous difficulties to overcome. The alternative of recording user interactions and inferring relevance judgments from this implicit feedback has received less attention.

In particular, previous research in learning to rank from clickthrough data has considered each query independently. However, studies of user behavior in Web search engines (for instance by Lau and Horvitz (1999) and by Silverstein

et al. (1998)) have observed that users often run more than just one query when they visit a search engine. Rather, users tend to perform a sequence of queries for any given question. This likely indicates that the results shown for the first query often do not satisfy the users' information needs. Sessions typically consist of 2 to 3 queries (as observed by Beeferman and Berger (2000); Cucerzan and Brill (2004); Jones and Fain (2003); Lau and Horvitz (1999)). Query chains were also observed in the eye tracking study performed by Granka et al, where mean query chain length was 2.2 queries (although the particular questions asked and the laboratory environment would be expected to have an influence on this value). This suggests that when working with implicit feedback, we should consider user behavior over entire sessions rather than just a single query.

We will show that ignoring query chains when learning from clickthrough data ignores valuable information that is hidden in the sequence of queries and clicks in a search session. For instance, if we repeatedly observe the query "special collections" followed by another for "rare books" on a library search system, we may deduce that Web pages relevant to "rare books" may also be relevant to "special collections". Moreover, when queries are considered independently, we can only infer implicit feedback on a few results at the top of the result set for each query because users very rarely look further down the list. The advantage of using query chains is that we can also deduce relevance judgments on the many more documents seen during an entire search session.

This chapter will describe how to interpret user behavior as preference judgments that provide relative relevance information about documents within individual query result sets, and between documents returned by different queries within the same query chain. The method used to generate the preference judg-

ments is validated using a controlled user study. We will then show how an adapted ranking SVM can be used to learn a ranking function from the preference judgments, evaluating on a real-world Web search system, the Cornell University library⁴ Web search.

An important innovation in this chapter is that we also learn a more general ranking function than previous work, by learning an association between query words and specific documents. Such a general approach has been used previously to learn to generate abstracts by Scholer and Williams (2002), but not to learn ranking functions. Prior approaches for learning to rank cannot learn to associate “new” documents with a given query because they usually only combine or re-order results obtained from one or more static ranking functions. In particular, given a query q , they cannot learn to retrieve any document not originally returned for q . Coming closest to solving this limitation previously, the method presented by Kemp and Ramamohanarao (2002) could be extended with query chains. However, they assume implicit absolute feedback, making their approach more likely to be susceptible to bias and noise, as seen in Chapter 2 and will be reviewed again later in this chapter.

4.2 Related Work

We now summarize the most closely related work to that presented in this chapter. When learning to rank, the method by which training data is collected offers an important way to distinguish between different approaches. This data usually consists of a set of statements as to the relevance of a document, or set

⁴<http://library.cornell.edu/>

of documents, to a given query. Such relevance judgments are either collected explicitly by asking experts (or users), or implicitly by observing user behavior and drawing conclusions. Moreover, the statements can be absolute or relative. Absolute feedback involves statements that a particular document is, or is not, relevant to a query. Relative feedback involves statements that a particular document is more relevant to a query than some other document.

Most previous work in learning to rank has assumed absolute relevance judgments. On the one hand, a number of methods in ordinal regression use explicit feedback to learn to rank, such as work by Herbrich et al. (2000), Crammer and Singer (2001) and Rajaram et al. (2003). However, explicit feedback is expensive to collect, making typical labeled data sets small and difficult to work with. A number of researchers have collected absolute relevance judgments implicitly from clickthrough logs, such as Boyan et al. (1996); Cohen et al. (1999); Kemp and Ramamohanarao (2002); Tan et al. (2004). They postulate that documents clicked on in search results are highly likely to be relevant. For example, Kemp and Ramamohanarao (2002) present a learning search engine using document transformation. They assume results clicked on are relevant to the query and append the query to these documents. However, implicit clickthrough data has been shown to be biased as it is relative to the retrieval function quality and ordering (Joachims, 2002; Joachims et al., 2005). This makes its interpretation as absolute feedback of questionable accuracy.

Cohen et al. (1999) and Joachims (2002) proposed transforming training examples collected from usage logs into relative pairwise preferences. Both approaches consider learning a ranking function from these preference judgments, along similar lines as this work. However, in contrast to our method, their learned

function is limited to a combination of rankings given by a fixed set of manually constructed rankers. This approach of learning a combination of functions is also used by most other work in this area (for example, by Bartell and Cottrell (1995); Bartell et al. (1994); Boyan et al. (1996); Oztekin et al. (2002)).

Additionally, while previous work has attempted to predict or suggest query reformulations (for example, Lau and Horvitz (1999); Jones and Fain (2003); Beeferman and Berger (2000); Furnas (1985); Wang and Zhai (2007)), reformulations have never been used to learn better retrieval functions. Of particular relevance to this work, Cucerzan and Brill (2004) used aggregate query frequency statistics to learn to correct spelling mistakes in queries. Looking specifically at reformulations, the approach presented here also automatically learns to associate misspelled queries with appropriate documents, although does so in a more general framework.

4.3 Analysis of User Behavior

Before we can infer implicit preference judgments from log files, we need to understand how users assess search results. While this question was studied in depth in Chapter 2, we now recapitulate the key observations.

Granka et al. (2004) performed an eye tracking study to observe how users assess the results returned by a search engine, and select the links they click on. Thirty-six undergraduate student volunteers were instructed to search for the answers to ten queries that involved finding a specific Web page or particular information. The subjects were asked to start from the Google search page and

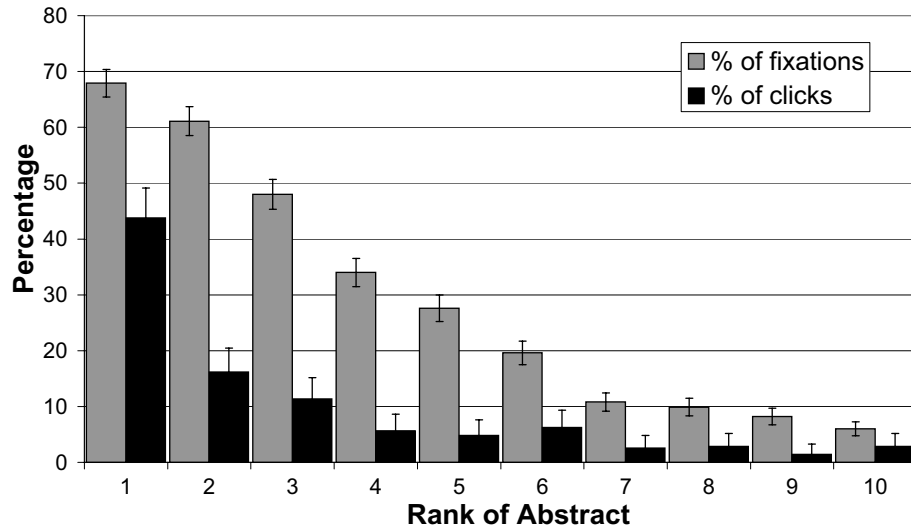


Figure 4.1: Percentage of time an abstract was viewed/clicked on depending on the rank of the result (from Joachims et al. (2007)).

find the answers. There were no restrictions on what queries they may choose, how and when to reformulate queries, or which links to follow. Users were told that the goal of the study was to observe how people search the Web, but were not told of the specific interest in their behavior on the results page of Google. All clicks, the results returned by Google, and the pages connected to the results were recorded by an HTTP proxy. Movement of the eyes was recorded using a commercial eye tracker.

Figure 4.1 shows the fraction of the time users looked at, and clicked on, each of the top 10 search results for a query. It tells us that users usually look at least at the top two result abstracts. Interestingly, note that despite the top two documents receiving almost equal attention, users were much more likely to click on the first result. Further analysis of which abstracts were observed prior to clicking showed that users usually scan the results in order from top to bottom, while also usually looking at the next abstract below any they click on.

We conclude that users typically look at most of the results from the first to the one below the last one clicked on.

4.4 Implicit Feedback Strategies

We now detail our approach for generating relative preference feedback from clickthrough logs. Following this, we will present an evaluation of this approach using results from the eye tracking study described in Chapter 2.

Consider the queries shown in Figure 4.2. The first shows the results presented to a user running the query “NDLF” on the Cornell University library search engine. The user is searching for the National Digital Library Foundation website, but has retrieved only meeting notes that reference people working for the NDLF. The desired page is not in these results, most probably because it does not contain the word “NDLF”. The second query is a search performed in Google by a participant in the eye tracking study in attempting to find the name of the house that Ezra Cornell built for himself. We get many results, but in fact none of the top 10 contain any relevant information. In both cases, single-query implicit feedback will not be informative because no relevant documents were retrieved. In the former case, the results simply do not contain any documents relevant to the query. In the latter, if there is a relevant document, the user is unlikely to look far enough in the results to see it.

On the other hand, after both of these queries, we observed that the user continued running other queries. Often, such later queries were more successful. If a user found a relevant document with a later query, it is reasonable to assume

<i>Query 1: NDLF</i>	
1.	http://.../staffweb/SMG/SMG970319.html
2.	http://.../staffweb/SMG/SMG970226.html
3.	http://.../staffweb/SMG/SMG960417.html
4.	http://.../staffweb/SMG/SMG960403.html
5.	http://.../staffweb/SMG/SMG960828.html

<i>Query 2: "Ezra Cornell" residence</i>	
1.	Dear Uncle Ezra – Questions for Tuesday, May. . .
2.	Dear Uncle Ezra – Questions for Thursday, . . .
3.	Ezra Cornell had close Albion ties
4.	October 1904 – Albion 100 Years Age
5.	Cornell competes with Off-Housing market
	⋮

Figure 4.2: Two example queries and result sets.

that the user would have preferred to have seen the relevant document over the results actually returned for the earlier queries. Recognizing the information necessary to make these deductions is present in search engine log files, we next describe specific strategies for generating such preference feedback from query chains. We defer a discussion of how to group queries into query chains to Section 4.5.

We generated preferences using six strategies. These strategies are illustrated in Figure 4.3. The first two strategies show preferences that can be inferred without query chains, and are very similar to two strategies described in Chapter 2. The first one, $\text{CLICK} >_q \text{SKIP ABOVE}$, was proposed by Cohen et al. (1999) and Joachims (2002). This strategy proposes that given a clicked-on document (marked **x** in the figure), any higher ranked document that was not clicked on is likely less relevant. The preference is indicated by an arrow labeled with the query, to show that the preference is with respect to that query. We expect this to be valid because users view results in order, and a user is unlikely to click

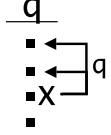
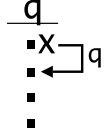
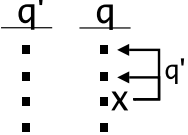
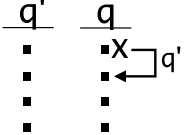
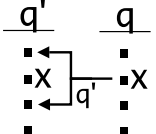
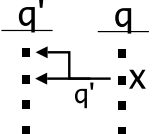
<p>CLICK $>_q$ SKIP ABOVE</p> 	<p>CLICK FIRST $>_q$ NO-CLICK SECOND</p> 
<p>CLICK $>_{q'}$ SKIP ABOVE</p> 	<p>CLICK FIRST $>_{q'}$ NO-CLICK SECOND</p> 
<p>CLICK $>_{q'}$ SKIP EARLIER QUERY</p> 	<p>CLICK $>_{q'}$ TOP TWO EARLIER QUERY</p> 

Figure 4.3: Feedback strategies. We either consider a single query, q , or a query q that has been preceded by a query q' . Given a query, a dot represents a result document and an x indicates the result was clicked on. We generate a constraint for each arrow shown, with respect to the query marked.

on a document he or she considers less relevant than another document she observed but did not click on. Note that these preferences are not stating that the clicked-on document *is* relevant, rather that it is *more likely* to be relevant than the ones not clicked on above. The second strategy, CLICK FIRST $>_q$ NO-CLICK SECOND makes use of the fact that users typically view both of the top two results before clicking. It states that if the first document is clicked on, but the second is not, the first is likely more relevant than the second. It seems reasonable to assume that having considered two options, the user is likely to click on the more relevant one.

The next two strategies are identical to the first two except that they generate feedback with respect to *previous* queries. The intuition behind this is that since the two queries belong to the same query chain, the user is looking for the same information with both. Had the user been presented with the new results for the earlier query, he or she would have preferred the clicked-on document over those skipped above.

The last two strategies make the most use of query chains. The strategy $\text{CLICK} >_{q'} \text{SKIP EARLIER QUERY}$ states that a clicked-on document is preferred over any result for an earlier query q' (within the same query chain) that we can be reasonably confident the user looked at, but which was not clicked on. These documents include all the documents down to one below the lowest ranked clicked document for q' , since the eye tracking study revealed that users usually look one document past the last one clicked on. Note that this judgment is made with respect to the earlier query,⁵ q' . Also, note that these preferences are particularly unlike any generated without considering query chains. They tend to indicate that a very low ranked result for the original query (as the result clicked on in the later query was presumably not returned at high rank for q' , but must have been ranked somewhere since ranking functions can compute scores for all documents) is preferred over those results that were ranked highly for q' . In the event that no documents were clicked on in the earlier query, we use the earlier observation that users usually look at the top two results. This is exploited in the feedback strategy $\text{CLICK} >_{q'} \text{TOP TWO EARLIER QUERY}$ by generating preferences for the top two results. In the unusual case where there

⁵It is unnecessary to state the same thing with respect to the later query q because presumably the preference is already satisfied, or the user would have seen the same result earlier.

<u>q1</u>	<u>q2</u>	
d1	d4 x	$d_2 >_{q1} d_1$ $d_4 >_{q2} d_5$ $d_4 >_{q1} d_5$
d2 x	d5	$d_4 >_{q1} d_1$ $d_4 >_{q1} d_3$
d3	d6	

Figure 4.4: Sample query chain and the feedback that would be generated using all six feedback strategies. Two queries were run, and each returned three documents. One document in each query was clicked on. $d_i >_q d_j$ means that d_i is preferred over d_j with respect to the query q .

are not enough results to the earlier query to use these strategies, we select a random document as if it had been at the end of the results.

Ultimately, given some query chain, we make use of all six strategies to generate the preference feedback. Figure 4.4 gives a sample query chain and the feedback that would be generated in this case.

4.5 Detecting Query Chains

To use query chains, we need a method to identify them. A number of researchers, including Beeferman and Berger (2000); Furnas (1985); Jones and Fain (2003) have previously successfully learned to predict query reformulations. Their success on this task suggests that the problem of detecting query chains, which we have to address, is feasible. We now turn to this question.

As a basis for evaluating potential approaches, we created a dataset using search logs from the Cornell University library Web search engine. We manually labeled query chains in the logs for a period of 5 weeks. The search logs recorded the query, date, IP address, results returned, number of clicks on the results and

a session id uniquely assigned to each user. We extracted the list of queries, grouped them by IP address and sorted them chronologically. Queries from an IP address with no other queries within 24 hours were automatically marked as not belonging to a query chain. This resulted in a dataset of 1,285 queries. Two judges then individually grouped the queries into query chains manually, using search engines to resolve uncertainties (for instance, one query for a person was followed by one for the department where the person is a faculty member). Finally, the judges combined their identified query chains, resolving the small number of disagreements between themselves through further investigation.

For each pair of queries from the same IP address within half an hour, we generated a training example by constructing a feature vector. The training example was labeled using the query chains identified manually. If the two queries belonged to the same query chain the example was labeled as positive. Otherwise it was labeled as negative. This led to 3,418 training examples of which 3,096 were labeled as positive. The feature vector generated given two queries $q1$ and $q2$ consisted of the 16 features shown in Table 4.1.

Using this data, we trained a number of SVM classifiers with various parameters. The classifiers learned tended to label almost all examples as positive. Among our best performing models was an SVM with an RBF kernel with $C = 100$ and $\gamma = 1$ (Joachims, 1999). Evaluating using five-fold cross validation, it gave an average accuracy of 94.3% and precision of 96.5%. This compares to a accuracy and precision of 91.6% for a simple non-learning strategy where we assume all pairs of queries from the same IP address within half an hour of each

Table 4.1: Features used to learn to classify query chains. $q1$ and $q2$ are two queries at times $t1$ and $t2$, with $t1 < t2$. $r1$ and $r2$ are the respective result sets, with $r1_{top}$ and $r2_{top}$ being the top 10 results.

<i>CosineDistance</i> ($q1, q2$)
<i>CosineDistance</i> (doc ids of $r1_{top}$, doc ids of $r2_{top}$)
<i>CosineDistance</i> (abstracts of $r1_{top}$, abstracts of $r2_{top}$)
<i>TrigramMatch</i> ($q1, q2$)
<i>ShareOneWord</i> ($q1, q2$)
<i>ShareTwoWords</i> ($q1, q2$)
<i>SharePhraseOfTwoWords</i> ($q1, q2$)
<i>NumberOfDifferentWords</i> ($q1, q2$)
$t2 - t1 \leq \{5, 10, 30, 100\}$ seconds
$t2 - t1 > 100$ seconds
<i>NormalizedNumberOfClicks</i> ($r1$)
<i>NormalizedMin</i> ($ r1 , r2 $)
<i>NormalizedMax</i> ($ r1 , r2 $)

other are in the same query chain⁶. As this difference is relatively small, and computing some of the feature values described above for every pair of queries is relatively expensive (in particular those that depend on the abstracts retrieved), we decided to rely on our simple half-hour heuristic. We judged that a precision of over 90% is sufficient for our present purposes. However, it is worth keeping in mind that such a heuristic may not necessarily perform as well in a general Web search setting as in a library search setting.

We also considered extending the half-hour window on our training data in order to increase recall, but decided that we were recognizing a sufficient number of query chains without doing so. In particular, given the large quantity of clickthrough data that can be easily collected, when obtaining relevance judgments it is more important that identified query chains are correct, rather

⁶Note that this heuristic is not necessarily transitive. Suppose we observe three queries separated by 20 minutes. While each sequential pair of queries will be considered as belonging to the same query chain, the two end queries will not be considered in the same query chain.

than that all query chains are found (in other words, precision is more important than recall).

However, to gain some insight into the properties of query chains we trained a linear SVM using the same data and computed the total weight on each feature. The features with largest positive weight were $CosineDistance(q1, q2)$, which measures the textual similarity between $q1$ and $q2$, and $CosineDistance(doc\ ids\ of\ r1_{top}, doc\ ids\ of\ r2_{top})$, which measures the overlap between the documents in the top 10 results. This indicates that if two queries are similar, or if they retrieve many of the same documents, then they are more likely to be in the same query chain. The feature with largest negative weight measures the minimum number of results returned by either query normalized between 0 and 1, $NormalizedMin(|r1|, |r2|)$. This indicates that if one of the queries returns few results, the queries are more likely to be in a query chain. Our interpretation is that if a query returns no results, the user is more likely to run a second query.

4.6 Accuracy of the Feedback Strategies

We now assess the accuracy of the feedback strategies proposed in Section 4.4. To determine the accuracy of each individual strategy, we collected additional data following the eye-tracking study described in Chapter 2. For 16 subjects, we evaluated whether the preferences derived from the feedback strategies across multiple queries agree with explicit relevance judgments made by independent judges.

Table 4.2: Accuracy of the strategies for generating pairwise preferences from clicks. The base of comparison are the explicit page judgments. Note that the first two cases cover two preference strategies each.

Strategy	Accuracy
CLICK $>_q$ SKIP ABOVE	78.2 \pm 5.6
CLICK FIRST $>_q$ NO-CLICK SECOND	63.4 \pm 16.5
CLICK $>_q$ SKIP EARLIER QUERY	68.0 \pm 8.4
CLICK $>_q$ TOP TWO EARLIER QUERY	84.5 \pm 6.1
Inter-Judge Agreement	86.4

Specifically, we grouped the results observed by each user by query chain and collected explicit relevance judgments from five judges. The judges were asked to weakly order all results encountered during each query chain according to their relevance to the question asked of the user. To avoid biasing the judges, the order in which results were presented to the judges was randomized and the judges were not given the abstracts Google used when presenting the results. Some of the query chains were assessed by two judges for inter-judge agreement verification. Whenever two judges expressed a strict preference between two pages, they agreed in the direction of preference in 86.4% of the cases.

Table 4.2 summarizes the extent to which the preferences generated agree with the explicit judgments. The table shows the percentage of preferences generated from clicks using the above strategies that agree with the strict preferences provided by the relevance judges. The first two lines in the table show the accuracy of the strategies that do not exploit query chains. The CLICK $>_q$ SKIP ABOVE strategy is 78.2% accurate, which is substantially and significantly better than a random baseline of 50%. Furthermore, it is reasonably close in accuracy to the average agreement of 86.4% between the explicit judgments from different judges. This serves as an upper bound for the accuracy one could ideally expect even from explicit user feedback. The second within-query strategy, CLICK FIRST

$>_q$ NO-CLICK SECOND, appears less accurate. However, since it produces fewer preferences (specifically, only on queries where the user clicked exclusively on the first link), the confidence intervals are large. Independent of the accuracy, the preferences from this strategy are probably less informative, since they only confirm the current ranking and never suggest a reordering.

Lines 3 and 4 in Table 4.2 show the accuracy of the two strategies that exploit query chains. Both $\text{CLICK } >_{q'} \text{ SKIP EARLIER QUERY}$ and $\text{CLICK } >_{q'} \text{ TOP TWO EARLIER QUERY}$ are significantly more accurate than random. In particular, the accuracy of $\text{CLICK } >_{q'} \text{ TOP TWO EARLIER QUERY}$ is very close to the average agreement between judges.

A possible explanation for the difference in accuracy between the two query chain strategies is that they apply to different types of query chains. While $\text{CLICK } >_{q'} \text{ SKIP EARLIER QUERY}$ is applied when the previous query received a click, the strategy $\text{CLICK } >_{q'} \text{ TOP TWO EARLIER QUERY}$ is applied precisely in the opposite case. To investigate the effect of this difference, we also evaluated a variant of $\text{CLICK } >_{q'} \text{ TOP TWO EARLIER QUERY}$. This variant generates preferences analogous to $\text{CLICK } >_{q'} \text{ TOP TWO EARLIER QUERY}$, but in chains where the previous query did receive a click (but excluding the clicked results). The accuracy of this strategy is $67.7\% \pm 9.4$, indicating that the absence of a click followed by another query with a click is particularly strong evidence regarding the relevance of the results of the earlier query.

Overall, we conclude that the preferences generated from the clickthrough logs are reasonably accurate and that they convey information regarding the user's preferences.

4.7 Learning Ranking Functions

Given log files recording user behavior on a Web search engine, we have shown that log records can be transformed into preference judgments in Section 4.4, after segmenting the queries into query chains as described in Section 4.5. Next, we present an algorithm to learn from these preferences, which we then evaluate using a real-world search engine.

As described in Chapter 3, we assume as input preference judgments over documents d_i and d_j for a given query q to be of the following form:

$$d_i \succ_q d_j \tag{4.1}$$

Such a preference judgment indicates that d_i is preferred over d_j given q . As our retrieval model, we chose a linear retrieval function:

$$rel(d_i, q) = w \cdot \Phi(d_i, q) \tag{4.2}$$

where $\Phi(d_i, q)$ (which we define shortly) is a function that maps documents and queries to a feature vector. This feature vector describes the extent to which the query q matches document d_i according to any number of dimensions. w is a weight vector that describes the importance of each of the features in Φ , thus giving us a real valued retrieval function where a higher score indicates a document d_i is estimated to be more relevant to the query q . The task of learning a ranking function becomes one of learning an optimal w .

4.7.1 Ranking SVMs

We used a modified Ranking SVM (Joachims, 2002), which was described in some more detail in Chapter 3, to learn w in Equation 4.2. Summarizing this algorithm, we start by rewriting Equation 4.1 as

$$w \cdot \Phi(d_i, q) > w \cdot \Phi(d_j, q) \quad (4.3)$$

We then add a margin, and non-negative slack variables to allow some of the preference constraints to be violated, as is done with classification SVMs. This yields a preference constraint over w .

$$w \cdot \Phi(d_i, q) \geq w \cdot \Phi(d_j, q) + 1 - \xi_{ij} \quad (4.4)$$

As noted in Chapter 3, we cannot efficiently find a w that minimizes the number of violated constraints. However, we can minimize an upper bound on the number of violated constraints, $\sum \xi_{ij}$. Simultaneously maximizing the margin leads to the convex quadratic optimization problem seen earlier:

$$\begin{aligned} & \min_{w, \xi_{ij}} \frac{1}{2} w \cdot w + C \sum_{ij} \xi_{ij} \\ & \text{subject to} \\ & \forall (q, i, j) : w \cdot \Phi(d_i, q) \geq w \cdot \Phi(d_j, q) + 1 - \xi_{ij} \\ & \forall i, j : \xi_{ij} \geq 0 \end{aligned} \quad (4.5)$$

We will later add more constraints to the optimization problem taking advantage of prior knowledge in the learning to rank setting.

4.7.2 Retrieval Function Model

We now specify the feature mapping $\Phi(d_i, q)$ needed to compute the feature vector given to the Ranking SVM. This definition is key in determining what class of ranking functions that we can learn.

We define two types of features: rank features $\phi_{rank}^f(d, q)$ and term/document features $\phi_{terms}(d, q)$. Rank features serve to exploit existing retrieval functions rel_0^f , while term/document features allow us to learn more fine-grained relationships between particular query terms and specific documents.

First we need a few definitions. Let $T := \{t_1, \dots, t_N\}$ be all the terms (words) in our dictionary. A query q is a set of terms $q := \{t'_1, \dots, t'_n\}$ where $t'_i \in T$. Let $\mathcal{C} := \{d_1, \dots, d_M\}$ be the set of all documents in our collection. We assume the original search engine has a number of available retrieval functions F that provide relevance scores $rel_0^f(d, q)$ for $f \in F$. We define $r_0^f(q)$ as the ordered set of results as ranked by rel_0^f for query q . In the experiments in this chapter, F consists of a single ranking function as provided by Nutch⁷ for the sake of simplicity.

Now,

$$\Phi(d, q) = \begin{bmatrix} \phi_{rank}^{f_1}(d, q) \\ \vdots \\ \phi_{rank}^{f_F}(d, q) \\ \phi_{terms}(d, q) \end{bmatrix} \quad (4.6)$$

⁷An open source search engine implementation available at <http://www.nutch.org/>

$$\phi_{rank}^f(d, q) = \begin{bmatrix} \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 1) \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 2) \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 3) \\ \vdots \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 10) \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 15) \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 20) \\ \vdots \\ \mathbf{1}(\text{Rank}(d \text{ in } r_0^f(q)) \leq 100) \end{bmatrix} \quad (4.7)$$

$$\phi_{terms}(d, q) = \begin{bmatrix} \mathbf{1}(d = d_1 \wedge t_1 \in q) \\ \vdots \\ \mathbf{1}(d = d_M \wedge t_N \in q) \end{bmatrix} \quad (4.8)$$

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

Before looking at the term features $\phi_{terms}(d, q)$, we explore the rank features $\phi_{rank}^{f_i}(d, q)$. For each retrieval function $rel_0^{f_i}$ we have 28 rank features (for ranks 1, 2, 3, 4, ..., 10, 15, 20, 25, ..., 100). Each of these is set to 1 if the rank of the document d in $r_0^{f_i}$ is at or above the specified rank.

The rank features allow us to learn weights for the individual ranks of the original search functions. This allows the learned ranking function to combine different retrieval functions with different weights, as is done in prior work described earlier. We do not consider the specific scores assigned by rel_0^f in order to account for potentially different magnitudes of the scores from different retrieval functions. This also ensures that our method could generalize to settings where we do not have access to the scores assigned to documents but only the document

ranks. Additionally, by decomposing over the ranks, we can learn scores that are non-linear in the rank. As an example, if some document d is at rank 4 given query q and using retrieval function f_1 then $\phi_{rank}^{f_1}(d, q) = [0, 0, 0, 1, \dots, 1]^T$. If a document is not ranked in the top 100 by the retrieval function f_1 , then all the features of $\phi_{rank}^{f_1}$ are 0. This means that documents not ranked in the top 100 results by a retrieval function $rel_0^{f_i}$ are indistinguishable using the $\phi_{rank}^{f_i}$ features (although we could increase the maximum rank considered arbitrarily). We chose this cutoff as it is extremely rare for users to look beyond the top 100 results. As in this study we only consider one ranking function, the rank features simply tell us how much weight to place on the documents ranked highest by Nutch.

We also have NM term/document features. For convenience, let $\phi_{term}^{i,j}(d, q)$ correspond to the feature with d_i and t_j in $\phi_{terms}(d, q)$. There is one feature for every (term, document) pair in $T \times \mathcal{C}$. The term/document features allow the ranking function to learn associations between specific query words and documents by assigning a non-zero value to the appropriate weight. This is usually an extremely large number of features, although most never appear in our training data and can thus be ignored. Furthermore, the feature vector $\phi_{terms}(d, q)$ is very sparse: For any particular document $d \in \mathcal{C}$, given a query with $|q|$ terms, only $|q|$ of the $\phi_{term}^{i,j}(d, q)$ features are set to 1. Specifically, only the terms for one i value (where $d = d_i$) and with $t_j \in q$ are non-zero. The sparsity makes this problem well suited for solving using support vector machines. A positive value of the weight $w_{term}^{i,j}$ associated with the feature $\phi_{term}^{i,j}$ indicates that d_i is more likely to be relevant to queries containing the term t_j , while a negative value means the opposite.

4.8 Adding Prior Knowledge

We also have additional prior knowledge that should be incorporated into this learning problem. Absent any other information, documents with a higher rank in the original ranking should be ranked higher in the learned ranking system. This is intuitive because on average we would expect the document relevance to be a decreasing function of the original rank of the documents, unless the original ranking function is particularly poor. We define such additional constraints in this section.

It is also of practical importance to add these constraints. In our training data almost all of the relevance judgments generated state that a lower ranked document is preferred to a higher ranked document. Without additional constraints, a trivial and undesirable solution to the optimization problem in Equation 4.5 would be one that reverses the original ranking by assigning a negative value to each of the weights corresponding to rank features in Φ . To see this, consider again Figure 4.3. The $\text{CLICK} >_{q(q')} \text{SKIP ABOVE}$ preferences would be satisfied if the rankings were reversed. These preferences are much more common than $\text{CLICK FIRST} >_{q(q')} \text{NO-CLICK SECOND}$ preferences. In the last two preferences classes, the preferred document is also presumably somewhere much lower in the results for q' (if it is not in the results, we can think of it as being at the bottom of the results), and hence the preferences would also be satisfied if the entire ranking were reversed.

We add additional hard constraints to the optimization problem specified in Equation 4.5. These constraints require that weights for each of the rank features must be greater than a constant positive value w_{min} :

$$\forall i \in [1, 28|F|]. \quad w_{rank}^i \geq w_{min} \quad (4.9)$$

Intuitively, w_{min} limits how quickly the original ranking is changed by training data. To see this, briefly consider a setting where we have a single ranking function f and a query $q = t'$ that returns at least 100 results. Let d_i be the document ranked at position i in $r_0^f(q)$. In this case,

$$\begin{aligned} \phi_{rank}^f(d_{100}, q) &= [0, \dots, 0, 0, 1]^T \\ \phi_{rank}^f(d_{95}, q) &= [0, \dots, 0, 1, 1]^T \\ &\vdots \\ \phi_{rank}^f(d_1, q) &= [1, \dots, 1, 1, 1]^T \end{aligned}$$

Calling the part of w that corresponds to rank features w_{rank} , from Equation 4.9 we then get

$$\begin{aligned} w_{rank} \cdot \phi_{rank}^f(d_{100}, q) &\geq w_{min} \\ w_{rank} \cdot \phi_{rank}^f(d_{95}, q) &\geq 2w_{min} \\ &\vdots \\ w_{rank} \cdot \phi_{rank}^f(d_1, q) &\geq 28w_{min} \end{aligned}$$

Now say we have a document d that is preferred over d_1 but is not in the original results. d would be ranked highest if $rel(d, q) > rel(d_1, q)$. We know

from Section 4.7.2 that only $\phi_{term}^{t',d}(d, q)$ is non-zero in $\phi_{terms}(d, q)$. Expanding and simplifying, this would imply:

$$w_{terms} \cdot \phi_{terms}(d, q) \geq 28w_{min} + w_{terms} \cdot \phi_{terms}(d_1, q) \quad (4.10)$$

$$w_{term}^{d,q} \geq 28w_{min} + w_{terms}^{d_1,q} \quad (4.11)$$

where $w_{term}^{\alpha,\beta}$ corresponds to $\phi_{term}^{\alpha,\beta}(d, q)$.

The larger w_{min} , the larger in magnitude $w_{term}^{d,q} - w_{terms}^{d_1,q}$ must be for this to happen. A ranking SVM minimizes over $\frac{1}{2}w \cdot w + C \sum \xi_{ij}$ (Equation 4.5), so the terms will only become large if there is sufficient training data to support a reordering.

4.9 Evaluation

To collect training data, we implemented a real-world, publicly accessible search engine called Osmot. The search engine implements a full-text search of Web pages maintained or indexed by the Cornell University library. This collection includes over 13,500 Web pages. We used the Nutch search engine⁸ as a starting point, with the Osmot search engine effectively being a wrapper around Nutch that implements logging, log analysis, learning, reranking and evaluation functionality. In the experiments in this paper, we chose Nutch’s built-in retrieval function as the baseline to compare against and build upon. The Nutch retrieval function is based on the cosine distance and incorporates several modifications to make it more suitable for Web search including special cases for phrase matches and HTML fields. Osmot is available for download by the research community⁹.

⁸<http://www.nutch.org/>

⁹<http://www.cs.cornell.edu/~filip/osmot/>

The name is derived from the word *osmosis*, as learning from implicit feedback is almost as good as learning from users by osmosis.

We collected training data from the Cornell University library search engine using the ranking function built into Nutch between June and December 2004. During this time, we recorded user queries and clicks, observing 9,949 queries and 7,429 clicks. While we were collecting this data, the users saw results as ranked by the built-in Nutch retrieval function, which we denote as rel_0 . This gave 120,134 preferences constraints by applying all six strategies introduced earlier. We call these preferences P_{QC} . Of these, 45,610 preferences were generated without using the query chain strategies. We call this subset of the preferences P_{NC} .

After adding the constraints as described in Section 4.8, we trained a ranking SVM for each of the two sets of preferences with a linear kernel and a default value of C using *SVM^{light}* (Joachims, 1999). We set $w_{min} = 1$. Using the preferences P_{QC} we learned a retrieval function rel_{QC} and using the preferences P_{NC} we learned rel_{NC} . The former model has 41,354 support vectors, while the latter has 18,034.

The ranking model learned using query chains, rel_{QC} , instantiated 18,748 features. The number of features instantiated can be expected to grow almost linearly in the size of the document collection, and sub-linearly in the amount of training data collected (depending on overall user search behavior). However, this did not pose a problem for the SVM solver because all the preference judgments were sparse.

4.9.1 Interleaved Evaluation

To evaluate our results, we need an unbiased method for comparing two ranked retrieval functions. For this purpose we use the interleaved ranking method of Joachims (2003), presented on page 43 in Chapter 3. We now describe this technique again at a high level.

Interleaving assesses the relative retrieval quality of two ranking functions. Given two ranking functions, we presented users with a combination of the results from both. We know that users scan results from top to bottom, so we intertwine the results such that there is no presentation bias favoring either ranking function. This evaluation method is built into the Osmot search engine.

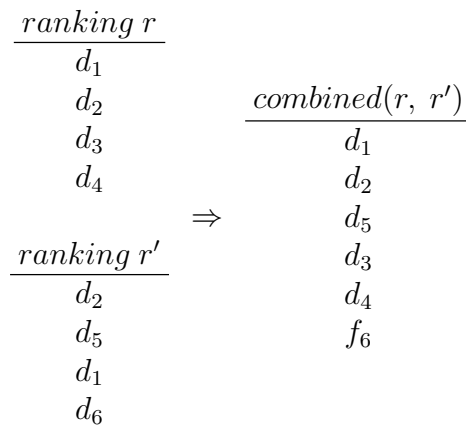


Figure 4.5: Two example rankings with four results each, and the combined outputs we would generate by starting with the top ranked document from ranking r .

Figure 4.5 shows two example rankings, r and r' , from two different retrieval functions as well as a combination of them, $\textit{combined}(r, r')$. The combination is designed such that no matter how many results from the combined ranking the user considers, in expectation the user sees the same number of results from both

r and r' . For instance, in the above example rankings, if the user looks at the top three results in the combined ranking, this user has seen 2 results from r and 2 results from r' . If the user looks at the top five results, he or she has seen 4 and 3 from r and r' respectively. To compensate for a bias toward the results of r , we randomly switch r and r' half the time.

Once we have presented the user with a combined ranking, we can evaluate which of the two rankings is preferred based on the user's clicking behavior. Specifically, we can count how many results from the top of each ranking the user clicked on. For example, in Figure 4.5, suppose the user clicked on d_1 and d_5 . The top three results shown combine the top two positions of rankings r and r' . In this case, we see that the user clicked on one result from the top two of r' and one result from the top two of r . This tells us that the user does not prefer either ranking.

If over a large number of users and queries we see that on average one of the rankings receives more clicks, we can conclude that ranking function is better.

4.9.2 Results and Discussion

We evaluated the ranking functions learned using the preferences inferred from the clickthrough data collected on the Cornell University library search engine from 10 December, 2004 through 18 February, 2005 using interleaved evaluation. When a user connected to the search engine, we randomly selected an experimental condition for that user. The user either saw a ranking combining rel_0 and rel_{QC} or a ranking combining rel_{QC} and rel_{NC} . For consistency, we kept the same combination for the duration of each user's session (otherwise, if the

Table 4.3: Results on Cornell University library search engine. rel_0 is the original retrieval function, rel_{QC} is that trained using query chains, and rel_{NC} is that trained without using query chains. The differences in performance are statistically significant with 99% confidence.

Evaluation Mode	User Prefers		
	Chains	Other	Indifferent
rel_{QC} vs. rel_0	392 (32%)	239 (20%)	579 (47%)
rel_{QC} vs. rel_{NC}	211 (17%)	160 (13%)	855 (70%)

user immediately re-ran the same query, he or she may confusingly get different results).

During the evaluation, we collected about 1200 queries in each evaluation mode. The results for both evaluation modes are shown in Table 4.3. These results show a number of interesting properties. First, 53% of the time rel_{QC} , the ranking function trained using query chains, performs differently to the original ranking function, rel_0 . 30% of the time the two trained ranking functions perform differently. In particular, the first of these values indicates that our method often makes a difference in search engine performance. Given that the original ranking function is reasonable, it would be surprising if these values were much higher. As long as our method does not cause relevant documents that are ranked highly by rel_0 to be lowered in rank, we would see identical performance in cases when rel_0 performs well.

Second, from Table 4.3 we see that rel_{QC} outperforms rel_0 more often than we would expect at random if the two ranking functions were equally good. Using a binomial sign test, and the null hypothesis that the two ranking functions are equally effective, we are able to reject the null hypothesis with 99% confidence.

This establishes that our learned ranking function is an improvement over the original one. Of course, given the new ranking function, we can collect new training data and then re-run the whole learning process. We expect this to further improve ranking performance.

Finally, the model trained using query chains outperforms the model trained without using query chains with 99% confidence, using the same test. This demonstrates that by exploiting the information about query chains present in log files, we are able to see a measurable additional improvement in search engine performance over what we would see without using this extra information.

One may wonder if it makes sense to learn associations between specific query words and documents. Given our initial 9,949 training queries, Table 4.4 shows the top ten words that appeared most frequently in queries. We see that queries tend to be repetitive. Ignoring the three stopwords in the top ten words and collecting the remaining rows in the table, we found that one or more of the remaining seven words appeared in 12% of all queries. At least one of the top 100 words (removing stopwords) appeared in 38% of all queries. Moreover, for many popular queries, there appear to be only a few documents that are truly relevant to the query. Hence it is not surprising that by learning individual query word/document associations, we can see significant improvements in ranking results.

To understand where the improvements are coming from, it is useful to look at the term/document features with largest positive and negative weights. The top and bottom five features are given in Table 4.5. First we consider the top five features, which for the most part describe very sensible associations. The feature for “lexus” is associated with the main homepage of the Lexis-Nexis

Table 4.4: The most common words to appear in queries in the training data, and the fraction of queries in which they occur.

Word	Fraction of queries
of	3.56 %
library	2.75 %
bibliography	2.60 %
and	2.55 %
annotated	2.42 %
reserve	2.32 %
citation	1.99 %
web	1.48 %
the	1.41 %
course	1.33 %

library resource. This is clearly a spelling correction, with a search for “lexus” originally returning no results. The same search now places the correct document at the top of the results. The feature for “ebook” returns the main ebooks Web page. A search for ebook previously returned seven results, none of which were particularly useful. The top one, titled “Answers to Frequent Job Searching Research Questions”, happened to mention access to ebooks from off campus. The feature for “reuleaux” is associated with an FAQ page about the Cornell University library digital collections. The Web page provides a clear link to a site that describes models designed by Professor Reuleaux. This contrasts with the original top result being a broken link, and the second result being a newsletter with only passing reference to the model collection. The feature for “and” is of little practical interest (we did not remove stopwords). Finally, the fifth word “oed” is an acronym for the “Oxford English Dictionary”. The associated document clearly links to it, in contrast with the original top result which was an information bulletin showing a set of screen shots how to get to the OED among other things.

Table 4.5: Five most positive and most negative feature weights in the ranking function learned using query chains on the Cornell University Library search engine

Word	Document	Weight
lexus	Lexis-Nexis Academic Universe	22.8
ebook	CUL eContent Collection	22.5
reuleaux	CUL Digital Collections	21.8
and	Printable News and Notes 07/03	19.6
oed	Dictionaries and Encyclopedias	19.5
ndlf	Management meeting notes 03/97	-21.0
ndlf	Management meeting notes 02/97	-20.6
ndlf	Management meeting notes 04/96	-19.5
ndlf	Management meeting notes 04/96	-18.6
instruction	Library Research Workshops	-18.3

The five features with the most negative weights in Table 4.5 are equally interesting. Four of them relate to meeting notes mentioning the National Digital Library Foundation. Using the original ranking function, this search generated just 6 results with only such meeting notes. With the learned system, a search for “ndlf” now returns similar results to a search for “National Digital Library Foundation”. These results appear slightly more useful from the short abstracts that are presented. However, we discovered that in fact the search engine had not indexed the main NDLF Web page. We see here that the search system has recognized users running chains of queries looking for the NDLF website, although none have been successful in finding it. Despite this, some of the worst results for this query have indeed been pushed down the results list. The fifth feature is harder to interpret, but from log files it appears that users looking for the Department of Learning and Instruction saw this result and repeatedly skipped over it. This document used to appear as the top result given the query “instruction”.

4.10 Summary

This chapter has shown how to use information in query reformulations to obtain more useful implicit relevance judgments from clickthrough logs. A number of strategies for collecting such judgments were presented and shown to be valid by comparing to relevance judgments obtained by explicitly collecting judgments from experts. Further, the judgments collected from clickthrough data were used to learn an improved ranking function for a real document collection used by regular users. The results demonstrated the additional usefulness of using query chains over previous techniques for obtaining implicit relevance judgments.

CHAPTER 5

AVOIDING BIAS IN IMPLICIT FEEDBACK

In the previous chapter, we showed how to obtain useful implicit relevance judgments from clickthrough data. We saw that, in particular, query chains provide valuable judgments that identify relevant documents to queries that initially present only poor matches to users. We also saw how these relevance judgments can be used to learn an improved ranking function using Ranking SVMs. However, as described in Section 4.8, the relevance judgments used as training data mostly oppose the original ranking order. They were thus supplemented with background knowledge to avoid a degenerate solution, which simply learns to reverse the original rankings. In this chapter we will take a principled approach to study these bias effects, and show how they can be eliminated when learning to rank from implicit feedback. The research presented in this chapter was originally published in (Radlinski & Joachims, 2006)

5.1 Introduction

Learning to rank from implicit feedback allows the judgments of real users to be reflected in learned ranking functions. This contrasts with most previous work, which uses relevance judgments collected from human experts. We have seen in previous chapters how the clicking behavior of users can be translated into judgments about the relative relevance of individual documents. However, let us consider again the strategies proposed for generating relative relevance judgments in the previous chapter. They are shown once more in Figure 5.1.

<p>CLICK $>_q$ SKIP ABOVE</p>	<p>CLICK FIRST $>_q$ NO-CLICK SECOND</p>
<p>CLICK $>_{q'}$ SKIP ABOVE</p>	<p>CLICK FIRST $>_{q'}$ NO-CLICK SECOND</p>
<p>CLICK $>_{q'}$ SKIP EARLIER QUERY</p>	<p>CLICK $>_{q'}$ TOP TWO EARLIER QUERY</p>

Figure 5.1: Feedback strategies from Chapter 4. We either consider a single query, q , or a query q that has been preceded by a query q' . Given a query, a dot represents a result document and an x indicates the result was clicked on. A preference is generated for all arrows shown.

Four of these strategies, CLICK $>_q$ SKIP ABOVE, CLICK $>_{q'}$ SKIP ABOVE, CLICK $>_{q'}$ SKIP EARLIER QUERY and CLICK $>_{q'}$ TOP TWO EARLIER QUERY generate preferences that oppose the original ranking order. In other words, these strategies state that a lower ranked document is preferred over a higher ranked document. While the other two strategies do not share this property, it is relatively rare that users click only on the top ranked document and do not reformulate or click on any other documents. This means that the vast majority of preferences generated by these strategies oppose the original order. The same

observation can be made about the strategies studied by Joachims et al. (2005) and presented in Table 2.1 on page 34.

Importantly, whatever function was used to rank the documents, most preferences would be satisfied if the rankings presented were simply reversed. This reversal pressure is present because bias in user behavior does not allow us to validly obtain many relative relevance judgments that support the original ranking order. In this chapter, we will see that through experiment design it is possible to modify the presentation of search results to collect cleaner training data. In particular, we describe an algorithm called *FairPairs* that has a small effect on the quality of the results presented while provably providing training data that, under two reasonable assumptions, does not suffer from presentation bias. We will also verify the validity of the assumptions empirically using a real world search engine.

Additionally, we will see that learning data collected using the *FairPairs* algorithm guarantees that a learning algorithm that minimizes the number of misordered pairs will converge to an ideal ranking if one exists. Finally, using *FairPairs* it is also possible to measure the confidence that a particular pair of results is ranked in the correct order.

From a theoretical standpoint, many researchers have previously considered the question of stability and convergence when learning to rank, as the amount of training data grows (including Freund et al. (1998); Herbrich et al. (2000); Cohen et al. (1999); Crammer and Singer (2001); Chu and Keerthi (2005)). Most of this research has been for problems in ordinal regression, which considers the problem of learning to map results to a partial order and does not apply directly to more general ranking problems. Also, previous work does not consider how

user behavior biases the training data that can be collected. Instead, it assumes simple models of noise in training data.

Finally, we view FairPairs as experiment design (see, for example, Hinkelmann and Kempthorne (1994)). In traditional experiment design, the researcher asks the question of what to measure to ensure conclusive and unbiased results. In a Web-based search engine, we view the presentation of results to users as part of an interactive process that can also be designed to provide unbiased data for machine learning purposes. For this reason, we consider the data collection phase as part of the learning process.

5.2 Presentation Bias

We now describe the concept of presentation bias, which was first introduced on page 14 in Chapter 1. In normal Web search, users pay significantly more attention to results ranked highly than to those ranked lower. For example, the Osmot search engine described in the previous chapter was used to provide search functionality over the arXiv e-print archive, a large collection of academic articles¹⁰. We observed that users click on the fifth ranked result for only about 5% of queries, and click on lower ranked results even less often. However, if we take the fiftieth result and place it first or second in the ranking, users click on it more than 5% of the time. Does this indicate that we have a poor ranking function where the fiftieth result tends to be more relevant than the fifth? No. Rather, it demonstrates the concept of presentation bias in clickthrough data, even for search engines where users tend to be academic researchers.

¹⁰Available at <http://search.arxiv.org/>

Table 5.1: Results from a user study using the Google search engine presented by Joachims et al. (2005). In the “normal” condition, straight Google results were presented, while the top two results were swapped in the “swapped” condition. The counts show how often each result was clicked on when the Google’s top result was more or less relevant than the second result.

Relative relevance	“Normal”		“Swapped”	
	d_1	d_2	d_1	d_2
$rel(d_1) > rel(d_2)$	20/36	2/36	16/28	2/28
$rel(d_1) < rel(d_2)$	7/20	4/20	12/36	9/36

Similarly, Joachims et al. (2005) performed a controlled user study, described in Chapter 2, where volunteer subjects were asked to search for specific information using Google. The results viewed by the subjects were afterward assessed by expert judges for relevance. Table 5.1 shows a small selection of the results, where the subjects saw one of two experimental conditions. In the “normal” condition, the results were presented as ranked by Google. When the result presented at the top (d_1) was judged by a human expert to be more relevant than the result presented next (d_2), users clicked on the top result 20 out of 36 times and on the second result just twice, as could be expected. However, in the “swapped” condition the top two results from Google were reversed before being presented to users. Even when the second-ranked result (which was returned by Google as the top result) was more relevant, users still clicked predominantly on the result presented at the top. This again shows that presentation strongly influences user behavior.

Definition 5.1. Presentation bias *is manifested when users preferentially click on higher ranked results, irrespective of relevance.*

Presentation bias may occur for a number of reasons, such as users trusting the search engine to always present the most relevant result first. The question we now address is how to tease out information about the relevance of the search results from clickthrough logs despite such effects.

5.3 Bias-Free Feedback

In this section, we review the notion of relative relevance preferences and then present the FairPairs algorithm. Training data for learning to rank can be represented either as *absolute* or as *relative* relevance statements. The former involve data of the form $relevance(document_i | query) = r_i$ where r_i is an absolute measure of relevance. This approach requires an absolute relevance scale in the training data, for example specifying $r_i \in [0, 1]$. It is often particularly difficult to obtain well calibrated partial relevance judgments. For example, in ranking movies from 1 to 5 stars, different people may interpret a rating of 3 stars differently. Instead we consider relative statements, with training data in the form of preferences such as $relevance(document_i | query) > relevance(document_j | query)$. The aim is to obtain judgments where the probability some $document_i$ is judged more relevant than some $document_j$ is independent of the ranks at which they are presented.

We now present FairPairs by example, then provide the formal algorithm. The key idea is to randomize part of the presentation to eliminate the effect of presentation bias while making only minimal changes to the ranking. Consider some query that returns the documents $(d_1, d_2, d_3, d_4, d_5, d_6, \dots)$. We perturb the result set so that we can elicit relevance judgments unaffected by presentation

Algorithm 5.1 FairPairs

- 1: Let $R = (d_1, d_2, \dots, d_n)$ be the results for some query.
 - 2: Randomly choose $k \in \{0, 1\}$ with uniform probability.
 - 3: **if** $k = 0$ **then**
 - 4: **for** $i \in \{1, 3, 5, \dots\}$ **do**
 - 5: Swap d_i and d_{i+1} in R with 50% probability.
 - 6: **end for**
 - 7: **else**
 - 8: **for** $i \in \{2, 4, 6, \dots\}$ **do**
 - 9: Swap d_i and d_{i+1} in R with 50% probability.
 - 10: **end for**
 - 11: **end if**
 - 12: Present R to the user, recording clicks on results.
 - 13: Every time the lower result in a pair that was considered for flipping is clicked, record this as a preference for that result over the one above it.
-

bias. We first randomly pick $k \in \{0, 1\}$. If $k = 0$, we consider the result set as pairs $((d_1, d_2), (d_3, d_4), (d_5, d_6), \dots)$. Each pair of results is now independently flipped with 50% probability. For example, the final ranking might end up as $(d_1, d_2, d_4, d_3, d_5, d_6, \dots)$ with only d_3 and d_4 flipped. Alternatively, we could end up flipping all the pairs: each time FairPairs is executed, a different reordering may occur. Similarly, if $k = 1$, we do the same thing except consider the result set as pairs $(d_1, (d_2, d_3), (d_4, d_5), \dots)$. Then we take the result set generated in this way and present it to the user. In expectation half the results will be presented at their original position, and all results will be presented within one rank of their original position. The FairPairs algorithm is formally presented in Algorithm 5.1.

To interpret the clickthrough results of FairPairs, consider the results for some query q that returns (d_1, d_2, \dots, d_n) . Let $d_j \triangleleft d_i$ denote that d_j is presented just above d_i (that is, the user sees d_j first if they read from the top) and that k is such that d_i and d_j are in the same pair (for example, when $k = 0$, d_3 and d_4 are in the same pair, but d_2 and d_3 are not). Let n_{ij} count of how often this occurs. Also, let c_{ij} denote the number of times a user clicks on d_i when $d_j \triangleleft d_i$ (that is, when d_i is the bottom result in a pair). By perturbing the results with FairPairs, we have designed the experiment such that we can interpret c_{ij} as the number of votes for $\text{relevance}(d_i) > \text{relevance}(d_j)$, and c_{ji} as the number of votes for $\text{relevance}(d_j) > \text{relevance}(d_i)$. The votes are counted only if the results are presented in equivalent ways, providing an unbiased set of preferences because both sets of votes are affected by presentation bias in the same way. We formalize this property and prove its correctness in the next section. Note that if a user clicks multiple times on some set of results, they are making multiple votes.

Although in this chapter we focus on preferences generated from user clicks on the bottom result of a pair, in fact most of the properties discussed also appear to hold for preferences generated from clicks on the top result of a pair. The reason we chose to focus on clicks on bottom results is that, as described in Chapter 2, users typically read search engine results top to bottom and are less likely to look at the result immediately below one they click on than they are to look at one immediately above it.

5.4 Theoretical Analysis

In this section we will show that, given any presentation bias that satisfies two simple assumptions, FairPairs is guaranteed to give preference data that is unaffected by presentation bias.

We start by presenting our assumptions. Let $r_i(q)$ be the relevance of document d_i to a query q (we will usually omit q for brevity). The probability of a particular document being clicked by a user depends on its position in the search results, its relevance to the query, as well as potentially on every other document presented to the user. Assume the user selects d_{bot} from the list $(\mathbf{d}_\uparrow, d_{top}, d_{bot}, \mathbf{d}_\downarrow)$, where \mathbf{d}_\uparrow are the documents preceding (ranked above) d_{top} and \mathbf{d}_\downarrow are those after (ranked below) d_{bot} . In particular, d_{top} is the document just before d_{bot} . Let

$$P(d_{bot} | \mathbf{d}_\uparrow, (d_{top}, d_{bot}), \mathbf{d}_\downarrow) \quad (5.1)$$

be the probability that d_{bot} is clicked by the user given the list of choices.

Assumption 5.1 (Document Identity). *The probability of a user clicking depends only on the relevance of the documents presented, not their particular identity. Formally, we can write this as*

$$P(d_{bot} | \mathbf{d}_\uparrow, (d_{top}, d_{bot}), \mathbf{d}_\downarrow) = P(d_{bot} | \mathbf{r}_\uparrow, (r_{top}, r_{bot}), \mathbf{r}_\downarrow) \quad (5.2)$$

This assumption essentially states that the user is looking for any sufficiently relevant document. It also requires that users do not choose to skip over documents they recognize and know to be relevant. We come back to this later.

We now define two scores. First, the item relevance score measures how much more likely users are to click on more relevant results.

Definition 5.2 (Item Relevance Score). *If we take a ranking of documents and replace some document d_1 with a less relevant one d_2 while leaving all others unchanged, the difference between the probability of d_1 being selected and that of d_2 being selected is the item relevance score. Formally, if d_1 and d_2 have relevance r_1 and r_2 with $r_1 > r_2$, and $d_1, d_2 \notin \mathbf{d}_\uparrow \cup \mathbf{d}_\downarrow$, then*

$$\delta_{12}^{rel} = P(d_1 | \mathbf{r}_\uparrow, (r_{top}, r_1), \mathbf{r}_\downarrow) - P(d_2 | \mathbf{r}_\uparrow, (r_{top}, r_2), \mathbf{r}_\downarrow) \quad (5.3)$$

Analogously, consider the effect of replacing the document before the one that the user selects.

Definition 5.3 (Ignored Relevance Score). *If we take a ranking of documents and replace some document with a more relevant one while leaving all others unchanged, the difference between the probability of the user selecting the next document (after the one replaced) and the same probability without the change is the ignored relevance score. Formally, if d_1 and d_2 have relevance r_1 and r_2 where $r_1 > r_2$, and $d_1, d_2 \notin \mathbf{d}_\uparrow \cup \mathbf{d}_\downarrow$, then*

$$\delta_{12}^{ign} = P(d_{bot} | \mathbf{r}_\uparrow, (r_1, r_{bot}), \mathbf{r}_\downarrow) - P(d_{bot} | \mathbf{r}_\uparrow, (r_2, r_{bot}), \mathbf{r}_\downarrow) \quad (5.4)$$

This score measures how replacing the previous document changes the probability of a user clicking on a result. If δ_{12}^{ign} is negative, it means that replacing the previous document with a more relevant one reduces the probability of users clicking on the document under consideration. While we may expect this to be the case, it is possible that δ_{12}^{ign} is positive: if a user sees a very irrelevant document, they may be more likely to give up and not even consider the next result presented. On the other hand, if a user sees a somewhat relevant document, they may be more inclined to consider further results. We will measure this later.

Our second assumption relates to the relative magnitude of these two scores. Note that it would be trivially satisfied if the first is positive and the second negative.

Assumption 5.2 (Relevance Score Assumption). *For all $\mathbf{d}_\uparrow, \mathbf{d}_\downarrow, d_1, d_2$ with $d_1, d_2 \notin \mathbf{d}_\uparrow \cup \mathbf{d}_\downarrow$, the item relevance score is larger than the ignored relevance score.*

$$\delta_{ij}^{rel} > \delta_{ij}^{ign} \quad (5.5)$$

We will evaluate the validity of our assumptions in the experimental results section. Also, note that they are satisfied by many common item selection models (for example, users selecting results where they judge their probability of success above some threshold, as described by Miller and Remington (2004)).

We will now prove that the data collected using FairPairs is unaffected by presentation bias. Theorem 5.1 tells us that if the documents before and after a pair being considered vary independently of how the pair is ordered, observing that the expectation of the users' probability of selecting d_i when $d_j \triangleleft d_i$ is higher than the expectation of the users' probability of selecting d_j when $d_i \triangleleft d_j$ is both necessary and sufficient to deduce that $r_i > r_j$.

Theorem 5.1. *Let d_i and d_j be two documents with relevance r_i and r_j . If assumptions 5.1 and 5.2 are satisfied and $P(\mathbf{r}_\uparrow, \mathbf{r}_\downarrow | d_i \triangleleft d_j) = P(\mathbf{r}_\uparrow, \mathbf{r}_\downarrow | d_j \triangleleft d_i)$ then $r_i > r_j \Leftrightarrow P_{ij} > P_{ji}$, where $P_{ij} = E_{\mathbf{r}_\uparrow, \mathbf{r}_\downarrow} [P(d_i | \mathbf{r}_\uparrow, (r_j, r_i), \mathbf{r}_\downarrow)]$.*

Proof. We start by rewriting the expectations of the probabilities and simplifying:

$$P_{ij} = \sum_{\mathbf{r}_\uparrow, \mathbf{r}_\downarrow} P(d_i | \mathbf{r}_\uparrow, (r_j, r_i), \mathbf{r}_\downarrow) P(\mathbf{r}_\uparrow, \mathbf{r}_\downarrow | d_i \triangleleft d_j) \quad (5.6)$$

$$P_{ji} = \sum_{\mathbf{r}_\uparrow, \mathbf{r}_\downarrow} P(d_j | \mathbf{r}_\uparrow, (r_i, r_j), \mathbf{r}_\downarrow) P(\mathbf{r}_\uparrow, \mathbf{r}_\downarrow | d_i \triangleleft d_j) \quad (5.7)$$

$$P_{ij} - P_{ji} = \sum_{\mathbf{r}_\uparrow, \mathbf{r}_\downarrow} [P(d_i|\mathbf{r}_{ij}) - P(d_j|\mathbf{r}_{ji})] P(\mathbf{r}_\uparrow, \mathbf{r}_\downarrow | d_i \triangleleft d_j) \quad (5.8)$$

where $\mathbf{r}_{ij} = (\mathbf{r}_\uparrow, (r_j, r_i), \mathbf{r}_\downarrow)$. Say $P_{ij} - P_{ji}$ is positive. The sum can be positive if and only if the first term is positive for at least one \mathbf{r}_\uparrow and \mathbf{r}_\downarrow .

Applying Assumption 5.2, we see $r_i > r_j$ implies that

$$P(d_i|\mathbf{r}_\uparrow, (r_j, r_i), \mathbf{r}_\downarrow) = P(d_j|\mathbf{r}_\uparrow, (r_j, r_j), \mathbf{r}_\downarrow) + \delta_{ij}^{rel} \quad (5.9)$$

$$> P(d_j|\mathbf{r}_\uparrow, (r_j, r_j), \mathbf{r}_\downarrow) + \delta_{ij}^{ign} \quad (5.10)$$

$$= P(d_j|\mathbf{r}_\uparrow, (r_i, r_j), \mathbf{r}_\downarrow) \quad (5.11)$$

Moreover, if $r_j > r_i$ we get the reverse conclusion. By the law of the excluded middle we see that this is in fact an equivalence.

This equivalence means that if the first term in the summation is positive, then $r_i > r_j$. Hence $P(d_i|\mathbf{r}_\uparrow, (r_j, r_i), \mathbf{r}_\downarrow) > P(d_j|\mathbf{r}_\uparrow, (r_i, r_j), \mathbf{r}_\downarrow)$ for all \mathbf{r}_\uparrow and \mathbf{r}_\downarrow so the first term must always be positive. The same applies if the difference is negative. Hence the difference in expectations on the number of clicks always has the same sign as the difference in document relevance. \square

The theorem also tells us that we can collect relevance judgments about many pairs in the result set at the same time, by independently randomly reordering pairs, as is the case with FairPairs.

5.5 Practical Considerations

We now discuss how the data collected using FairPairs is affected by variations between search engines and in user behavior. This gives rise to practical issues

that should be kept in mind. The first effect to note is that prior to deciding whether to click, users only observe the abstracts presented by the search engine. A less relevant document presented with a misleadingly appealing abstract may generate more clicks than one that is more relevant but has a less appealing abstract. While addressed by Assumption 5.2, in practice this requires the search engine to generate snippets in an unbiased way, where the quality of a snippet does not vary differently for different documents depending on the types of queries entered by users. An alternative that may avoid this assumption could consider user dwell time on results in addition to clicks. Different search engines may also have users who are more or less prepared to click on results ranked highly irrespective of the abstract. However this is not a concern as both documents within a pair are always ranked highly equally often and hence benefit from this trust equally. Other presentation effects, such as a bias against users clicking on documents that are not visible unless the user scrolls also do not introduce bias into the training data, as confirmed by the results presented later in this chapter.

Another issue to consider is that of user familiarity with results: documents that are known to be relevant by the user but not clicked on may collect fewer votes than would be expected. However, it has been established that users often revisit Web pages, suggesting that this is not a concern (McKenzie & Cockburn, 2001). Nevertheless, on specific collections for specific user groups this may be a limitation. Similarly, if the relevance of documents evolves over time, data collected may become out of date, although this is true for any data collection method. Finally, one well known user behavior that FairPairs does not exploit is that of query reformulation, explored in the previous chapter. FairPairs does not

allow preferences to be generated in a fair way between documents returned by sequential queries, although extending it in this way is an interesting challenge.

5.6 Learning Convergence

In this section, we consider the convergence properties of a learning algorithm that minimizes the error rate trained on data collected with FairPairs. For simplicity, assume that no two documents have the same relevance to a query.

Theorem 5.2. *Let n_{ij} be the number of times the user saw $d_j \triangleleft d_i$ and c_{ij} be the number of times a user clicked on d_i in this situation. Let $\epsilon = \frac{1}{2} \min_{i,j} |P_{ij} - P_{ji}|$.*

Assume we have collected enough data using FairPairs such that $\forall d_i, d_j, |1 - n_{ji}/n_{ij}| < \epsilon$ and $|p_{ij} - P_{ij}| < \frac{1}{2}\epsilon$, where $p_{ij} = c_{ij}/n_{ij}$. Moreover, assume there exists a ranking function f^ that ranks the documents perfectly in terms of decreasing relevance. Then, a learning algorithm that minimizes error rate will return f^**

Proof. Assume for the purpose of a contradiction that the learning algorithm learns a ranking function $f \neq f^*$ that has a lower error rate on the training data. Since the rankings differ, there must be at least one pair of documents (d_i, d_j) where the rankings disagree. Assign d_i to be such that $r_i > r_j$, which is equivalent to d_i being returned higher than d_j by the ranking function f^* .

The number of violated constraints involving d_i and d_j for the ranking function f^* is

$$\begin{aligned}
err_{ij}^* &= n_{ij}p_{ij}\mathbf{1}[\text{rank}^*(d_i) > \text{rank}^*(d_j)] \\
&+ n_{ji}p_{ji}\mathbf{1}[\text{rank}^*(d_j) > \text{rank}^*(d_i)]
\end{aligned} \tag{5.12}$$

$$= n_{ji}p_{ji}, \tag{5.13}$$

where $\mathbf{1}$ is the indicator function and $\text{rank}^*(d_i)$ is the rank at which d_i is returned by f^* . The number of violated constraints involving d_i and d_j for the ranking function f is:

$$\begin{aligned}
err_{ij}^f &= n_{ij}p_{ij}\mathbf{1}[\text{rank}^f(d_i) > \text{rank}^f(d_j)] \\
&+ n_{ji}p_{ji}\mathbf{1}[\text{rank}^f(d_j) > \text{rank}^f(d_i)]
\end{aligned} \tag{5.14}$$

$$= n_{ij}p_{ij}, \tag{5.15}$$

since we know that rank^f disagrees with rank^* on the order of d_i and d_j . By assumption, we know that

$$(1 + \epsilon)n_{ij} > n_{ji} > (1 - \epsilon)n_{ij} \tag{5.16}$$

Next, by Theorem 5.1, $P_{ij} > P_{ji}$ since $r_i > r_j$. The definition of ϵ implies $P_{ij} - P_{ji} \geq 2\epsilon$. Since we know that $|p_{ij} - P_{ij}| < \frac{1}{2}\epsilon$ and similarly for p_{ji} , we get $p_{ij} - p_{ji} > \epsilon$.

Pulling this all together,

$$err_{ij}^f - err_{ij}^* = n_{ij}p_{ij} - n_{ji}p_{ji} \tag{5.17}$$

$$> n_{ij}p_{ij} - n_{ij}(1 + \epsilon)p_{ji} \tag{5.18}$$

$$= n_{ij}[(p_{ij} - p_{ji}) - \epsilon p_{ji}] \tag{5.19}$$

$$> n_{ij}[\epsilon - \epsilon p_{ji}] \geq 0 \tag{5.20}$$

Since the difference in the number of violated constraints is zero for pairs where f and f^* agree, and positive for all others, the error rate of the learned function f must be higher than that of f^* , meaning we have a contradiction. \square

Note that the data collected using the techniques proposed by Joachims (2002), described in Chapter 2, and the extensions to query chains described in Chapter 4, does not have this property of eventual convergence because it tends to learn to reverse any presented ranking. In fact, notice that if for any presented ranking preferences collected always oppose the ranking order, any algorithm that minimizes the number of violated preferences is guaranteed to eventually reverse the ranking given enough training data.

To ensure eventual convergence, we now need to ensure that sufficient data is collected about every pair of documents so that $\forall d_i, d_j, |1 - n_{ji}/n_{ij}| < \epsilon$ and $|p_{ij} - P_{ij}| < \frac{1}{2}\epsilon$. The first condition is eventually satisfied since each flip is performed with 50% probability. The second becomes satisfied after a pair is observed sufficiently often, because the probability of observing a click approaches its expectation by the law of large numbers. One strategy to obtain sufficient data would be for the search engine to occasionally insert random documents into the result set, and to assume users have a non-zero probability of viewing results at any rank. While this strategy would work, there are probably much more efficient exploration strategies, providing an interesting area for future research. Additionally, although the theorem is stated in terms of observing the relative relevance of all pairs of documents, if we assume relevance is transitive then the number of observations necessary may be substantially reduced.

5.7 Experimental Results

We now evaluate the validity of the assumptions presented and measure the click probabilities of data collected with FairPairs. For these experiments we used

Osmot, the search engine described in the previous chapter, on the arXiv e-print collection. This collection consists of about 350,000 academic articles. Osmot was modified to perform FairPairs on the search results before presenting them to real users who were unaware of the experiment. We measured the probability of users clicking on both the top and bottom results of each pair. We recorded user clicks for about three months, and counted how often each pair of ranks was presented and how often each result was clicked on. During our experiments, we observed 44,399 queries coming from 13,304 distinct IP addresses. We recorded 48,976 clicks on results, often with many clicks for the same query.

Because we did not have expert relevance judgments, and collecting them was impractical, we hypothesized that on average the fiftieth ranked result returned by the search system is less relevant to the query than the top few results. To check if we could confirm this, after FairPairs was performed on the results of a query, Osmot randomly swapped result fifty and one of the top eight results whenever there were more than fifty results for a query. This modified result set was then displayed to users.

5.7.1 Item Relevance Score

The left side of Figure 5.2 shows how often users clicked on the bottom result of a pair. Four types of pairs were observed. Pairs of the form 1-2, 2-3 involve two adjacent results from the original ranking function in their original order (1-2 indicates it was the original first and second results, in the original order). Pairs of the form 2-1, 3-2 involve two originally adjacent results in reverse order. Due to the fiftieth result being randomly inserted, we also have pairs of the form

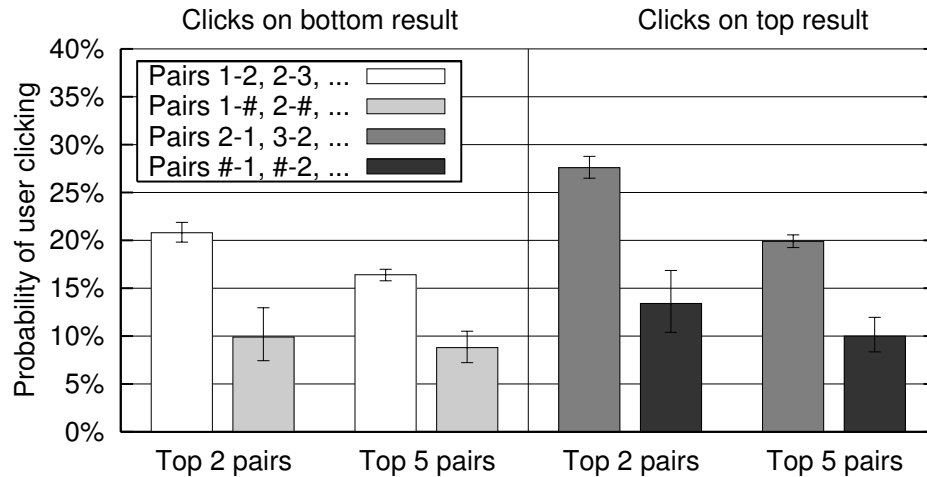


Figure 5.2: Click probability measurements of the Item Relevance Score.

1-# (indicating the first result followed by the 50th result) and #-1 (indicating the same pair reversed). We summed up the counts for all pairs in these four groups, either for the top two pairs presented (for example, for 1-2 and 2-3) or for the top five pairs (for example, for 1-2 through 5-6) counting over all queries where a user clicked on at least one result. In the figure, we see that if the lower result in a pair is result 50 (postulated to be less relevant than those in the top six), the probability of the user clicking on that lower result is smaller than if the lower result was from the original top six. The error bars indicate 95% binomial confidence intervals, showing the differences to be statistically significant. This shows that the Item Relevance Score is positive and gives an idea of its average magnitude at different ranks for this dataset. In addition, the right side of Figure 5.2 shows that keeping the lower result fixed, a similar score could be defined for the change in click probability on the top result as it is more or less relevant.

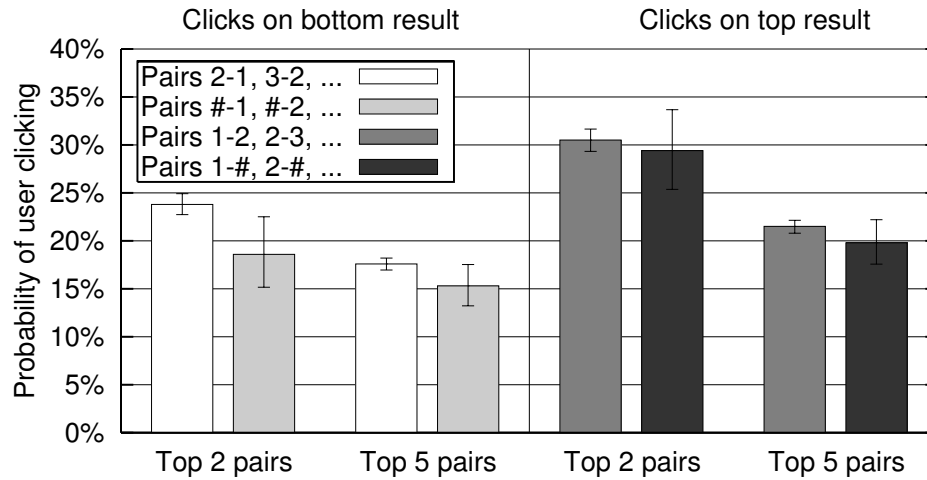


Figure 5.3: Click probability measurements of the Ignored Relevance Score.

5.7.2 Ignored Relevance Score

The ignored relevance score measures the change in click probability as the result *before* the one clicked on varies. We see in the left side of Figure 5.3 that if the result before the one selected is more relevant, the next document is slightly more likely to be clicked on. We attribute this to result 50 tending to be much less relevant, making users more likely to stop considering results once they encounter it. This means that in our experiments, δ_{ij}^{ign} tends to be positive. However, the magnitude of the decrease in click probability is much smaller than that seen in Figure 5.2, thus the Relevance Score Assumption holds. Additionally, we observe that this score quickly decreases for lower results, unlike the item relevance score. Also, this is consistent with the right hand side of Figure 5.2 – there we saw the probability of the user clicking on the top result, whereas here we are evaluating the probability of the user clicking on the bottom result. Together, these figures show that placing a less relevant document as the top result in a pair makes both results in the pair less likely to be clicked on.

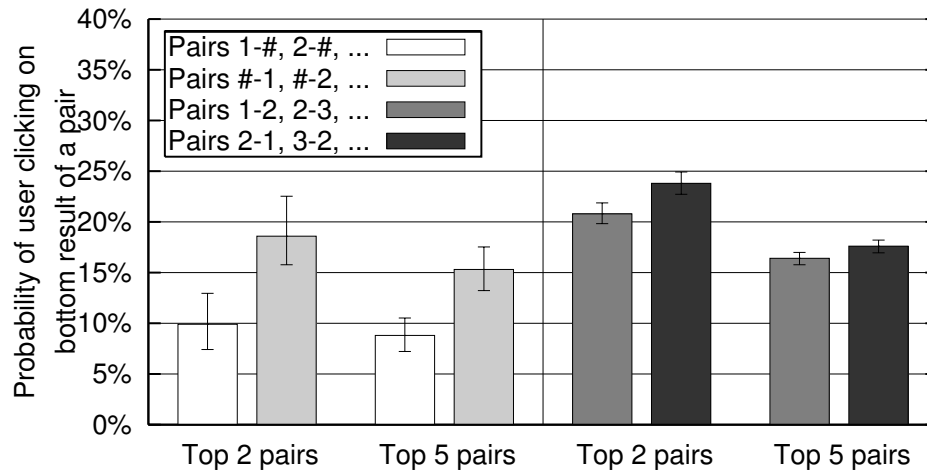


Figure 5.4: Evaluation of the relative relevance of search results returned by the arXiv search engine.

5.7.3 FairPairs Preference Test

Next, to confirm the correctness of data generated with FairPairs directly, consider the difference between the bottom click probabilities when two results are swapped. The left side of Figure 5.4 shows that reversing a top-five result and the 50th result within a pair behaves as the theory tells us it should. We see that when the fiftieth result is at the bottom of a pair, it is significantly less likely to be clicked on than when an original top-five result is at the bottom of the pair. On the right side of the figure, we see the click probability on the bottom result for pairs of the form 1-2 and for pairs of the form 2-1. In fact, summing the counts for the top 2 pairs (1-2 and 2-3), the difference in click probability is statistically significant. This shows that on average the top three results returned by the search engine are ranked in the correct order.

We also evaluated our approach in a situation where we have the true relative relevance of documents as assessed by human judges. Using the results of the eye tracking study by Joachims et al. (2005) and described in Chapter 2, we computed

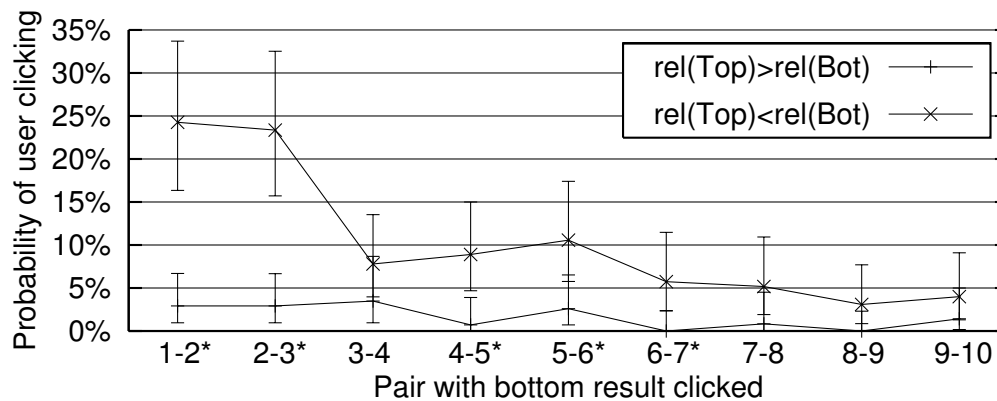


Figure 5.5: Probability of user clicking only on the bottom result of a pair as a function of the pair. The two curves are for when the document immediately above the document clicked was judged strictly more relevant or strictly less relevant by expert human judges. * indicates the difference is statistically significant with 95% confidence using a Fisher Exact test.

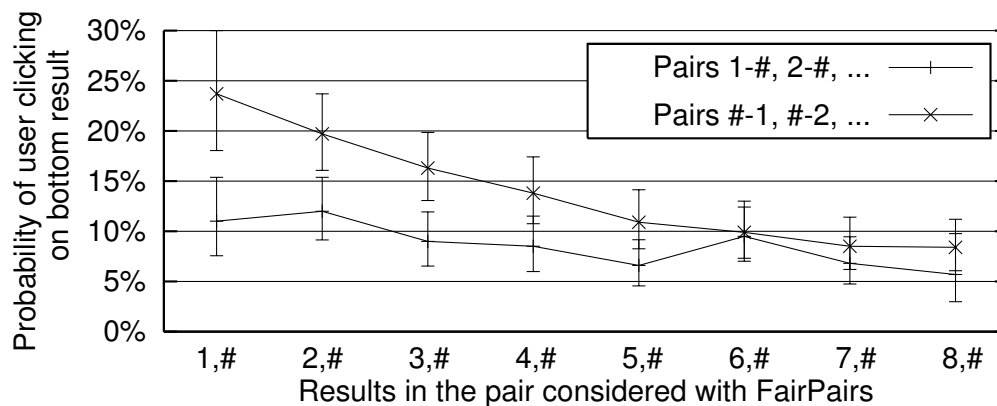


Figure 5.6: Probability of user clicking only on the bottom result of a pair as a function of the pair for all queries generating at least one user click. The two curves are the cases where the document that was originally ranked fiftieth is the top or the bottom document in the pair. The error bars indicate 95% confidence intervals.

the probability of a participant in the user study clicking on the bottom result of a pair of results when the top result was judged strictly more relevant or strictly less relevant by expert human judges. Figure 5.5 shows that although FairPairs was not performed on the results in the study, the data supports the FairPairs premise that the probability of a user clicking on a document d_i at rank i is higher if $rel(d_{i-1}) < rel(d_i)$ than if $rel(d_{i-1}) > rel(d_i)$.

Figure 5.6 shows the equivalent curve for the arXiv search engine, in effect providing a more detailed view of Figure 5.4. We again considered all queries that generated at least one click and exploited symmetries in our experiment design to obtain the maximal amount of data for this figure. It shows that if the fiftieth ranked document is displayed in a pair with a top-eight document, the FairPairs data collected is in agreement with our hypothesis that the fiftieth ranked document is less relevant than any from the top eight. In particular, the first five differences in click probabilities are statistically significant. For lower ranks the curves appear to proceed in a similar manner. This includes result pairs below the sixth, which are usually are not visible without users scrolling.

5.8 Summary

In this chapter we introduced FairPairs, a method to modify the presentation of search engine results with the purpose of collecting more reliable relevance feedback from normal user behavior. We showed that under reasonable assumptions the data gathered is provably unaffected by presentation bias. We also showed that given sufficient clickthrough data, training data generated with FairPairs will allow a learning algorithm to converge to the ideal ranking. We

performed real world experiments that evaluated the assumptions and conclusions in practice. Given bias-free training data generated in this way, it is possible to use existing methods for learning to rank without additional modifications to compensate for presentation bias being necessary.

CHAPTER 6

ACTIVE METHODS FOR OPTIMIZING DATA COLLECTION

The analysis in this thesis has, thus far, assumed that clickthrough data is collected passively, or at best with minimal intervention as in the previous chapter. In effect, we simply infer relevance judgments from recorded interactions that take place anyway. We now describe techniques to guide users, in order to combat evaluation bias and provide more useful training data for a learning search engine. This research was originally published in (Radlinski & Joachims, 2007).

6.1 Introduction

When learning to rank, we have seen that two alternatives for obtaining training data are expert relevance judgments or relevance judgments collected implicitly by observing user behavior. Assuming that we wish to avoid the difficulties associated with collecting judgments from experts, as described in Chapter 1, consider once more the properties of user behavior described in Chapter 2.

We saw that users usually execute a query, and then perhaps consider the first two or three results presented by the search engine (Granka et al, 2004). The feedback (clicks) on these results can be recorded and used to infer relevance judgments. These judgments can then be used to train a learning algorithm such as a Ranking Support Vector Machine, as described in Chapter 4. In particular, the eye tracking study showed that users very rarely even look at results beyond the first few. Similarly, other researchers have previously noticed that users click

predominantly on search results at high ranks (for example, see Agichtein et al. (2006)).

Hence clickthrough data is strongly biased toward documents already ranked highly. Highly relevant results that are not initially ranked highly for any query may never be observed and evaluated. This means that if the ranking function used by a search engine initially performs poorly for some class of queries, training examples that identify truly relevant results for these queries may never be observed. This would make it difficult for a learned ranking to ever converge to an optimal ranking.

To avoid this evaluation bias in which documents are evaluated, this chapter presents a new formulation for learning to rank, where the ranking presented to users is optimized to obtain useful data rather than strictly in terms of estimated document relevance. The goal this formulation addresses is to minimize the total loss from presenting poor rankings over *all* time.

There are many approaches by which more useful training data could potentially be collected. For example, one possibility would be to intentionally present unevaluated documents in the top few positions of search engine results, aiming to collect more feedback on them. However, such an ad-hoc approach is unlikely to be useful in the long run, and would hurt user satisfaction substantially in the short run by often presenting suboptimal results. We instead introduce principled modifications that can be made to the rankings presented. These changes, which do not substantially reduce the quality of the ranking shown to users, produce much more informative training data and quickly lead to higher quality rankings being shown to users. In contrast with previous work by Chu and Ghahramani (2005a), we do not simply ask which relevance

judgments should be obtained to reduce uncertainty in establishing which is the correct ranking. Rather, we consider how to obtain training data that will quickly improve the quality of rankings using metrics suitable for measuring search engine performance.

We will now formalize the learning problem as an optimization task, present a suitable Bayesian probabilistic model and discuss inference and learning. Following this, we present strategies to modify the rankings shown to users so that performance of learned rankings improves rapidly over time. An evaluation of this approach is then presented, using both synthetic data and TREC-10 Web data. In particular, we see the improvements using our exploration strategies are much faster than with passive or random data collection.

6.2 Formalizing the Learning Problem

Assume we have a document corpus $\mathcal{C} = \{d_1, \dots, d_{|\mathcal{C}|}\}$ and some fixed user query q . For this query, we want to estimate the relevance $\mu_i^* \in \mathfrak{R}$ of each document d_i . From earlier chapters, we know that users can provide us with noisy judgments of the form $\mu_i^* > \mu_j^*$. We assume that some ranking function can provide initial estimates of μ_i^* . The goal is accurately estimate μ_i^* with as little training data as possible.

The estimation task involves a three step iterative process: First, given relevance estimates, we must select a ranking to display to users. Second, given the ranking displayed, users provide relevance feedback. Third, using the relevance feedback, we update the relevance estimates and repeat the process for the next

user. Our focus in this chapter will be mostly on the first step, namely selecting rankings of documents to show users so that the collected judgments allow the relevance estimates to be improved quickly, while at the same time maximizing the quality of the rankings.

6.2.1 Probabilistic Model

Let $M^* = (\mu_1^*, \dots, \mu_{|C|}^*) \in \mathcal{M}$ be the true relevance values of the documents in \mathcal{C} . Modeling the problem of finding M^* given training data \mathcal{D} in a Bayesian framework, we want to maintain our knowledge about M^* in the distribution

$$P(M|\mathcal{D}) = \frac{P(\mathcal{D}|M)P(M)}{P(\mathcal{D})} \quad (6.1)$$

We assume that $P(M|\mathcal{D})$ is multivariate normal with zero covariance:

$$P(M|\mathcal{D}) = \mathcal{N}(\nu_1, \dots, \nu_{|C|}; \sigma_1^2, \dots, \sigma_{|C|}^2) \quad (6.2)$$

Graphically, we can draw $P(M|\mathcal{D})$ as a set of Gaussians centered at ν_i (our current estimate of document relevance) with variance σ_i^2 (our current uncertainty). This is illustrated in Figure 6.1.

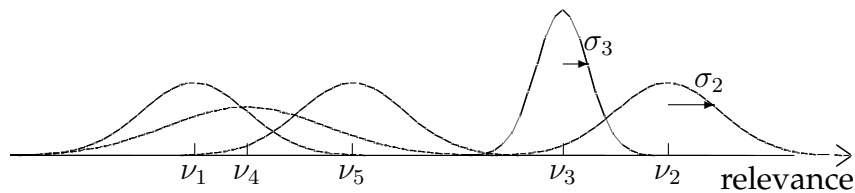


Figure 6.1: Example of how we can maintain estimates of the relevance of documents to a fixed query.

This model is motivated by ability estimates maintained for chess players given the outcomes of chess games, as described by Glickman (1999). In the most closely related previous work, Chu and Ghahramani (2005a) address a similar problem using Gaussian Processes . However, instead maintaining the distribution $P(M|\mathcal{D})$, they directly estimate M^* given \mathcal{D} . This is also true of other related prior work, for instance by Dekel et al. (2003); Fuhr (1989); Lin et al. (2006). The key difference in our approach is that we are not simply finding the optimizing ranking. Rather, maintaining $P(M|\mathcal{D})$ is key as it allows us to optimize for collected training data.

6.2.2 Inference

We measure the difference between relevance assignments using a loss function $\mathcal{L} : \mathcal{M} \times \mathcal{M} \rightarrow \mathfrak{R}$. To find good relevance estimates, we want to find an $M = (\mu_1, \dots, \mu_{|\mathcal{C}|}) \in \mathcal{M}$ such that $\mathcal{L}(M, M^*)$ is small. Noting that M^* is unknown, we want to find the ranking that minimizes the expected loss given what we know about M^* , namely $P(M|\mathcal{D})$:

$$\operatorname{argmin}_M E_{M^* \sim P(M|\mathcal{D})} [\mathcal{L}(M, M^*)] \quad (6.3)$$

where M^* is drawn from the probability distribution $P(M|\mathcal{D})$.

Suppose the loss function \mathcal{L} can be decomposed over pairs of documents in \mathcal{C} . We can then decompose the expected loss into a form easier to work with:

$$E_{P(M^*|\mathcal{D})} [\mathcal{L}(M, M^*)] = E_{P(M^*|\mathcal{D})} \left[\sum_{i=1}^{|\mathcal{C}|} \sum_{j=i+1}^{|\mathcal{C}|} \mathcal{L}^{pair}(M, M^*, i, j) \right] \quad (6.4)$$

$$= \sum_{i=1}^{|\mathcal{C}|} \sum_{j=i+1}^{|\mathcal{C}|} E_{P(M^*|\mathcal{D})} [\mathcal{L}^{pair}(M, M^*, i, j)] \quad (6.5)$$

where $P(M^*|\mathcal{D})$ is shorthand for $M^* \sim P(M|\mathcal{D})$.

We will now show that the mode of $P(M|\mathcal{D})$, namely $\hat{M} = (\nu_1, \dots, \nu_{|C|})$, is often the solution to Equation 6.3. Consider solving Equation 6.3 for a loss function that counts the number of misordered pairs of documents. The assignment with minimum expected loss is the mode of $P(M|\mathcal{D})$.

Lemma 6.1. *Let $P(M|\mathcal{D}) = \mathcal{N}(\nu_1, \dots, \nu_{|C|}; \sigma_1, \dots, \sigma_{|C|})$ be a distribution over models. Assume $\mathcal{L}(M, M^*)$ counts the number of differently ordered pairs of documents, when they are sorted by μ_i and μ_i^* respectively. A solution of*

$$\operatorname{argmin}_M E_{M^* \sim P(M|\mathcal{D})} [\mathcal{L}(M, M^*)]$$

is $\hat{M} = (\nu_1, \dots, \nu_{|C|})$.

Proof. Assume $M^{opt} = (\mu_1^{opt}, \dots, \mu_{|C|}^{opt})$ is the minimizing relevance assignment, and has lower loss than \hat{M} . There must exist two documents d_i and d_j that are ranked adjacently when documents are ordered by M^{opt} yet are ordered differently by \hat{M} , i.e. $\mu_i^{opt} > \mu_j^{opt}$ and $\nu_i < \nu_j$. Let M^{flip} be the ranking obtained by reversing d_i and d_j . Let these rankings have expected loss E^{flip} and E^{opt} .

As the documents are adjacent, the loss of M^{opt} and M^{flip} only differs in the contribution of the pair (d_i, d_j) . Plugging in the loss function we get

$$E^{flip} - E^{opt} = P(\mu_i^* > \mu_j^*) - P(\mu_j^* > \mu_i^*) \quad (6.6)$$

$$= \Phi\left(\frac{\nu_i - \nu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}}\right) - \Phi\left(\frac{\nu_j - \nu_i}{\sqrt{\sigma_i^2 + \sigma_j^2}}\right) < 0 \quad (6.7)$$

where Φ is the cumulative distribution function of the standard normal distribution, since $\nu_i < \nu_j$. Hence we have a contradiction as M^{opt} is not the minimizing ranking. \square

We see a similar result for the loss function that penalizes any error in the difference of document relevances, $\mathcal{L}^{pair}(M, M^*, i, j) = ((\mu_i - \mu_j) - (\mu_i^* - \mu_j^*))^2$.

Lemma 6.2. *Let $P(M|\mathcal{D}) = \mathcal{N}(\nu_1, \dots, \nu_{|C|}; \sigma_1, \dots, \sigma_{|C|})$ be a distribution over models.*

Assume $\mathcal{L}^{pair}(M, M^, i, j) = ((\mu_i - \mu_j) - (\mu_i^* - \mu_j^*))^2$. A solution of*

$$\operatorname{argmin}_M E_{M^* \sim P(M|\mathcal{D})} [\mathcal{L}(M, M^*)]$$

is $\hat{M} = (\nu_1, \dots, \nu_{|C|})$.

Proof. Let M^{opt} be the minimizing model. Let $\delta_{ij}^{opt} = \mu_i^{opt} - \mu_j^{opt}$ be the difference in relevance estimates of d_i and d_j according to M^{opt} , and $\hat{\delta}_{ij}$ and δ_{ij}^* be defined equivalently for \hat{M} and M^* respectively. Let $\sigma_{ij} = (\sigma_i^2 + \sigma_j^2)^{1/2}$. The contribution to the expected loss for the pair of documents (d_i, d_j) is

$$\begin{aligned} & E_{P(M^*|\mathcal{D})}[\mathcal{L}^{pair}(M, M^*, i, j)] \\ &= \frac{1}{\sigma_{ij}\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left(-\frac{(\delta_{ij}^* - \hat{\delta}_{ij})^2}{2\hat{\sigma}_{ij}^2}\right) (\delta_{ij}^* - \delta_{ij}^{opt})^2 d\delta_{ij}^* \end{aligned} \quad (6.8)$$

$$= \sigma_{ij}^2 + (\hat{\delta}_{ij} - \delta_{ij}^{opt})^2 \quad (6.9)$$

which is minimized if $\delta_{ij}^{opt} = \hat{\delta}_{ij}$. Hence $M^{opt} = \hat{M}$ minimizes all terms in the sum in Equation 6.5 simultaneously and thus minimizes the expected loss. \square

We see that the mode of the distribution $P(M|\mathcal{D})$ minimizes the expected loss for two reasonable loss functions. As it can also be obtained very efficiently given $P(M|\mathcal{D})$, for the remainder of this chapter we will assume that the mode is, or is close to, the minimizer of the expected loss. We will refer to the ranking obtained by sorting documents by their relevance according to the mode of $P(M|\mathcal{D})$ as the *mode ranking*.

6.2.3 Loss Function

Given our understanding of real user behavior from Chapter 2, we see that the loss functions discussed above are too simple. Specifically, two properties to expect of an appropriate loss function are (1) The loss for ranking a less relevant document above a more relevant document should be larger if the documents are presented higher in the ranking (where users are more likely to observe them); (2) The loss should be larger if the difference in relevance is larger. As there is no commonly used pairwise decomposable loss function with these properties, we propose a quadratic hinge-loss function with cost of misordering decaying exponentially with rank:

$$\mathcal{L}^{pair}(M, M^*, i, j) = e^{-r_{ij}} \left((\mu_i - \mu_j) - (\mu_i^* - \mu_j^*) \right)^2 \mathbf{1}_{misordered} \quad (6.10)$$

With r_{ij} we denote the minimum rank of d_i or d_j when all documents are ordered by M (which are the relevance assignments used to present results to users) divided by 10, and $\mathbf{1}$ is the indicator function. A pair of documents is considered misordered if the relative ranking according to M does not agree with that according to M^* . Making use of the pairwise form of the loss function and plugging in the mode ranking \hat{M} , the inner term of Equation 6.5 can now be written as

$$\begin{aligned} & E_{P(M^*|\mathcal{D})}[\mathcal{L}^{pair}(\hat{M}, M^*, i, j)] \\ &= \int P(\mu_i^* | \nu_i, \sigma_i) \int P(\mu_j^* | \nu_j, \sigma_j) \mathcal{L}^{pair}(\hat{M}, M^*, i, j) d\mu_i^* d\mu_j^* \\ &= \frac{1}{\sqrt{2\pi}\sigma_{ij}} \int_{-\infty}^{\infty} \exp\left(-\frac{(\delta_{ij}^* - \hat{\delta}_{ij})^2}{2\sigma_{ij}^2}\right) \mathcal{L}^{pair}(\delta_{ij}^*, \hat{\delta}_{ij}, r_{ij}) d\delta_{ij}^* \end{aligned} \quad (6.11)$$

where $\delta_{ij}^* = \mu_i^* - \mu_j^*$, $\hat{\delta}_{ij} = \nu_i - \nu_j$ and $\sigma_{ij}^2 = \sigma_i^2 + \sigma_j^2$, noting that the difference of two normally distributed variables is also normally distributed. Plugging in

the loss, and choosing to sum over the pairs such that $\hat{\delta}_{ij}$ is always negative, Equation 6.11 becomes:

$$\begin{aligned}
& E_{P(M^*|\mathcal{D})}[\mathcal{L}^{pair}(\hat{M}, M^*, i, j)] \\
&= \frac{e^{-r_{ij}}}{\sqrt{2\pi}\sigma_{ij}} \int_0^\infty (\delta_{ij}^* - \hat{\delta}_{ij})^2 \exp\left(-\frac{(\delta_{ij}^* - \hat{\delta}_{ij})^2}{2\sigma_{ij}^2}\right) d\delta_{ij}^* \\
&= e^{-r_{ij}} \left[\frac{\sigma_{ij}^2}{2} \left(1 + \operatorname{erf}\left(\frac{\hat{\delta}_{ij}}{\sqrt{2}\sigma_{ij}}\right)\right) - \frac{\hat{\delta}_{ij}\sigma_{ij}}{\sqrt{2\pi}} \exp\left(\frac{-\hat{\delta}_{ij}^2}{2\sigma_{ij}^2}\right) \right] \quad (6.12)
\end{aligned}$$

where $\operatorname{erf}()$ is the error function. Substituting this into Equation 6.5 gives an easy to compute closed form expression for the expected loss.

6.2.4 Estimating the Model Parameters

We now need a method to maintain $P(M|\mathcal{D})$ as we collect relevance judgments.

First we note that, as shown in Chapter 2, clickthrough data is best interpreted as relative relevance judgments. However, clicking behavior also tends to be noisy. Therefore, we model the probability of obtaining individual pairwise comparisons using the Bradley-Terry model (Bradley & Terry, 1952), which as described on page 55 in Chapter 3 is a standard approach for modeling noise in pairwise comparisons. This model assumes that the probability of document d_i being preferred to document d_j can be written as:

$$P(d_i \succ_q d_j) = \frac{\operatorname{rel}(d_i, q)}{\operatorname{rel}(d_i, q) + \operatorname{rel}(d_j, q)}, \quad (6.13)$$

If we also assume that the document relevances to a query are distributed according to a Gaussian prior, we can use the Glicko chess rating system (Glickman, 1999) to estimate the relevance of each document and the uncertainty in the

$$\nu_i \leftarrow \nu_i + \frac{q}{\frac{1}{\sigma_i^2} + \frac{1}{\delta^2}} g(\sigma_j^2) (s_i - E(s|\nu_i, \nu_j, \sigma_j^2)) \quad (6.14)$$

$$\sigma_i^2 \leftarrow \left(\frac{1}{\sigma_i^2} + \frac{1}{\delta^2} \right)^{-1} \quad (6.15)$$

where

$$q = \frac{\log 10}{400}$$

$$g(\sigma^2) = \frac{1}{\sqrt{1 + 3q^2\sigma^2/\pi^2}}$$

$$E(s|\nu_i, \nu_j, \sigma_j^2) = \frac{1}{1 + 10^{-g(\sigma_j^2)(\nu_i - \nu_j)/400}}$$

$$\delta^2 = \frac{1}{q^2 g(\sigma_j^2)^2} \times \frac{1}{E(s|\nu_i, \nu_j, \sigma_j^2)(1 - E(s|\nu_i, \nu_j, \sigma_j^2))}$$

Figure 6.2: The Glicko update equations, which describe how the estimated relevance ν_i and estimated variance σ_i^2 for document d_i should be updated following a comparison to document d_j . s_i is 1 if d_i was judged more relevant than d_j , and 0 otherwise.

relevance. In particular, this algorithm provides a set of closed form approximate update equations to update our beliefs about document relevance following pairwise comparisons. This approach is particularly appealing as it provides simple to compute online updates as data is collected. The update equations are shown in Figure 6.2. While it would also be interesting to compare alternative ways of maintaining $P(M|\mathcal{D})$ (for example, TrueSkill, developed by Herbrich and Graepel (2006)), or using a batch algorithm (for example, Hunter (2004) discusses a number of alternatives), the simplicity and online aspects of the Glicko system are appealing. In particular, in real world settings where large amounts of data are collected for large document collections with a large number of queries, a global optimization is likely to be slow and thus infeasible.

6.3 Exploration Strategies

We turn to the question of optimizing the data collection process to most quickly minimize the loss. As we have seen, users are much more likely to provide feedback on highly ranked documents. By selecting which documents to present at high rank, we influence the pairs of documents for which we obtain relevance judgments. Here, we consider modifications that change two documents in a ranking, limiting ourselves to the top two most of the time. We will see that despite the simplicity of this approach, substantial improvements in performance can be obtained at small cost in presented ranking quality.

We will consider the following five strategies for determining which ranking to present users, the first two being baselines for comparing against.

Strategy 6.1. Passive Collection (TOP2) *Present the mode ranking, sorting documents by $\hat{M} = (\nu_1, \dots, \nu_{|c|})$.*

The strategy TOP2 assumes no changes are made to the mode ranking, ignoring evaluation bias in data collection. This is the approach used in most previous work in learning to rank, and would be effective if users provided feedback about results throughout the ranking. In some settings this may be the case, for example in search engines for academic articles where many users thoroughly consider all retrieved results. However in general Web search settings, as discussed above, users focus their attention on the highest ranked results.

Strategy 6.2. Random Exploration (RANDOM) *Select a random pair of documents and present them first and second. Then rank the remaining documents according to \hat{M} .*

This strategy is a naïve modification of the mode ranking. Two documents are picked uniformly at random and inserted at the top of the ranking presented to users. Given the uniform distribution, this perturbation is likely to often pick documents that have a low prior expectation of being relevant, thus likely presents users with poorer results. However, it benefits from the potential for feedback on all documents regardless of rank, even in the presence of significant evaluation bias. A similar method was proposed by Pandey et al. (2005) in the context of identifying new Web pages that would soon become popular, suggesting to randomly insert new documents into Web search results.

Strategy 6.3. Largest Expected Loss Pair (LELPAIR) *Select the pair of documents d_i and d_j that have the largest pairwise expected loss contribution, and present these first and second. Rank the remaining documents according to \hat{M} . Formally, this means we select the pair d_i and d_j that satisfies:*

$$\operatorname{argmax}_{d_i, d_j \in \mathcal{C}, i \neq j} E_{P(M^* | \mathcal{D})} \left[\mathcal{L}^{pair}(\hat{M}, M^*, i, j) \right] \quad (6.16)$$

LELPAIR selects the pair of documents with the largest pairwise contribution to the expected loss out of all pairs of documents. By presenting these documents at a high rank, the feedback given on them will reduce the uncertainty in the relative relevance of these documents. This will, in the long run, drive the expected loss contribution of the pair of documents down. Given the Glicko update rules, the pairwise contribution of all other pairs of documents will not increase. Hence this method will eventually drive the total expected loss down. Additionally, due to the exponential decay in the loss contribution from misordered pairs as the rank increases, LELPAIR tends to select pairs of documents where at least one has a high estimated relevance. Lower ranked documents are also eventually selected, but only after high rank documents have been evaluated

and their expected loss contribution is reduced. If we ignore the effect of rank in the loss function, this approach is similar to previous work in active learning where users are asked to label items where the predicted label is most uncertain (see, for example, Brinker (2004); Saar-Tsechansky and Provost (2004)). In our setting, document pairs with high pairwise contribution tend to be those with large estimated errors in relevance (σ_i and σ_j values).

Strategy 6.4. One Step Lookahead (OSL) *For each pair of documents, compute the expected pairwise loss, and the expected pairwise loss after a comparison based on the Bradley-Terry model (using \hat{M} to estimate the probability of possible outcomes) and Glicko updates. Select the pair of documents with the largest expected reduction in the pairwise loss and present these first and second. Rank the remaining documents according to \hat{M} . Formally, if \hat{M}'_{ij} is the mode of $P(M|\mathcal{D})$ after updating it given the outcome of a comparison of d_i and d_j , we select the pair d_i and d_j that satisfies:*

$$\operatorname{argmax}_{d_i, d_j \in \mathcal{C}, i \neq j} \left(E_{P(M^*|\mathcal{D})} [\mathcal{L}^{pair}(\hat{M}, M^*, i, j)] - E_{\hat{M}'_{ij}} \left[E_{P(M^*|\mathcal{D})} [\mathcal{L}^{pair}(\hat{M}'_{ij}, M^*, i, j)] \right] \right) \quad (6.17)$$

Intuitively, this strategy performs approximate gradient descent on the loss function. OSL finds the pair of documents whose contribution to the expected loss is likely to decrease most following a pairwise comparison. The expected contribution of the pair after a comparison is a weighted sum of the expected loss contribution for the two possible outcomes (either d_i wins the comparison or d_j wins). In this computation, we ignore the effect of possible rank changes for efficiency reasons. This method is also related to an approach proposed by Chajewska et al. (2000) in the context of utility estimation, where they found that the true utility of many different outcomes can be quickly discovered by maximizing the reduction in expected loss given new data.

Strategy 6.5. Largest Expected Loss Documents (LELDOC) For each document d_i , compute the total contribution of all pairs of documents including d_i to the expected loss of the ranking. Present the two documents with highest total contributions first and second, and rank the remainder according to \hat{M} . Formally, this method selects the pair d_i and d_j that satisfies:

$$\operatorname{argmax}_{d_i, d_j \in \mathcal{C}, i \neq j} \left(\sum_{a \neq i} E_{P(M^*|\mathcal{D})} [\mathcal{L}^{pair}(\hat{M}, M^*, i, a)] + \sum_{a \neq j} E_{P(M^*|\mathcal{D})} [\mathcal{L}^{pair}(\hat{M}, M^*, j, a)] \right) \quad (6.18)$$

This strategy addresses a potential limitation of LELPAIR and OSL: They only consider individual pairwise document contributions to the expected loss, despite the contributions of pairs not being independent. LELDOC addresses this by computing the total contribution of each document, summing over all pairs including that document. For example, if some document d is ranked third, it's total contribution is that from d and the top ranked document, plus the contribution from d and the second document, plus that from d and the fourth document and so forth. LELDOC selects the two documents with highest total contributions and presents them first and second. By comparing these two documents and reducing the uncertainty in their relevances, we are likely to reduce the contributions to the risk of all pairs including the documents.

An alternative selection algorithm proposed in previous work (for example by Glickman and Jensen (2005); Chu and Ghahramani (2005a)) is to compare pairs of items such that the probability distribution over models changes most in terms of KL-divergence or entropy. We do not pursue this alternative as it does not take into account the loss function being optimized.

Finally, we note that exploration strategies for rankings are related to the opponent assignment problem in sports tournaments. However, there are two key differences. First, a tournament has a different concept of loss. A criterion often optimized is the probability of the true best player winning the final game (for example, see Glickman (2008); Ryvkin (2005)). Second, pairwise comparisons in a tournament have no cost. In fact, in most sports a common constraint is that all teams or players *must* compete for at least n rounds. This means that each “item” must be compared with some other item every round. The optimization problem is to select which pairs are compared such that the loss is minimized after all these comparisons, rather than as quickly as possible.

6.4 Evaluation Methodology

We now have five strategies for eliciting useful training data from users of a search system, as well as a method to estimate the relevance of the documents using our probabilistic model. In this section, we describe how these strategies can be evaluated. In particular, we will compare how effective each strategy is at improving the quality of the rankings shown to users.

We evaluate as follows. Given an initial ranking of one thousand documents as returned by a search engine in response to a query, we derive a prior $P(M)$. This prior initializes $P(M|\mathcal{D})$. For a particular exploration strategy, we next select a ranking to present to users. We compute the loss of the presented ranking, and of the mode ranking derived from \hat{M} . Next, we simulate user behavior on the presented ranking, using a simple behavioral model, and collect training data

that is used to update the model parameters. We repeat this process 3,000 times for each initial ranking. This experimental setup is formalized in Algorithm 6.1.

In particular, step 5 in the algorithm randomly swaps the top two documents. As seen in Chapter 2, Web search users are biased toward clicking on the top ranked document irrespective of document relevance. As shown in Chapter 5, this flipping approach is sufficient to avoid presentation bias.

The behavioral model we use to simulate clickthrough data is detailed in Algorithm 6.2. By using a simulation, it is possible to evaluate the exploration strategies in detail without needing large numbers of test subjects, and to avoid effects that may be unique to specific users (for example, to academic users). Our model simplifies real behavior by assuming that users only click on top two results, and do so with probability specified by the Bradley-Terry model. This model is also motivated by the fast decay observed in the number of clicks as rank increases in real search systems. Clearly, in a real setting some additional data would be collected from lower ranks, making the results we report conservative in this respect. However, the amount of data collected about results at lower ranks would be significantly smaller.

We repeated each experiment with either 30 or 100 initial rankings, each giving a different initial set of relevance estimates. We report the mean loss across all runs (normalized such that the initial loss is 1), or the mean average precision (MAP). In some results we present a single final performance, specifically the loss or MAP after 3,000 pairwise comparisons and model updates. Note that as our rankings are of 1,000 documents, 3,000 comparisons is on average just six noisy pairwise comparisons involving each document.

Algorithm 6.1 Evaluation Setup

- 1: Input: Estimated relevances $\{\nu_i\}$ for $d_i \in \mathcal{C}$ *Provided by a search function*
 - 2: Input: σ_0 *Uncertainty in estimated provided by search function.*
 - 3: $\sigma_i \leftarrow \sigma_0$ for $d_i \in \mathcal{C}$
 - 4: **for** iteration 1 through 3,000 **do** *For 3,000 feedback rounds*
 - 5: Pick two documents d_i, d_j to rank 1st and 2nd
 - 6: Randomly swap d_i and d_j
 - 7: Show the selected ranking to user
 - 8: Record training data given user feedback
 - 9: Update $\nu_i, \nu_j, \sigma_i, \sigma_j$ per Equations 6.14 and 6.15
 - 10: **end for**
-

Algorithm 6.2 User Behavioral Model

- 1: Input: Ranking of documents $(d_1, \dots, d_{|\mathcal{C}|})$, true relevances $(\mu_1^*, \dots, \mu_{|\mathcal{C}|}^*)$
 - 2: **if** $UniformRandom(0, 1) < \frac{1}{1+10^{-(\mu_1^* - \mu_2^*)/400}}$ **then**
 - 3: Winner is d_1 : $s_1 \leftarrow 1; s_2 \leftarrow 0$
 - 4: **else**
 - 5: Winner is d_2 : $s_1 \leftarrow 0; s_2 \leftarrow 1$
 - 6: **end if**
-

6.5 Results

We start by evaluating the exploration strategies on synthetic data, where we evaluate their effectiveness if the assumed prior distribution over document relevance matches the true generating model. This will be followed by an evaluation using more realistic TREC-10 data.

6.5.1 Synthetic Data

We randomly generated a corpus of 1000 documents with expected relevances μ_i^* drawn from $\mathcal{N}(1500, 147^2)$. We chose this scale as it is comparable to typical chess scores. We then drew ten independent initial models, drawing ν_i from $\mathcal{N}(\mu_i^*, 147^2)$ and initializing $\sigma_i = 147 \equiv \sigma_0$. We repeated this process three times, giving 30 initial rankings over three different random corpora.

For each initial ranking, we ran each strategy for 3,000 iterations. Figure 6.3 shows the loss of the mode ranking at each iteration. Along the horizontal axis is the number of pairwise comparisons. After each pairwise comparison, $P(M|\mathcal{D})$ is updated and a new pair to compare is selected. The vertical axis is the average loss of the mode ranking relative to the initial average loss. The error bars indicate one standard error in the mean scaled loss.

I. Which exploration strategy learns fastest?

The first question to answer is which strategy learns fastest. The passive TOP2 approach does not lead to a meaningful overall reduction in the loss. This is because our user model assumes that users only provide feedback on the top two documents. After a few comparisons, the top few documents have their relative position correctly established and no new documents are ever compared again. Our other baseline algorithm, RANDOM, sees the loss decrease slowly.

We see that the other exploration strategies all perform substantially better than the baselines. Both LELPAIR and OSL quickly reduce the total loss by selecting pairs of documents with high contributions to the expected loss, and

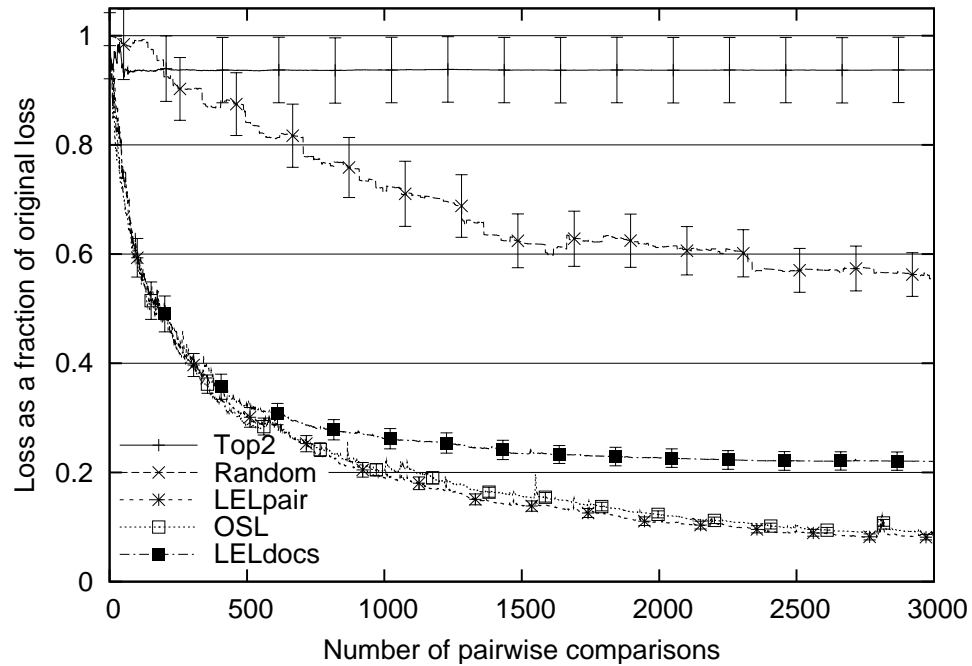


Figure 6.3: Change in loss as a function of the number of pairwise comparisons for each exploration strategy on synthetic data.

high expected reductions in it. We see this improvement continues for a large number of comparisons. In contrast, LELDOC appears to asymptote more quickly. This is because the documents selected continue to be those at high ranks even after many comparisons. In effect, LELDOC is too biased toward highly ranked documents. Comparing with LELPAIR and OSL, we see that while lower ranked documents may have lower total contributions to the expected loss, they often have higher individual pairwise contributions.

II. How robust is the approach to prior assumptions?

The second natural question to ask is how robust the results are to the weight given to the initial ranking, as in the case of real data the correct weight is likely to be unknown. This weight is encoded by the initial values of σ_i , which are

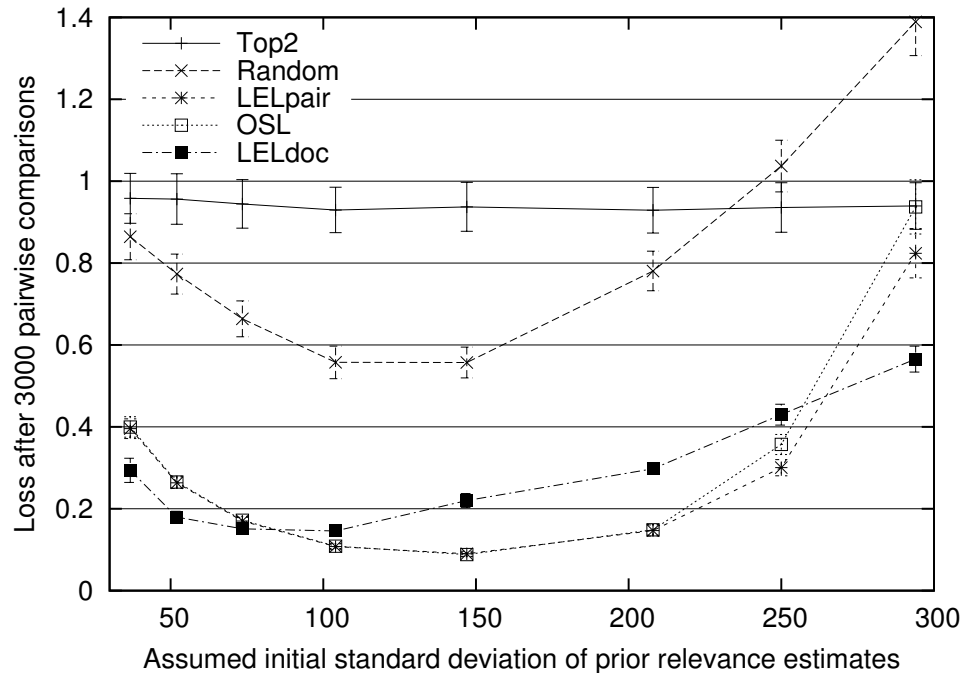


Figure 6.4: Effect of the weight given to the prior on the final loss, evaluated on synthetic data (true noise is $\sigma_0 = 147$).

set to σ_0 . We now explore the effect of changing that value. This experiment is possible because we know the level of noise used when generating synthetic data.

Figure 6.4 shows the effect of selecting an assumed noise level that differs from the true noise level. With our default setting, and that used to generate our synthetic datasets, the probability of a document in the top 10 according to the prior not being in the top 100 according to true relevance is about 8%. We can see that selecting a suboptimal σ_0 does not drastically reduce performance after a fixed number of pairwise comparisons. Apart from LELDOC, the best performance is achieved when the actual noise is correctly known. Interestingly, LELDOC performs best when the error in the prior is underestimated. In that

situation, we found that LELDOC selects documents further down the ranking for comparison, leading to better final performance.

6.5.2 TREC Data

In addition to the synthetic data, we also evaluated the exploration strategies using the TREC-10 Web track queries (topics 501 through 550) in the WT10g document corpus. This subset of the corpus includes 50 topics and topic descriptions, run as queries on documents that are part of the corpus. As part of the 10th Text REtrieval Conference (TREC-10) (Hawking & Craswell, 2001), 18 teams submitted a ranking of documents for each topic. Then, for each topic, documents ranked highly by the teams were manually judged to be either highly relevant (relevance score of 2), relevant (score of 1) or non-relevant (score of 0). All other documents in the corpus were assumed non-relevant (score of 0). The discretized nature of these relevance judgments is unrealistic, as few documents are likely to have precisely the same relevance in the real world. To compensate and make the learning problem more realistic, we added uniform random noise in the range $[-0.5, 0.5]$ to the true relevance judgments, preserving the relative order of highly relevant, relevant and non-relevant documents.

For each of the 50 TREC-10 topics, we randomly selected two submissions and used the submitted scores to initialize our model. We then repeated the evaluation described for synthetic data. Each submission includes a ranking of typically 1000 documents, with a score given to each document. The scores are unnormalized, and hence could be interpreted as a prior in any number of ways. As each ranking typically contains both highly relevant documents and

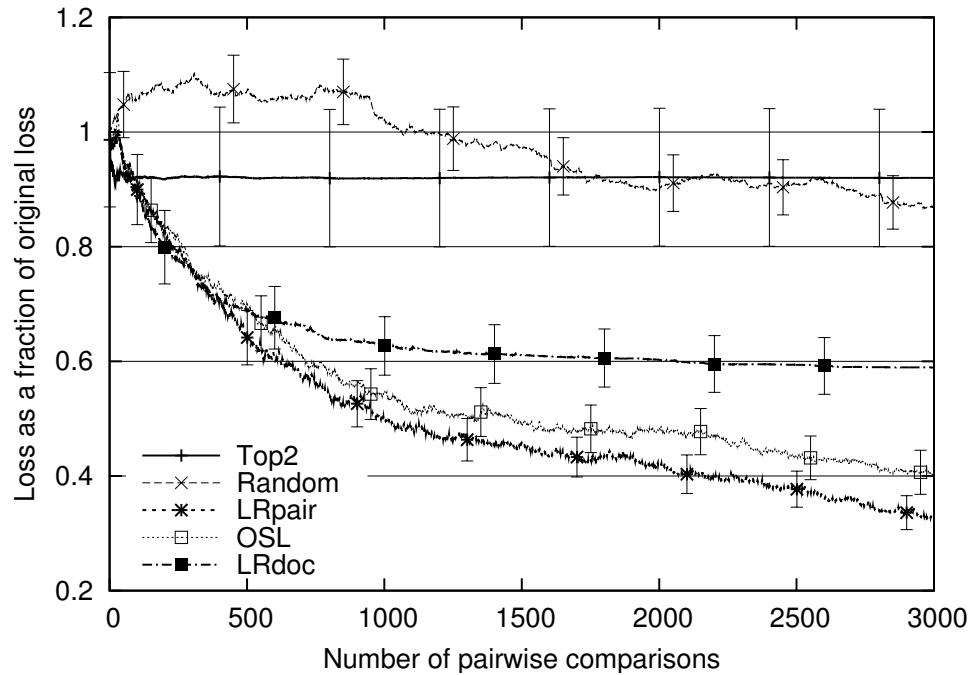


Figure 6.5: Change in loss as a function of the number of pairwise comparisons for each selection algorithm on TREC-10 data.

non-relevant documents, we chose to normalize the scores to a linear interval $[1500 + \sigma_0, 1500 - \sigma_0]$ with $\sigma_0 = 147$. The resulting scores were used to set the initial ν_i values. The initial estimated error σ_i were set to σ_0 . This means that a document with a score near the maximum score was estimated to have about a 30% percent chance of in fact being in the lower half of the ranking.

I. Which exploration strategy learns fastest?

Figure 6.5 shows the performance of the exploration strategies. We see that the loss improves rapidly with LELPAIR, OSL and LELDOC. We also see that TOP2 performs as it did on the synthetic data. On the other hand, RANDOM performs differently: the loss of the mode ranking initially increases, then improves slightly but remains high. We believe that this is due to the mismatch between

the prior and model. When two documents are compared, if the outcome is not the expected one then the update to the relevance estimates can be large. Sometimes, the lower ranked document moves to a much higher rank, and the loss does not quickly recover due to few comparisons per document. Interestingly, performance of RANDOM also depends on the loss of the initial ranking. When the initial loss is high, after 3,000 pairwise comparisons the loss tends to be reduced. The opposite is true when we start with a very good ranking.

II. How do the strategies perform with respect to MAP?

The loss function presented earlier is not one commonly used to evaluate rankings. A measure much more widely used by the research community is the mean average precision (MAP), described on page 40 in Chapter 3. To compute the MAP, we consider each document with true relevance μ_i^* above some threshold as relevant, and others as irrelevant. The average precision of a ranking is the average of the precision measured at each relevant document¹¹. The MAP score is the mean of the average precisions across all 100 experiments. We used a threshold of 0.5 scaled in the same way the scores were scaled.

Figure 6.6 shows how the MAP of the ranking changes as more pairwise comparisons are performed. MAP visibly behaves very similarly to our loss function. We see that LELPAIR and OSL result in the largest MAP improvement, and appear likely to continue to improve further with more comparisons. As before, LELDOC performance plateaus quickly and TOP2 sees almost no improvement.

¹¹For simplicity, we only consider the 1000 ranked documents when computing the MAP score. While the corpus may contain relevant documents that never made it into the top 1000, these do not contribute to the MAP score we compute.

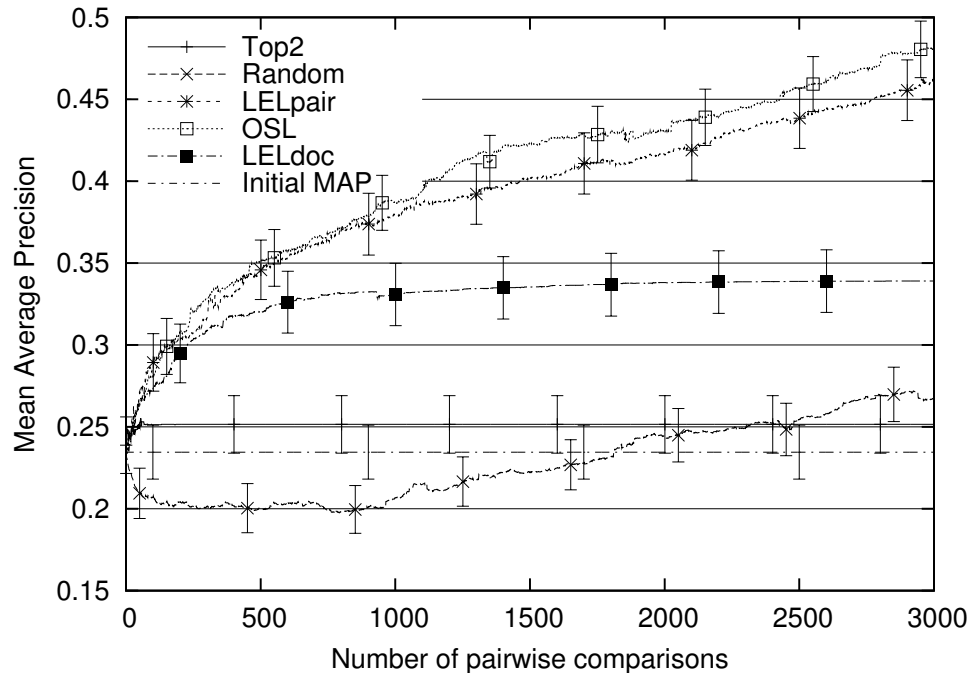


Figure 6.6: Change in MAP score as a function of the number of pairwise comparisons for each selection algorithm on TREC-10 data.

RANDOM performs poorly, with an initial drop in MAP although after 2,000 pairwise comparisons the MAP is above baseline.

III. How does noise influence learning speed?

So far, our simulation has assumed a particular amount of noise in user clicks. Given two documents that differ in true relevance by one TREC relevance level, our parameter settings specify that the user will click on the more relevant one 70% of the time. While this level of noise is realistic (Joachims et al, 2007), it is of interest to observe what would happen to the difficulty of the learning task if the noise level were different. We modified our user model to change the level of noise, and changed the Glicko update equations equivalently. Figure 6.7 shows the final MAP score after 3,000 pairwise comparisons as the level of noise

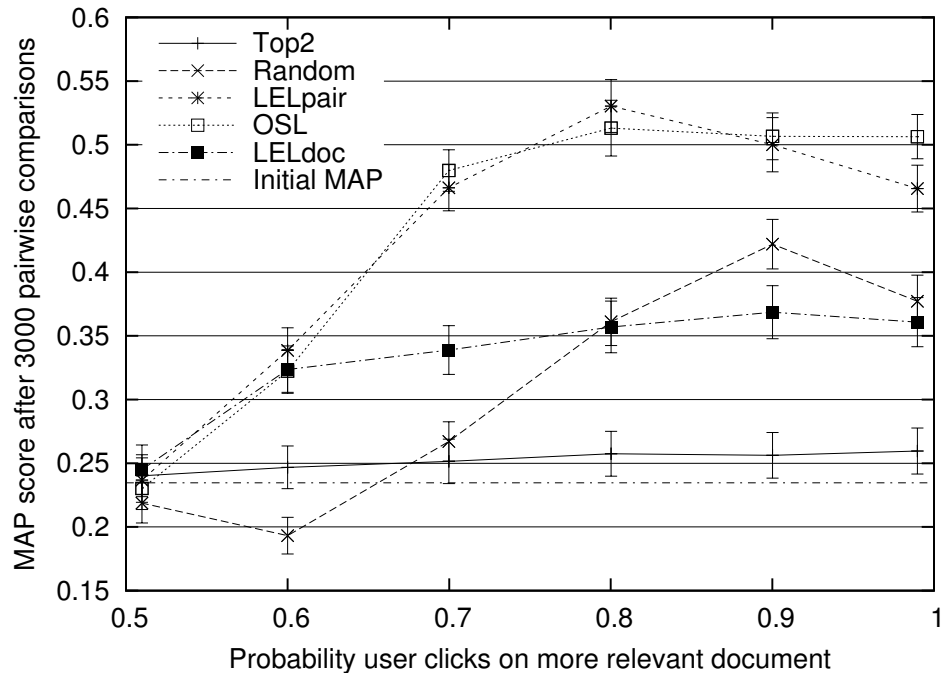


Figure 6.7: Effect of different noise levels in pairwise preferences on final MAP score, evaluated on TREC-10 data.

changes. It shows us that, irrespective of the noise level, the best exploration strategies remain LELPAIR and OSL. On the left of the figure we see that if there is a lot of noise in the preferences, all the algorithms perform more poorly. This is to be expected, given that the amount of information contained in 3,000 pairwise preferences decreases as the amount of noise in user clicks increases. On the other end of the scale, we see that even if users select the more relevant document almost 100% of the time, the final MAP score does not reach very high values. This appears to be a side-effect of the normal approximation implicit in the Glicko updates.

IV. How robust is this method to noise assumptions?

The above results assume the noise level is known in advance. Figure 6.8 shows the effect of a mismatch between the assumed noise level (used to scale the initial scores, and in the Glicko updates) and the true level of noise in pairwise preferences (in the user model). Along the horizontal axis we have the probability a user correctly selects the more relevant document in a pair, if the true relevances of the pair differ by one TREC relevance level. The figure shows how the MAP of the mode ranking after 3,000 pairwise comparisons is affected by different estimates of the noise in pairwise clicks. Each line corresponds to a different assumed noise level. The figure shows that if the amount of noise is underestimated, performance is poorer although not drastically so (unless the noise is underestimated substantially). On the other hand, we see that if the pairwise preferences are less noisy than assumed, the final performance does not suffer.

Of particular interest, these results tell us that the best strategy is to assume the level of noise conservatively to see the best performance improvements. Finally, it is worth noting that in the case of extremely noisy clicks it may be beneficial to aggregate user clicks. Specifically, we could collect a number of pairwise judgments for each pair of documents, and then count these as a single virtual pairwise judgment for the document with the most preference votes. This would reduce the effective level of noise in pairwise judgments at the expense of performing fewer updates given the same number of comparisons.

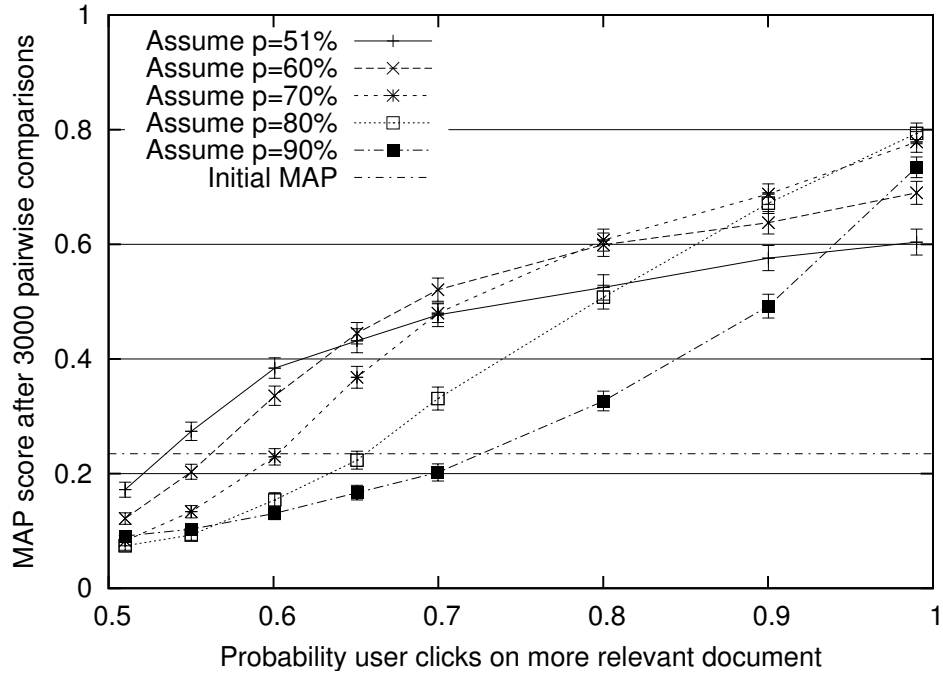


Figure 6.8: Effect of incorrect assumptions about the noise level in relevance judgments on final MAP score, evaluated on TREC-10 data using OSL.

Table 6.1: Mean Average Precision after 3000 iterations of optimizing different loss variants using OSL.

Loss Function	MAP Score
$e^{-r_{ij}} \left((\mu_i^* - \mu_j^*) - (\nu_i - \nu_j) \right)^2 \mathbf{1}_{misordered}$	0.481 ± 0.017
$\left((\mu_i^* - \mu_j^*) - (\nu_i - \nu_j) \right)^2 \mathbf{1}_{misordered}$	0.281 ± 0.017
$e^{-r_{ij}} \left((\mu_i^* - \mu_j^*) - (\nu_i - \nu_j) \right)^2$	0.287 ± 0.012
$e^{-r_{ij}} \mathbf{1}_{misordered}$	0.337 ± 0.020

V. Which loss functions are a good proxy for MAP?

So far, our experimental results show that minimizing the expected loss also improves the MAP. We now demonstrate that the loss function defined in Equation 6.10 leads to particularly good MAP performance. Table 6.1 shows the MAP performance after 3,000 pairwise comparisons if we optimize OSL to different variants of the loss function presented, modifying it to remove particular properties of the loss function.

We see that using a loss function without exponential decay, without a distance penalty or without a hinge leads to substantial and significant reductions in the final MAP scores. In particular, optimizing a loss function that simply depends on document ranks, rather than on the actual relevance estimates (the fourth line of the table) leads to poorer MAP performance. This shows that despite MAP only being sensitive to document order, minimizing error in relevance estimates leads to better MAP performance.

6.5.3 Controlling for Presentation Loss

The figures in the previous two sections show the loss and MAP of the mode ranking as pairwise preference data is collected. However, as described in Algorithm 6.1, when collecting data, the ranking of documents that is presented to users is not the mode ranking. Rather, users see rankings with a pair of results inserted at the top. This means that usually the presented ranking has a higher loss and a lower MAP score than the mode ranking. The difference between the MAP of the presented and mode rankings is shown in Figure 6.9.

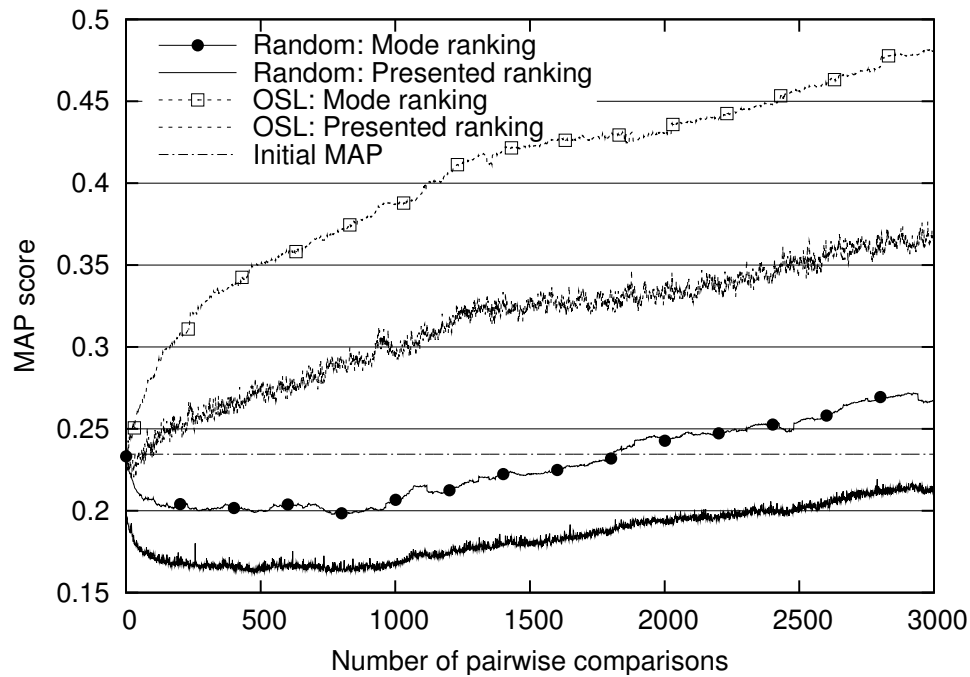


Figure 6.9: MAP scores of the mode rankings and the presented rankings, as a function of number of pairwise comparisons for OSL and RANDOM.

The top two lines are for OSL. The top line shows the MAP of the mode ranking. The second line shows the MAP of the ranking shown to users. We see that while the presented ranking is worse than the mode ranking, it is also almost immediately above the initial MAP, and improves quickly as data is collected. The separation between the two rankings increases because as the relevance of higher ranked documents is established, initially lower ranked documents are shown at the top more often. We see a similar effect comparing the presented and mode rankings of the RANDOM strategy in the lower two lines, although the presented ranking does not have a higher MAP than the initial ranking for all 3,000 pairwise comparisons. We also note that the MAP of the presented ranking using OSL is substantially better than that of the mode ranking when the RANDOM exploration strategy is used.

An interesting final experiment is to consider the tradeoff between the quality of the presented ranking and the quality of the mode ranking. One possibility to reduce the impact of data collection would be to present selected pairs at lower ranks instead of at the top two positions. With real users, this would lead to a reduction in the amount of data collected, but may improve the MAP of the presented ranking by reducing the performance gap. Figure 6.10 shows how the MAP score of the presented ranking after 3,000 pairwise comparisons would change if the selected pair was presented at a lower rank. The reduction in the amount of data collected as a function of rank is shown along the horizontal axis. For example, if for some user population moving the selected pair down by one rank reduces the number of pairwise preferences collected by 60%, the correct point for an evaluation would be 0.4 on the horizontal axis. This would mean that by presenting a pair at rank 2 and 3 rather than 1 and 2, we would only receive 40% as many clicks. Presenting selected pairs at ranks 3 and 4 would receive 16% as many clicks. At 0.4, we see that due to the reduction in the amount of data collected, the MAP of the *presented* ranking (after 3,000 rankings being shown to users) would be lower if the selected pairs were at ranks 3 and 4 rather than at ranks 1 and 2.

Taking Figure 6.10 and overlaying the MAP of the mode rankings for all three ranks, we get Figure 6.11. It shows that the separation between the mode ranking and the presented ranking decreases as the rank of the selected pair is decreased, as we may expect. However, note that as less data is collected, the MAP of the mode ranking after 3,000 pairwise comparisons also decreases. This is especially the case if much less data is collected when pairs are presented at lower ranks.

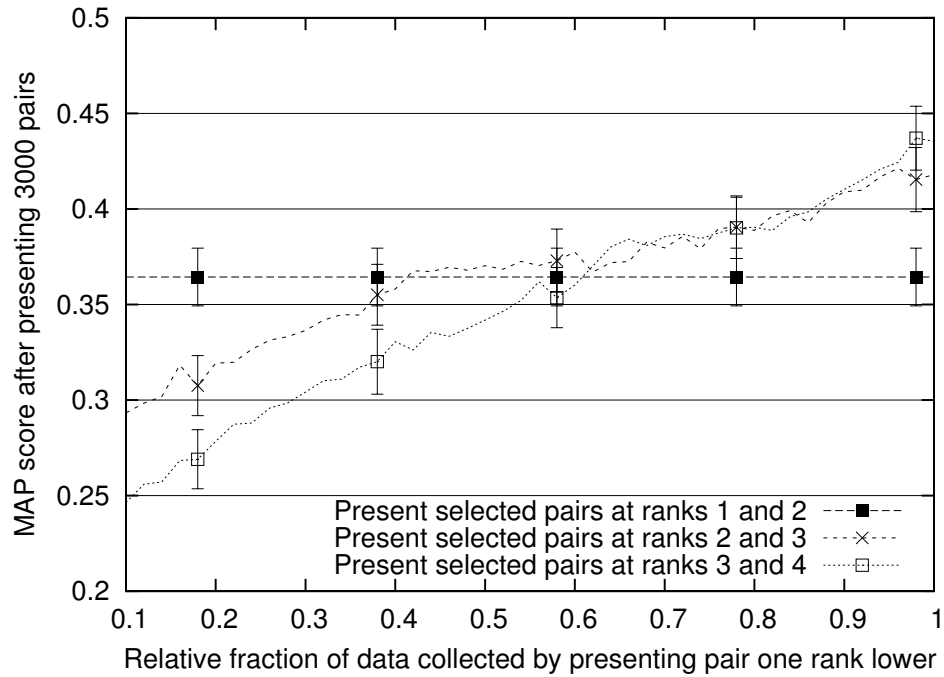


Figure 6.10: MAP scores of the presented ranking after 3,000 pairwise comparisons after presenting selected pairs at different ranks.

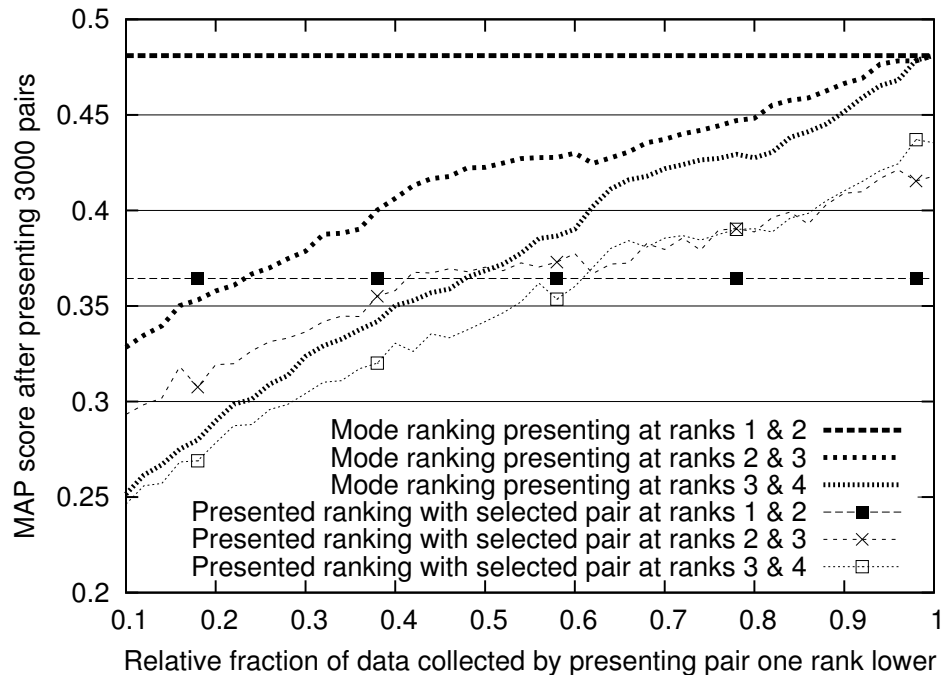


Figure 6.11: MAP scores of the mode and presented rankings after 3,000 pairwise comparisons.

6.6 Summary

This chapter has presented a new formulation of the learning to rank problem, where the goal is to minimize the total loss from presenting poor ranking functions over all time. We have formalized this problem, which we call the *ranked relevance elicitation problem*. The results presented demonstrate that by using active exploration, the quality of rankings shown to users can be improved faster than by collecting pairwise training data passively or naively. We presented a number of strategies to minimize the expected loss and showed that two in particular perform well. Our experiments showed a significant level of robustness to noise in the clickthrough data and to prior assumptions. We also demonstrated how presentation loss and the quality of learned ranking can be traded off.

CHAPTER 7

OPTIMIZING RANKINGS WITH DEPENDENT DOCUMENT RELEVANCES

An overarching assumption throughout the work presented so far in this thesis is that, given a document and a query, there exists a single number that captures the relevance of that document to the query. In particular, this relevance score is assumed to be fixed irrespective of which other documents may or may not be presented to users, and how relevant they may be. This assumption is also usually made whenever relevance judgments are collected from human experts. In this chapter we will argue that in many situations this assumption does not hold. We will then address the question of what should be optimized if not the correct fixed relevance estimates. This chapter introduces two theoretically appealing algorithms that optimize performance under a different relevance model, and shows that they lead to diverse rankings of documents being presented to users. The research presented in this chapter was originally published in (Radlinski et al, 2008b).

7.1 Introduction

Most previous work on learning to rank has assumed the availability of training data that either consists of relevance judgments for individual (query, document) pairs, or relevance judgments about the relative relevance of two documents to the same query. As described in previous chapters, these judgments are typically then used to optimize a ranking function offline, to a standard information retrieval metric. The learned function is then deployed in a live search engine.

This chapter proposes a new learning to rank problem formulation that differs in three fundamental ways. First, like the remainder of this thesis, the goal is to learn from and optimize to usage data rather than manually labeled relevance judgments. Although some researchers have transformed usage data into relevance judgments, as shown in Chapters 2 and 4, the goal here is to go one step further by also directly optimizing a usage-based error metric.

Second, we propose an online learning approach for learning from usage data. As training data is being collected, it immediately impacts the rankings shown. The learning problem this allows us to address is regret minimization: minimize the total number of poor rankings displayed over *all* time. In particular, in an online learning setting there is a natural tradeoff between exploration and exploitation – it may be valuable in the long run to present some rankings with unknown documents, to allow training data about these documents to be collected. In contrast, in the short run exploitation is typically optimal. With only few exceptions (such as the work presented in the previous chapter), previous work does not consider such an online approach. The work here differs in that we minimize regret rather than just future loss.

Third and most importantly, with few exceptions (such as Chen and Karger (2006)), previous algorithms for learning to rank have considered the relevance of each document independently of other documents. This is reflected in the performance measures typically optimized, such as Precision, Recall, Mean Average Precision (MAP) (Manning et al, 2008) and Normalized Discounted Cumulative Gain (NDCG) (Järvelin & Kekäläinen, 2000), as described in Chapter 3. Intuitively it stands to reason that presenting many slight variations of the same highly relevant document in Web search results may lead to high MAP and NDCG

scores, yet may be suboptimal for users. Moreover, Web queries often have different meanings for different users (a canonical example is the query *jaguar*) suggesting that a ranking with diverse documents may often be preferable.

This chapter will show how clickthrough data can be used to learn rankings maximizing the probability that any new user will find at least one relevant document high in the ranking.

7.2 Related Work

As we have seen throughout this thesis, the standard approach for learning to rank uses training data to learn parameters θ for a scoring function $f(q, d_i, \theta)$. Given a new query q , this function computes $f(q, d_i, \theta)$ for each document d_i *independently* and ranks documents by decreasing score (as done by, for example, Herbrich et al. (2000); Joachims (2002); Burges et al. (2005); Chu and Ghahramani (2005b), as well as in Chapter 4 of this thesis). This also applies to recent algorithms that learn θ to maximize nonlinear performance measures such as MAP (Metzler & Croft, 2005; Yue et al, 2007) and NDCG (Burges et al, 2006; Taylor et al, 2008).

As noted in Chapter 1, such an approach is justified by the probabilistic ranking principle (Robertson, 1977). In suggesting that documents be ranked by their probability of relevance to the query. This principle assumes that there are no statistical dependencies between the probabilities of relevance among documents. This assumption is violated in practice. For instance, if one document about jaguar cars is not relevant to a user who issues the query *jaguar*, other car

pages become less likely to be relevant. Also, users are often satisfied with finding a small number of, or even just one, relevant document. Hence the usefulness and relevance of a document does depend on other documents ranked higher.

As a result, most search engines today attempt to eliminate redundant results and produce *diverse* rankings that include documents that are potentially relevant to the query for different reasons. However, learning optimally diverse rankings using expert judgments would require document relevance to be measured for different possible meanings of a query. While the TREC interactive track¹² provides some documents labeled in this way for a small number of queries, such document collections are even more difficult to create than standard expert labeled collections.

Several non-learning algorithms for obtaining a diverse ranking of documents from a non-diverse ranking have been proposed. One common one is Maximal Marginal Relevance (MMR), introduced by Carbonell and Goldstein (1998). Given a similarity (relevance) measure between documents and queries $sim_1(d, q)$ and a similarity measure between pairs of documents $sim_2(d_i, d_j)$, MMR iteratively selects documents by repeatedly finding

$$d_i = \operatorname{argmax}_{d \in \mathcal{C}} \left(\lambda sim_1(d, q) - (1 - \lambda) \max_{d_j \in S} sim_2(d, d_j) \right) \quad (7.1)$$

where \mathcal{C} is the collection of documents, S is the set of documents already selected and λ is a tuning parameter. In this way MMR selects the most relevant documents that are also different from any documents already selected.

Critically, MMR requires that the relevance function $sim_1(d, q)$, and the similarity function $sim_2(d_i, d_j)$ be already known. It is usual to obtain sim_1 and sim_2

¹²http://trec.nist.gov/data/t11_interactive/t11i.html

using algorithms such as those discussed earlier in this thesis. The goal of MMR is simply to rerank an already learned ranking (that of ranking documents by decreasing sim_1 score) to improve diversity.

Other researchers have explored alternative algorithms, although with similar assumptions. For instance Zhu et al. (2007) proposed using a random walk over a graph constructed from the document collection, encoding document similarity in the structure of the graph and using relevance information to specify the starting points of this random walk. As with MMR, the similarity between documents, and the document relevance, must be obtained prior to running this algorithm. Zhai et al. (2003) and Zhang et al. (2005) also proposed other algorithms for this same problem, but assumed specific document similarity measures that are not learned. Thus those similarity measures do not necessarily correctly encode users' concept of document similarity or redundancy.

Using a different line of reasoning, Chen and Karger (2006) presented an alternative approach. They proposed to directly estimate the probability that relevant documents are present in a result set. Specifically, they suggest that when given a query q , the first document d_1 be selected to have maximal probability of relevance to the query: $d_1 = \operatorname{argmax}_d P(d \text{ is relevant} \mid q)$. The second document can then be selected to maximize the probability of relevance given that d_1 was not relevant: $d_2 = \operatorname{argmax}_d P(d \text{ is relevant} \mid q, d_1 \text{ is not relevant})$. In a similar way, documents can be selected one at a time to maximize the probability that a relevant document is found. However, like MMR and the other related approaches, we still need a model of relevance that can be provided as input, and that can quickly compute the probability of relevance conditioned on

other documents not being relevant. In contrast, we now present algorithms that directly learn a diverse ranking of documents using users' clicking behavior.

7.3 Problem Formalization

We address the problem of learning an optimally diverse ranking of documents $\mathcal{C} = \{d_1, \dots, d_n\}$ for one fixed query. Suppose we have a population of users, where each user u_i considers some subset of documents $A_i \subset \mathcal{C}$ as relevant to the query, and the remainder of the documents as non-relevant. Intuitively, users with different interpretations for the query would have different relevant sets, while users with similar interpretations would have similar relevant sets. An illustration of users and documents they find relevant for one query is shown in Figure 7.1. In this example, users 1 and 2 find similar documents relevant while user 1 and user 3 disagree on the meaning of the query.

At time t , we interact with user u_t with relevant set A_t . We present an ordered set of k documents, $B_t = (b_1(t), \dots, b_k(t))$. The user considers the results in order, and clicks on up to one document. The probability of user u_t clicking on document d_i (conditional on the user not clicking on a document presented earlier in the ranking) is assumed to be $p_{ti} \in [0, 1]$. We refer to the vector of probabilities $(p_{ti})_{i \in \mathcal{C}}$ as the *type* of user u_t . In the simplest case, we could take $p_{ti} = 1$ if $d_i \in A_t$ and 0 otherwise, in which case the user clicks on the first relevant document or does not click if no documents in B_t are relevant. However, in reality clicks tend to be noisy although more relevant documents are more likely to be clicked on. In our analysis, we will take $p_{ti} \in [0, 1]$.

	<i>Document 1</i>	<i>Document 2</i>	<i>Document 3</i>	<i>Document 4</i>	<i>Document 5</i>	<i>Document 6</i>	<i>Document 7</i>	<i>Document 8</i>	<i>Document 9</i>
User 1	✓				✓		✓	✓	
User 2		✓			✓		✓	✓	
User 3		✓	✓	✓		✓			
User 4	✓				✓		✓	✓	
User 5	✓		✓		✓			✓	
User 6			✓			✓			
User 7	✓				✓		✓	✓	
User 8					✓		✓		✓

Figure 7.1: Example of users and relevant documents for some query. We see that some users have very similar interests (e.g. user 1 and user 2) while others have different interests (e.g. user 1 and user 3). Each row represents A_i for that user.

We get payoff 1 if the user clicks, 0 if not. The goal is to maximize the total payoff, summing over all time. This payoff represents the number of users who clicked on any result, which can be interpreted as the user finding at least one potentially relevant document (so long as p_{ti} is higher when $d_i \in A_t$ than when $d_i \notin A_t$).

The event that a user does not click is called *abandonment* since the user abandoned the search results.

Definition 7.1. *The abandonment rate measures the fraction of queries for which search engine users do not click on any of the search results returned.*

Abandonment is an important measure of user satisfaction because it indicates that the user was likely presented with search results of no potential interest¹³.

7.4 Learning Algorithms

We now present two algorithms that seek to directly minimize the abandonment rate. At a high level, both algorithms learn a marginal utility for each document at each rank, displaying documents to maximize the probability that a new user of the search system would find at least one relevant document within the top k positions. The algorithms differ in their assumptions.

7.4.1 Ranked Explore and Commit

The first algorithm we present is a simple greedy strategy that assumes that user interests and documents do not change over time. As we will see, after T time steps this algorithm achieves a payoff of at least $(1 - 1/e - \epsilon)OPT - O(k^3n/\epsilon^2 \ln(k/\delta))$ with probability at least $1 - \delta$. OPT denotes the maximal payoff that could be obtained if the click probabilities p_{ti} were known ahead of time for all users and documents, and $(1 - 1/e)OPT$ is the best obtainable polynomial time approximation, as will be explained in Section 7.5.1.

As described in Algorithm 7.1, Ranked Explore and Commit (REC) iteratively selects documents for each rank. At each rank position i , every document

¹³Although abandonment may also be the result of users finding answers directly on the search results page, our model does not consider this possibility.

Algorithm 7.1 Ranked Explore and Commit

```
1: input: Documents  $(d_1, \dots, d_n)$ , parameters  $\epsilon, \delta, k$ .
2:  $x \leftarrow \lceil 2k^2 / \epsilon^2 \log(2k/\delta) \rceil$ 
3:  $(b_1, \dots, b_k) \leftarrow k$  arbitrary documents.
4: for  $i=1 \dots k$  do ..... At every rank
5:    $\forall j. p_j \leftarrow 0$ 
6:   for counter=1 ...  $x$  do ..... Loop  $x$  times
7:     for  $j=1 \dots n$  do ..... over every document  $d_j$ 
8:        $b_i \leftarrow d_j$ 
9:       display  $\{b_1, \dots, b_k\}$  to user; record clicks
10:      if user clicked on  $b_i$  then  $p_j \leftarrow p_j + 1$ 
11:    end for
12:  end for
13:   $j^* \leftarrow \operatorname{argmax}_j p_j$  ..... Commit to best document at this rank
14:   $b_i \leftarrow d_{j^*}$ 
15: end for
```

d_j is presented a fixed number x times, and the number of clicks it receives during these presentations is recorded. After nx presentations, the algorithm permanently assigns the document that received the most clicks to the current rank, and moves on to the next rank.

7.4.2 Ranked Bandits Algorithm

Ranked Explore and Commit is purely greedy, meaning that after each document is selected, this decision is never revisited. In particular, this means that if user interests or documents change, REC can perform arbitrarily poorly. In contrast, the Ranked Bandits Algorithm (RBA) achieves a combined payoff of $(1 - 1/e)OPT - O(k\sqrt{Tn \ln n})$ after T time steps even if documents and user interests change over time.

Algorithm 7.2 Ranked Bandits Algorithm

```
1: initialize  $MAB_1(n), \dots, MAB_k(n)$  ..... Initialize multi-armed bandits
2: for  $t = 1 \dots T$  do
3:   for  $i = 1 \dots k$  do ..... Sequentially select documents
4:      $\hat{b}_i(t) \leftarrow \text{select-arm}(MAB_i)$ 
5:     if  $\hat{b}_i(t) \in \{b_1(t), \dots, b_{i-1}(t)\}$  then ..... Replace repeats
6:        $b_i(t) \leftarrow \text{arbitrary unselected document}$ 
7:     else
8:        $b_i(t) \leftarrow \hat{b}_i(t)$ 
9:     end if
10:  end for
11:  display  $B_t = \{b_1(t), \dots, b_k(t)\}$  to user; record clicks
12:  for  $i = 1 \dots k$  do ..... Determine feedback for  $MAB_i$ 
13:    if user clicked  $b_i(t)$  and  $\hat{b}_i(t) = b_i(t)$  then
14:       $f_{it} = 1$ 
15:    else
16:       $f_{it} = 0$ 
17:    end if
18:    update  $(MAB_i, \text{arm} = \hat{b}_i(t), \text{reward} = f_{it})$ 
19:  end for
20: end for
```

This algorithm leverages standard theoretical results for multi-armed bandits. Multi-armed bandits (MAB) are modeled on casino slot machines (sometimes called one-armed bandits). The goal of standard MAB algorithms is to select the optimal sequence of slot machines to play to maximize the expected total reward collected. For further details, refer to Auer et al. (2002a) and Kleinberg (2005). The ranked bandits algorithm runs an MAB instance MAB_i for *each rank* i . Each of the k copies of the multi-armed bandit algorithm maintains a value (or index) for every document. When selecting the ranking to display to users, the algorithm MAB_1 is responsible for choosing which document is shown at rank 1. Next, the algorithm MAB_2 determines which document is shown at rank 2, unless the same document was selected at the highest rank. In that case, the

second document is picked arbitrarily. This process is repeated to select all top k documents.

Next, after a user considers up to the top k documents in order, and clicks on one or none, we need the MAB instances to update their indices. If the user clicks on a document actually selected by an MAB instance, the reward for the arm corresponding to that document for the multi-armed bandit at that rank is 1. The reward for the arms corresponding to all other selected documents is 0. In particular, note that the RBA treats the bandits corresponding to each rank independently. Precise pseudo-code for the algorithm is presented in Algorithm 7.2. A generalization of this algorithm, in an abstract setting without the application to information retrieval, was discovered independently by Streeter and Golovin (2007).

The actual MAB algorithm used for each MAB_i instance is not critical, and in fact any algorithm for the non-stochastic multi-armed bandit problem will suffice. Our theoretical analysis only requires that:

- The algorithm has a set S of n strategies.
- In each period t a payoff function $f_t : S \rightarrow [0, 1]$ is defined. This function is not revealed to the algorithm, and may depend on the algorithm's choices before time t .
- In each period t the algorithm chooses a (possibly random) element $y_t \in S$ based on the feedback revealed in prior periods.
- The feedback revealed in period t is $f_t(y_t)$.

- The expected payoffs of the chosen strategies satisfy:

$$\sum_{t=1}^T \mathbf{E}[f_t(y_t)] \geq \max_{y \in \mathcal{S}} \sum_{t=1}^T \mathbf{E}[f_t(y)] - R(T) \quad (7.2)$$

where $R(T)$ is an explicit function in $o(T)$ which depends on the particular multi-armed bandit algorithm chosen, and the expectation is over any randomness in the algorithm. We will use the **Exp3** algorithm in our analysis, where $R(T) = O(\sqrt{Tn \ln n})$ (Auer et al, 2002b).

We will also later see that although these conditions are needed to bound worst-case performance, better practical performance may be obtained at the expense of worst-case performance if they are relaxed.

7.5 Theoretical Analysis

We now present a theoretical analysis of the algorithms presented in Section 7.4. First however, we discuss the offline version of this optimization problem.

7.5.1 The Offline Optimization Problem

The problem of choosing the optimum set of k documents for a given user population is NP-hard, even if all the information about the user population (i.e. the set of relevant documents for each user) is given offline and we restrict ourselves to $p_{ti} \in \{0, 1\}$. This is because selecting the optimal set of documents is equivalent to the maximum coverage problem: Given a positive integer k and a collection of subsets S_1, S_2, \dots, S_n of an m -element set, find k of the subsets whose union has the largest possible cardinality.

The standard greedy algorithm for the maximum coverage problem, translated to our setting, iteratively chooses the document that is relevant to the most users for whom a relevant document has not yet been selected. This algorithm is a $(1 - 1/e)$ -approximation algorithm for this maximization problem (Nemhauser et al, 1978). The $(1 - 1/e)$ factor is optimal and no better worst-case approximation ratio is achievable in polynomial time unless $NP \subseteq DTIME(n^{\log \log n})$ (Khuller et al, 1997).

7.5.2 Analysis of Ranked Bandits Algorithm

We start by analyzing the Ranked Bandits Algorithm. This algorithm works by simulating the offline greedy algorithm, using a separate instance of the multi-armed bandit algorithm for each step of the greedy algorithm. Except for the sublinear regret term, the combined payoff is as high as possible without violating the hardness-of-approximation result stated in the preceding paragraph.

To analyze the RBA, we first restrict ourselves to users who click on any given document with probability either 0 or 1. We refer to this restricted type of user as a *deterministic user*; we will relax the requirement later. Additionally, this analysis applies to a worst case (and hence fixed) sequence of users.

Further, it is useful to introduce some notation. For a set A and a sequence $B = (b_1, b_2, \dots, b_k)$, let

$$G_i(A, B) = \begin{cases} 1 & \text{if } A \text{ intersects } \{b_1, \dots, b_i\} \\ 0 & \text{otherwise} \end{cases} \quad (7.3)$$

$$g_i(A, B) = G_i(A, B) - G_{i-1}(A, B) \quad (7.4)$$

Recalling that A_t is the set of documents relevant to user u_t , we see that $G_k(A_t, B)$ is the payoff of presenting B to the user u_t . Let

$$B^* = \operatorname{argmax}_B \sum_{t=1}^T G_k(A_t, B), \quad (7.5)$$

$$OPT = \sum_{t=1}^T G_k(A_t, B^*). \quad (7.6)$$

Recall that $(\hat{b}_1(t), \dots, \hat{b}_k(t))$ is the sequence of documents chosen by the algorithms $\text{MAB}_1, \dots, \text{MAB}_k$ at time t , and that $(b_1(t), \dots, b_k(t))$ is the sequence of documents presented to the user. We define the feedback function f_{it} for algorithm MAB_i at time t , as follows:

$$f_{it}(b) = \begin{cases} 1 & \text{if } G_{i-1}(A_t, B_t) = 0 \text{ and } b \in A_t \\ 0 & \text{otherwise} \end{cases}. \quad (7.7)$$

Note that the value of f_{it} defined in the pseudocode for the Ranked Bandits Algorithms is equal to $f_{it}(\hat{b}_i(t))$.

Lemma 7.1. *For all i ,*

$$\mathbf{E} \left[\sum_{t=1}^T g_i(A_t, B_t) \right] \geq \frac{1}{k} \mathbf{E} \left[\sum_{t=1}^T (G_k(A_t, B^*) - G_{i-1}(A_t, B_t)) \right] - R(T) \quad (7.8)$$

$$= \frac{1}{k} OPT - \frac{1}{k} \mathbf{E} \left[\sum_{t=1}^T G_{i-1}(A_t, B_t) \right] - R(T). \quad (7.9)$$

Proof. First, note that

$$g_i(A_t, B_t) \geq f_{it}(\hat{b}_i(t)). \quad (7.10)$$

This is trivially true when $f_{it}(\hat{b}_i(t)) = 0$. When $f_{it}(\hat{b}_i(t)) = 1$, $G_{i-1}(A_t, B_t) = 0$ and $\hat{b}_i(t) \in A_t$. This implies that $b_i(t) = \hat{b}_i(t)$ and that $g_i(A_t, B_t) = 1$.

Now using the regret bound for MAB_i we obtain

$$\begin{aligned} \sum_{t=1}^T \mathbf{E}[f_{it}(\hat{b}_i(t))] &\geq \max_b \sum_{t=1}^T \mathbf{E}[f_{it}(b)] - R(T) \\ &\geq \frac{1}{k} \mathbf{E} \left[\sum_{b \in B^*} \sum_{t=1}^T f_{it}(b) \right] - R(T). \end{aligned} \quad (7.11)$$

To complete the proof of the lemma, we will prove that

$$\sum_{b \in B^*} f_{it}(b) \geq G_k(A_t, B^*) - G_{i-1}(A_t, B_t). \quad (7.12)$$

The lemma follows immediately by combining (7.10)-(7.12). Observe that the left side of (7.12) is a non-negative integer, while the right side takes one of the values $\{-1, 0, 1\}$. Thus, to prove (7.12) it suffices to show that the left side is greater than or equal to 1 whenever the right side is equal to 1. The right side equals 1 only when $G_{i-1}(A_t, B_t) = 0$ and A_t intersects B^* . In this case it is clear that there exists at least one $b \in B^*$ such that $f_{it}(b) = 1$, hence the left side is greater than or equal to 1. \square

Theorem 7.1. *The algorithm's combined payoff after T rounds satisfies:*

$$\mathbf{E} \left[\sum_{t=1}^T G_k(A_t, B_t) \right] \geq \left(1 - \frac{1}{e}\right) OPT - kR(T). \quad (7.13)$$

Proof. We will prove, by induction on i , that

$$OPT - \mathbf{E} \left[\sum_{t=1}^T G_i(A_t, B_t) \right] \leq \left(1 - \frac{1}{k}\right)^i OPT + iR(T). \quad (7.14)$$

The theorem follows by taking $i = k$ and using the inequality $(1 - \frac{1}{k})^k < \frac{1}{e}$. In the base case $i = 0$, inequality (7.14) is trivial. For the induction step, let

$$Z_i = OPT - \mathbf{E} \left[\sum_{t=1}^T G_i(A_t, B_t) \right]. \quad (7.15)$$

We have

$$Z_i = Z_{i-1} - \mathbf{E} \left[\sum_{t=1}^T g_i(A_t, B_t) \right], \quad (7.16)$$

and Lemma 7.1 says that

$$\mathbf{E} \left[\sum_{t=1}^T g_i(A_t, B_t) \right] \geq \frac{1}{k} Z_{i-1} - R(T). \quad (7.17)$$

Combining (7.16) with (7.17), we obtain

$$Z_i \leq \left(1 - \frac{1}{k}\right) Z_{i-1} + R(T). \quad (7.18)$$

Combining this with the induction hypothesis proves (7.14). \square

The general case, in which user u_i 's type vector $(p_{ij})_{j \in \mathcal{C}}$ is an arbitrary element of $[0, 1]^{\mathcal{C}}$, can be reduced via a simple transformation to the case of deterministic users analyzed above. We replace user u_i with a *random* deterministic user \hat{u}_i whose type vector $\hat{p}_i \in \{0, 1\}^{\mathcal{C}}$ is sampled using the following rule: the random variable \hat{p}_{ij} has distribution

$$\hat{p}_{ij} = \begin{cases} 1 & \text{with probability } p_{ij} \\ 0 & \text{with probability } 1 - p_{ij}, \end{cases} \quad (7.19)$$

and these random variables are mutually independent. Note that the clicking behavior of user u_i when presented with a ranking B is *identical* to the clicking behavior observed when a random user type \hat{u}_i is sampled from the above distribution, and the ranking B is presented to \hat{u}_i . Thus, if we apply the specified transformation to users u_1, u_2, \dots, u_T , obtaining a random sequence $\hat{u}_1, \hat{u}_2, \dots, \hat{u}_T$ of deterministic users, this transformation changes neither the algorithm's expected payoff nor that of the optimum ranking B^* . Thus, Theorem 7.1 for general users can be deduced by applying the same theorem to the random sequence

$\hat{u}_1, \dots, \hat{u}_T$ and taking the expectation of the left and right sides of (7.13) over the random choices involved in sampling $\hat{u}_1, \dots, \hat{u}_T$.

Note also that B^* is defined as the optimal *subset* of k documents, and OPT is the payoff of presenting B^* , without specifying the order in which documents are presented. However, the Ranked Bandits Algorithm learns an order for the documents in addition to identifying a set of documents. In particular, given $k' < k$, $\text{RBA}(k')$ would receive exactly the same feedback as the first k' instances of MAB_i receive when running $\text{RBA}(k)$. Hence any k' sized prefix of the learned ranking also has the same performance bound with respect the appropriate smaller set B'^* .

Finally, it is worth noting that this analysis *cannot* be trivially extended to non-binary payoffs. For instance, one common application with non-binary payoffs is Web search advertising (Edelman et al, 2007). When ranking Web advertisements, the payoff is usually different for each advertisement since advertisement slots are usually auctioned off to advertisers. Each advertiser then pays per click on their advertisement, but the price paid for advertisements in different positions is different. If payoffs are not binary, the greedy algorithm on which RBA is based can obtain a payoff that is a factor of $k - \varepsilon$ below optimal, for any $\varepsilon > 0$.

7.5.3 Analysis of Ranked Explore and Commit

The analysis of the Ranked Explore and Commit (REC) algorithm is analogous to that of the Ranked Bandits algorithm, except that the equivalents of Lemma 7.1 and Theorem 7.1 are only true with high probability after $t_0 = nxk$ time steps of exploration have occurred.

Let B denote the ranking selected by REC.

Lemma 7.2. *Let $x = 2k^2/\epsilon^2 \ln(2k/\delta)$. Assume A_t is drawn i.i.d. from a fixed distribution of user types. For any i , with probability $1 - \delta/k$,*

$$\mathbf{E} \left[\sum_{t=t_0}^T g_i(A_t, B) \right] \geq \frac{1}{k} \mathbf{E} \left[\sum_{t=t_0}^T (G_k(A_t, B^*) - G_{i-1}(A_t, B)) \right] - \frac{\epsilon}{k} T. \quad (7.20)$$

Proof Outline. First note that in this setting, B^* and OPT are defined in expectation over the A_t drawn. For any document, by Hoeffding's inequality, with probability $1 - \delta/2k$ the true payoff of that document explored at rank i is within $\epsilon/2k$ of the observed mean payoff. Hence the document selected at rank i is within ϵ/k of the payoff of the best document available at rank i . Now, the same proof as for Lemma 7.1 applies, although with a different regret $R(T)$. \square

Theorem 7.2. *With probability $(1 - \delta)$, the algorithm's combined payoff after T rounds satisfies:*

$$\mathbf{E} \left[\sum_{t=1}^T G_k(A_t, B) \right] \geq \left(1 - \frac{1}{e}\right) OPT - \epsilon T - nkx \quad (7.21)$$

Proof Outline. Applying Lemma 7.2 for all $i \in \{1, \dots, k\}$, with probability $(1 - k\delta/k) = (1 - \delta)$ the conclusion of the Lemma holds for all i .

Next, an analogous proof as for Theorem 7.1 applies, except replacing $R(T)$ with $\frac{\epsilon}{k}T$ and noting that the regret during the nkx exploration steps is at most 1 for every time step. \square

It is interesting to note that, in contrast to the Ranked Bandits Algorithm, this algorithm can be adapted to the case where clicked documents provide real valued payoffs. The only modification necessary is that documents should always be presented by decreasing payoff value.

7.6 Evaluation

In this section, we evaluate the Ranked Bandits algorithm and Ranked Explore and Commit algorithm, as well as two variants of RBA, with simulations using a user and document model.

We chose a model that produces a user population and document distribution designed to be realistic yet allow us to evaluate the performance of the presented algorithms under different levels of noise in user clicking behavior. Our model first assigns each of 20 users to topics of interest using a Chinese Restaurant Process (Aldous, 1985) with parameter $\theta = 3$. This leads to a mean of 6.5 unique topics, with topic popularity decaying according to a power law. Taking a collection of 50 documents, we then randomly assign as many documents to each topic as there were users assigned to the topic, leading to topics with more users having more documents. We set each document assigned to a topic as relevant to all users assigned to that topic, and all other documents as non relevant. The probabilities of a user clicking on relevant and non-relevant documents are set to constants p_R and p_{NR} respectively.

We tested by drawing one user uniformly from the user population at each time step, and presented this user with the ranking selected by each algorithm, using $k = 5$. We report the average number of time steps when the user clicked on a result, and the average number of time steps when at least one of the presented documents was relevant to the user. All numbers we report are averages over 1,000 algorithm runs.

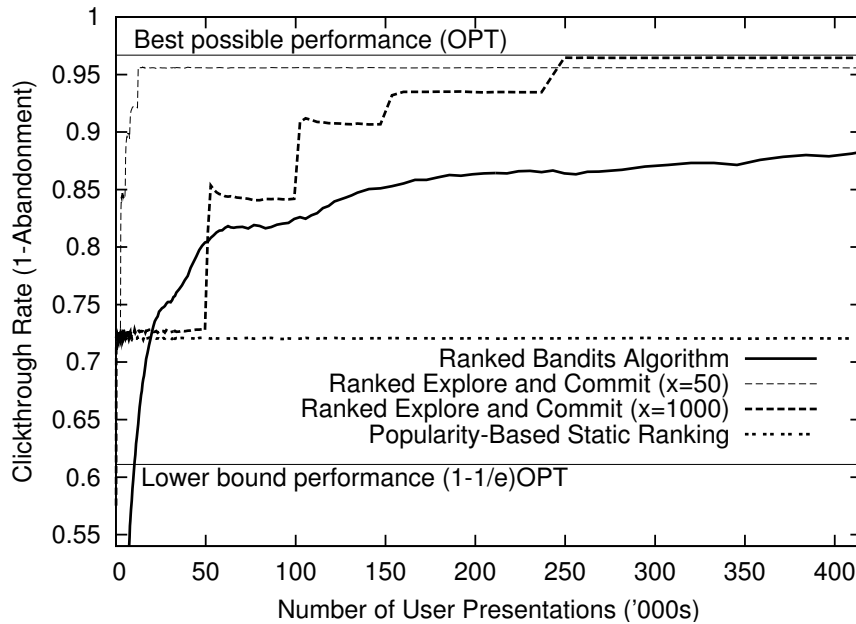


Figure 7.2: Clickthrough rate of the learned ranking as a function of the number of times the ranking was presented to users.

7.6.1 Performance Without Click Noise

We start by evaluating how well the REC and RBA algorithms maximize the clickthrough rate in the simplest case when $p_R = 1$ and $p_{NR} = 0$. We also compare their performance to the clickthrough rate that the same users would generate if presented with documents selected by a static system that orders documents by decreasing true probability of relevance to the users assuming document relevances are independent. Figure 7.2 shows that both REC and RBA perform well above the static baseline and well above the performance guarantee provided by the theoretical results. This is not surprising, as the $(1 - 1/e)OPT$ bound is a worst-case bound. In fact, we see that REC with $x = 1000$ nearly matches the performance of the best possible ranking after finishing its initial exploration phase. We also see that the exploration parameter of REC plays a significant role in the performance, with lower exploration leading

to faster convergence but slightly lower final performance. Note that despite REC performing best here, the ranking learned by REC is fixed after the exploration steps have been performed. If user interests and documents change over time, the performance of REC could fall arbitrarily. In contrast, RBA is guaranteed to remain near or above the $(1 - 1/e)OPT$ bound.

7.6.2 Effect of Click Noise

In Figure 7.2, the clickthrough rate and fraction of users who found a relevant document in the top k positions is identical (since users click if and only if they are presented with a relevant document). In contrast, Figure 7.3 shows how the fraction of users who find a relevant document decays as the probability of a user clicking becomes noisier. The figure presents the performance lines for REC and RBA across a range of click probabilities, from $(p_R = 1, p_{NR} = 0)$ to $(p_R = 0.7, p_{NR} = 0.3)$. We see that both algorithms decay gracefully: as the clicks become noisier, the fraction of users presented with a relevant documents decays slowly.

7.6.3 Practical Considerations

Despite the theoretical results shown earlier, it would be surprising if an algorithm designed for the worst case had best average case performance. Figure 7.4 shows the clickthrough rate (which the algorithms optimize), and fraction of users who find relevant documents (which is of more interest to information retrieval practitioners), for variants building on the insights of the ranked bandits

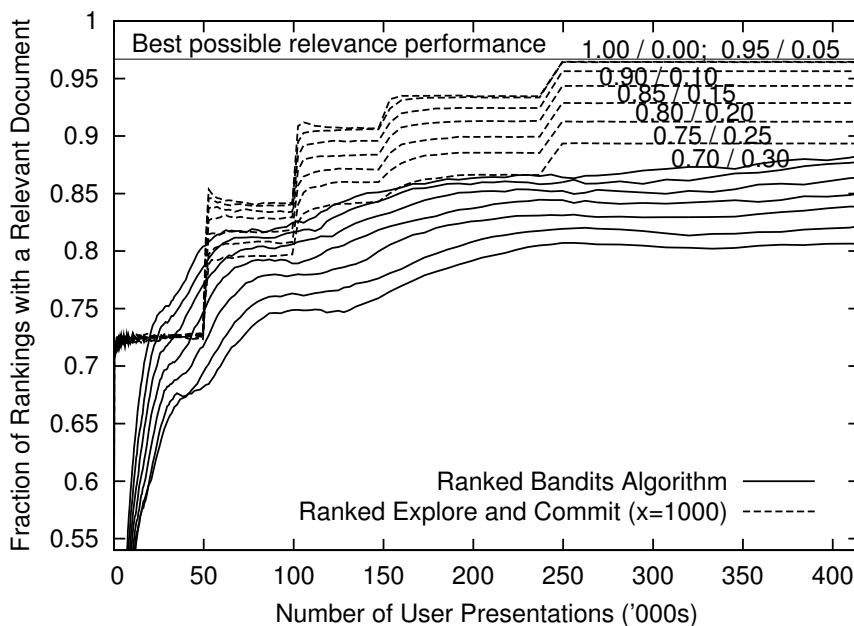


Figure 7.3: Effect of noise in clicking behavior on the quality of the learned ranking.

idea. Specifically, two variants of RBA that have the best performance we could obtain in our simulation are shown. The first variant uses a modified EXP3 arm selection algorithm. In particular, we found that instead of updating weights using the standard EXP3 scheme, $w_j(t+1) = w_j(t) \exp(\gamma \hat{x}_j / K)$, updating using a larger step size $\eta \gamma \hat{x}_j / K$ with $\eta = 7$ resulted in faster performance improvement. The second variant uses a modified UCB1-based multi-armed bandit algorithm (Auer et al, 2002a). In this case, we found that by modifying the confidence bound used in UCB1 improved performance. In particular, instead of computing the confidence interval $\sqrt{2 \ln n / n_j}$, we used $1 / \sqrt{n_j}$.

We found that using a UCB1-based multi-armed bandit algorithm in place of EXP3 improves the performance of RBA substantially when user interests are static. Note however, that UCB1 does not satisfy the constraints presented in Section 7.4.2 because it assumes rewards are identically distributed over

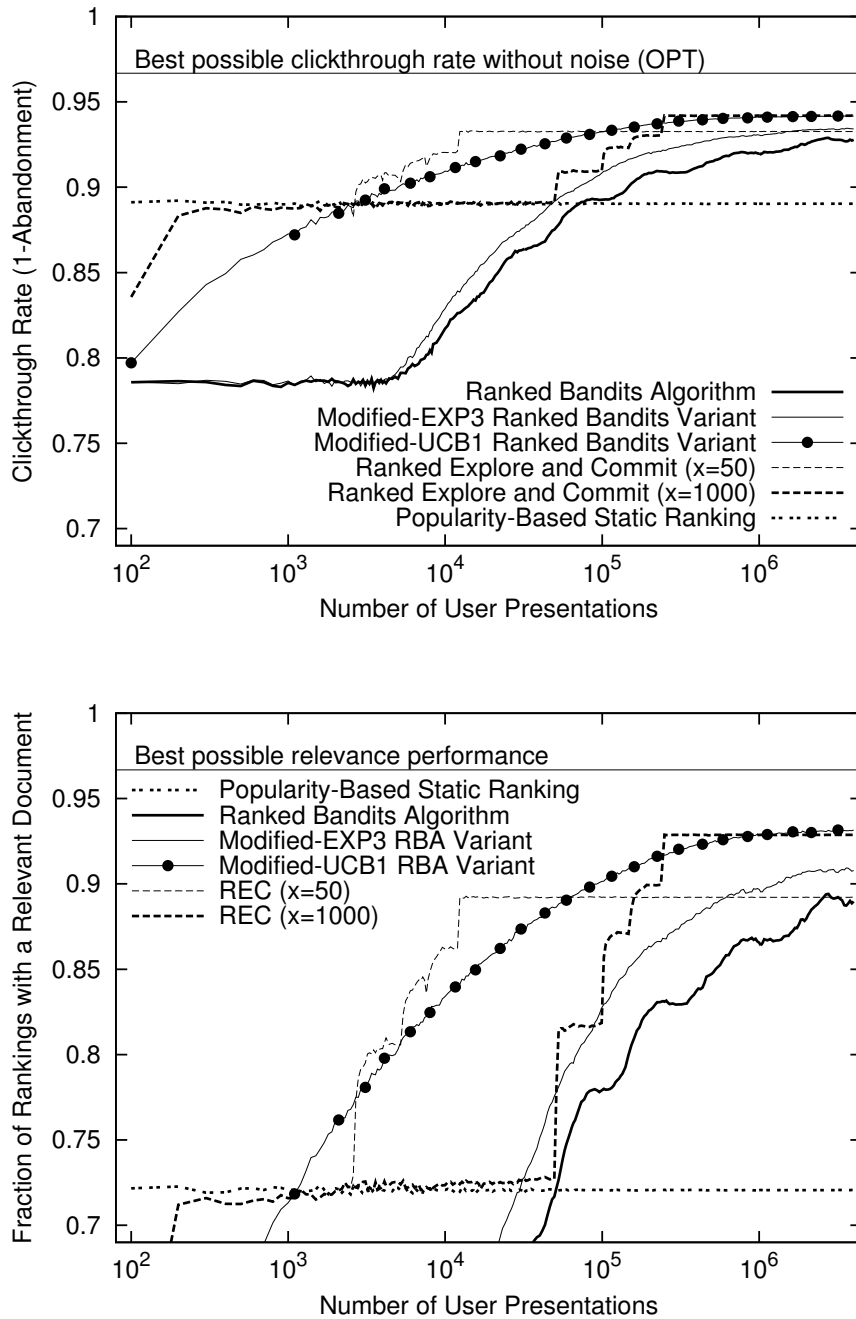


Figure 7.4: Performance of RBA, REC and two RBA variants. We see that in practical settings, variants with weaker theoretical guarantees may achieve higher performance. These results use realistic noise settings $p_R = 0.8$, $p_{NR} = 0.2$.

time, an assumption violated in our setting when changes in the documents presented above rank i alter the reward distribution at rank i . Moreover, our modified confidence interval no longer guarantees optimality of UCB1 even in the standard Multi-Armed Bandit setting. Nevertheless, we see that this modification substantially improves the performance of RBA. We expect such an algorithm to perform best when few documents are prone to radical shifts in popularity.

7.7 Summary

This chapter has presented a new formulation of the learning to rank problem that explicitly takes into account the relevance of different documents being interdependent. We presented, analyzed and evaluated two algorithms and two variants for this learning setting. We have shown that the learning problem can be solved in a theoretically sound manner, and have provided evidence from simulations that our algorithms are likely to perform reasonably in practice.

CHAPTER 8

CONCLUSIONS AND OPEN QUESTIONS

This thesis has studied the question of how to improve interactive information ranking systems by observing the natural behavior of users in response to information with which they are presented. We now summarize the four key contributions presented here, and describe three important directions that remain open for further study.

8.1 Thesis Conclusions

Many of the key tools made available through the Internet rely on presenting ranked lists of information to users. These rankings help users find the information they need, be it about books, movies, websites, people, or any of a tremendous array of items about which information is available online. Throughout this thesis, we have considered the question of how the natural interactions of users with such ranking systems provide information that can be used to improve them automatically using machine learning techniques. The records of these actions, termed *implicit feedback*, are particularly valuable as they provide a real time view of the needs users actually have, and also tell us when needs are not being satisfied.

After setting the stage for the thesis in the Introduction, we first presented an overview of user behavior both offline and online in Chapter 2. In particular, this chapter described how users act in a Web setting, and thus which effects we need to take care to consider when interpreting user behavior as implicit feedback.

Following this, we presented an overview of some of the many approaches for learning to rank, and for measuring the performance of these algorithms in the third chapter. The four key contributions of this thesis were presented next:

I. Many Web search engine users pose a sequence of queries during a single visit, in what we call *query chains*. We showed that these query reformulations provide valuable information about user needs. In particular, reformulations can describe when needs are being unmet by a current system. We showed how to take reformulations and clicks recorded on a Web search engine, and transform them into relative relevance judgments that can then be used to learn an improved ranking function. This improvement was demonstrated on a real search system implemented for the Cornell University library.

II. We described a specific difficulty with the relevance judgments obtained from clickthrough data, namely the problem of *presentation bias*. We showed that by using an algorithm called *FairPairs*, presentation bias can be avoided. This algorithm collects pairwise relevance judgments that provably do not suffer from presentation bias, under reasonable assumptions. The presentation bias effect, and FairPairs algorithm, were both demonstrated on a real search system that provided search functionality on the arXiv e-print archive.

III. We discussed the difficulty of *evaluation bias*: users pay substantially more attention to top ranked results than lower ranked results. When collecting training data for improving the performance of interactive information ranking systems, this bias means that much less information is obtained about results that are not initially presented at high ranks. We presented a new approach to ranking, considering the training data that will be obtained when presenting results to users rather than simply ordering results by presumed relevance. A

number of principled search strategies for choosing which rankings to display were evaluated. This evaluation, using a simulation of user behavior, showed that two of the strategies proposed lead to much faster learning by ranking systems.

IV. We presented a new formulation of learning to rank that optimizes a *user based* performance measure, specifically *abandonment*. This formulation also encodes interdependencies between the relevance of individual results. In particular, if two of the ranked items are essentially redundant, the benefit from presenting both is often only marginally higher than the benefit from presenting just one. Moreover, the algorithms presented for this problem also address *user diversity*. In a Web search setting, user diversity this is manifested by different users interpreting the same query differently. One of the algorithms presented obtains the best possible polynomial time approximation factor to minimizing user abandonment using a worst-case bound.

Taken together, these results have shown that implicit feedback is a valuable source of data for improving interactive information ranking systems. While biases exist in user behavior, we have shown that careful design of data collection and data interpretation techniques can avoid, or compensate for, these difficulties. We presented a number of new learning algorithms that use implicit feedback. Our evaluations showed that the algorithms are effective in practice, as well as having desirable theoretical properties.

8.2 Open Questions and Future Directions

Many important questions relating to learning to rank from implicit feedback, however, remain unanswered. In this section, We briefly consider three of them, and describe preliminary results for the first two.

8.2.1 Personalization

Given the large quantities of implicit feedback that can be collected, can this data be used to further personalize the current generation of interactive information ranking systems? For instance, personalizing Web search is one important direction for more research. It asks – if different users understand the same query differently, why should they all be presented with the same results? A number of researchers have studied various aspects of this question, for example, Sugiyama et al. (2004); Teevan et al. (2005b); Zhang et al. (2002); Ziegler et al. (2005). Similarly, a number of personalizing commercial Web search engines are available.

One increasingly common approach to personalization is to provide a user profile to search engines, which can then use this profile to bias search results toward the user’s interests¹⁴. However, this requires the search engine to perform personalization at additional computational expense, and requires that the users trust the search engine with the personal information encoded in their profiles. We now present preliminary work, published with more details in (Radlinski & Dumais, 2006), that instead describes a client-side approach for diversifying

¹⁴For example, the Google search engine allows users to log in, after which a profile of their searches and clicks is built implicitly.

Web search results. At a high level, client-side personalization proposes that in response to a query, the user's browser is provided with a number of search results (for instance, 100). The search results can then be ranked by the client to place more promising results for the user at higher ranks.

The primary difficulty with client-side ranking of search results is that the client can only rank the limited number of top results that are made available to it. While this may allow effective personalization when Web pages of particular interest to the user are present in the set, it cannot be effective if all available results are similar and of less interest to the user.

Anagnostopoulos et al. (2005) proposed a method to sample search results to obtain a heterogeneous sample of the search results for a query. An alternative that we study is to use query chains to understand the variety of user intents for a query, and improve the effectiveness of client-side ranking. The main insight in our approach is that by observing how large numbers of users reformulate their queries, we can see which kinds of results tend to be missing from the top of search results, from the users' perspective. For example, when studying logs from a large Web search engine, we observed that the query "windows" is often followed by specializations such as "windows xp" or "house windows". This suggests that if we want to personalize results for a user who issued the query "windows", we may also want to consider results from both of these reformulations. Analyzing query chains can thus allow interesting diversity to be added to query results.

I. Diversifying using Query Chains

Suppose that we want to personalize search by reranking 100 results client-side. Given a query q , we propose to generate a set of k related queries $Q(q)$. We then take $\frac{100}{k+1}$ results from each query in $Q(q)$, and from q . This gives \mathcal{C} , a corpus of 100 results to rerank. In our evaluation, we will use $k \in \{0, 2, 4, 9, 19\}$. When $k=0$, the top 100 results from the original query are considered for reranking, and when $k=19$, the top 5 results from q and from 19 reformulations are considered.

To obtain query reformulations, we analyzed a large sample of the query logs from a popular Web search engine over about 6 weeks. For each query q_i we measured n_i , the number of times the query was observed, and p_i , the empirical probability that q_i was followed by any other query from the same IP address within a thirty minute time window. For a pair of queries (q_i, q_j) , let n_{ij} be the number of times q_i was followed by q_j . $p_{ij} = \frac{n_{ij}}{n_i}$ is the empirical probability of q_i being followed by q_j . p_{ij}^* is the related symmetric measure, $p_{ij}^* = \sqrt{p_{ij}p_{ji}}$.

We developed three methods for generating $Q(q)$. The Most Frequent (MF) method sets $Q(q_i)$ to the queries q_j with highest n_{ij} . These are the queries that most often follow q_i . The Maximum Result Variety (MRV) method greedily selects queries that are both frequent reformulations (using p_{ij}) and different from other queries that have already been selected (using p_{jk}^*). We used a weighted combination of these two factors, $\operatorname{argmax}_{q_j} (\lambda p_{ij} - (1 - \lambda) \max_{q_k \in R(q_i)} p_{jk}^*)$, with $\lambda = 0.5$. MRV is motivated by the MMR approach of Carbonell and Goldstein (1998) that is described on page 148 of this thesis. It aims to select a set of queries that are related to q_i yet different from each other. Finally, the Most Satisfied (MS) method sets $Q(q_i)$ as the set of queries q_j with minimum p_j that also satisfy

$p_{ij} > 0.001$ and $n_{ij} \geq 2$. This method finds queries that tend not to be further reformulated yet occur with some minimum frequency.

II. Diversity Evaluation

Let $match(d_i, u)$ measure how well document d_i matches the interests of the user u . We can measure $match(d_i, u)$ using the relevance-feedback approach developed by Teevan et al. (2005b). They proposed weighting words using relevance information obtained from a local representation of users' interests:

$$w_t = \log \frac{(r_t + 0.5)(N - n_t + 0.5)}{(n_t + 0.5)(R - r_t + 0.5)}, \quad (8.1)$$

where N is the number of documents in the corpus of documents being ranked, R is the number for which we have relevance feedback, and n_t and r_t are the number of documents in N and R that contain t . N and n_t were computed from a sample of 1.5 billion Web pages. R and r_t were computed for each user using a full text index of the files, emails and Web pages on their computer hard drives to represent their interests (Teevan et al., 2005b). We computed w_t for pairs of adjacent words (bigrams), using them to compute the match function:

$$match(d_i, u) = \sum_{t_i, t_j \in d_i} w_{t_i, t_j} \quad (8.2)$$

The maximum match in \mathcal{C} , $diversity(\mathcal{C}) = \max_{d_i \in \mathcal{C}} match(d_i, u)$, reflects the extent to which at least one result is very similar to a user's interests. We used the average value of $diversity(\mathcal{C})$ across all users as a measure of diversity for each method.

III. Preliminary Results

We evaluated this approach for 33 volunteers. Figure 8.1 shows the bigram match results for the diversification methods and five values of k .

The evaluation was performed on two types of queries. The lower three curves show the results for a set of 30 fixed queries chosen from the search engine log. The queries varied in frequency, topic, and typical reformulation patterns. The upper two curves show the results for the most recent queries in each user's browser history, averaging 76 queries per user. The MS method did not generate enough reformulations for some of the user-specific queries so we omit it. Computing the F-statistic, we see that the main effect of query type (fixed, user) is reliable ($F(1, 32) = 4.82, p = 0.022$). This means that the match score for queries of interest to the user is higher than the match score for the fixed set of general queries. The main effect of diversification method is marginally reliable ($F(1, 32) = 3.30, p = 0.079$), with MRV leading to somewhat higher diversity scores. The main effect of k is reliable ($F(4, 128) = 3.82, p = 0.006$), showing that diversity scores increase as the number of reformulations considered increases. Interestingly, for the MF method the first few reformulations *reduce* the result diversity. This suggests that the most frequent reformulations are not very different in topic from the original query. Even with this small initial dip, the linear correlation between k and diversity score is strong and significant ($r = 0.90, p = 0.037$).

In all, these results suggest that query reformulations can be used to improve personalized Web search in ways other than through pairwise relevance judgments about pairs of documents.

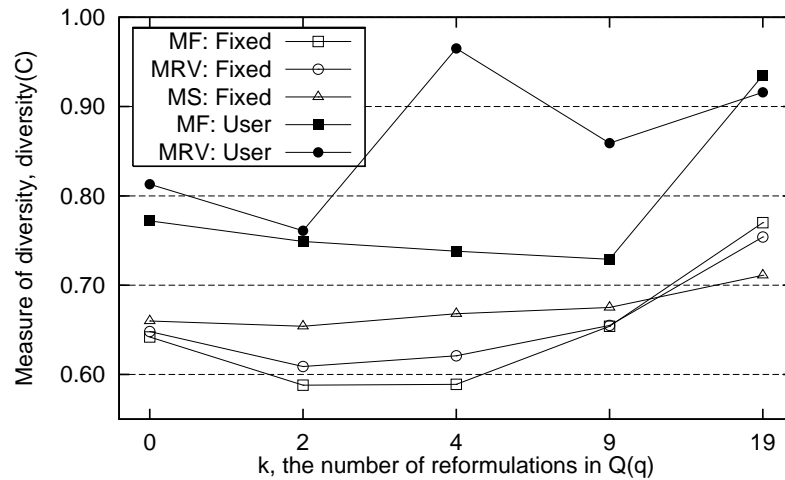


Figure 8.1: Evaluation of MF, MRV and MS diversity methods on a fixed query set (Fixed), as well as on queries taken from users' Web browser cache (User).

8.2.2 Malicious Noise

A second important direction for future study involves malicious noise in click-through data. Once implicit feedback becomes widely used for improving ranking systems on the Internet, commercial incentives will increase malicious behavior. The uncertainty is in the extent to which this malicious behavior will affect the usefulness of implicit feedback.

One possible approach to mitigate the effect of such malicious behavior, which in the context of clickthrough data we term *click-spam*, is to partition users into groups. Implicit feedback generated by each group can then be used only to affect rankings seen by that group. This can reduce the economic incentives of click-spam, as we will now show in one specific setting by describing part of a preliminary study published in longer form in (Radlinski, 2007).

I. Modeling Search Preferences

For many Web search queries, we can model the ranking problem as having a large number of competitors C_i providing commercial services (such as selling sporting goods, books or pharmaceuticals). Search engine users enter a query, and in response are presented with an ordered list of competitors. For such commercial searches, as for all searches, the user is much more likely to visit highly ranked websites than those ranked lower. Hence, ensuring that the ranking is based on some reasonable measure (for example reliability, price or geographic proximity to the user) is important.

To make it possible to study such a system theoretically, consider a setting where we have a single query and some set of m competitors C_1 through C_m . Further, suppose our algorithm for learning to rank converts clickthrough data into simple votes for each competitor C_i . A sensible approach for ranking the competitors would then be to rank them by decreasing votes. To eliminate the obvious difficulties if one user were able to provide more votes than other users, we assume each user of the search engine can provide one vote. For example, the IP address or a cookie could be used as a proxy for user identity. In this setting, a spammer can produce click-spam by taking other users' votes. In practice, this would involve fraudulent clicks caused by compromised systems or by paid users. We will call all users or systems creating malicious click data *spam hosts*. In particular, it could be in the interest of the lower ranked competitor to produce click-spam if the cost of obtaining enough spam hosts to be ranked higher is less than the financial benefit of being ranked higher.

II. Ranking Utility

Let the fraction of users whose preferences are satisfied by the ranking they are presented with be called the *user utility*. This is the measure that we are interested in optimizing. However, there is also a utility of the ranking from the perspective of the competitors C_i , as each possible ranking has a specific value to each competitor being ranked. We call this utility the *competitor utility*. We now formalize these utilities.

Assume that there are n users who are interested in buying the products that C_1 through C_m compete to sell. Let p^{C_i} be the percentage of users who prefer competitor C_i . Say we divide the population of users into k partitions, P_1 through P_k with $k \ll n$ and with each partition large enough that we do not need to consider small sample effects. The percentage of users in partition P_i is p_i , with $p_i^{C_j}$ being the percentage of all users who are in partition i and prefer C_j . For example if there are two competitors, and partition P_1 includes 10% of the users where three quarters of them prefer C_1 then $p_1 = 0.1$, $p_1^{C_1} = 0.075$ and $p_1^{C_2} = 0.025$.

We define the user utility of a partitioning as the fraction of users whose preferred competitor is ranked first if the learned ranking takes a majority vote in each partition. Without any click-spam present, this is

$$util^u(P) = \sum_{i=1}^k \max_j p_i^{C_j} \quad (8.3)$$

When there is click-spam, a spammer may flip which competitor is ranked highest in any or all of the partitions. We define a spammer by η , the fraction of all hosts that the spammer turns into spam hosts. We assume spammers can

obtain any number of spam hosts, but restrict ourselves to spammers that do not know which partition a particular host is in before it is compromised. This means that the additional number of votes that an η -spammer trying to promote C_s can gain is proportional to $\eta(1 - p^{C_s})$, since the remaining compromised hosts will already be voting for C_s . Within a particular partition P_i , an η -spammer could change the number of votes for C_s from $p_i^{C_s}$ to $p_{i,\eta}^{C_s} = p_i^{C_s} + \eta(p_i - p_i^{C_s})$, gaining η of the votes in P_i not already cast for C_s . Substituting this modification for click-spam into the user utility, we can write the user utility given an η -spammer promoting C_s as

$$util^u(P) = \sum_i \begin{cases} p_i^{C_s} & \text{if } p_{i,\eta}^{C_s} > \max_{j \neq s} (1 - \eta)p_i^{C_j} \\ \max_{j \neq s} p_i^{C_j} & \text{otherwise} \end{cases} \quad (8.4)$$

A primary consideration for most commercial websites is how many visitors their website attracts. Motivated by the approximately Zipfian form of the number of clicks on search results as a function of rank (for example, see Agichtein et al. (2006)), we define the utility of a ranking from C_i 's perspective as a simple approximation of the number of clicks that C_i might receive: the reciprocal rank of C_i averaged across all users. We define our pricing units such that the difference in utility between a competitor between being ranked first and second is one unit. Without any spam present, the competitor utility of a partitioning P to competitor C_s is

$$util^c(P, C_s) = 2n \sum_{i=1}^k p_i \left(1 + \left| \left\{ j : p_i^{C_j} > p_i^{C_s} \right\} \right| \right)^{-1} \quad (8.5)$$

when there are n users in the population (the last term is the reciprocal rank of C_s in P_i). If C_s hires an η -spammer, the utility becomes

$$2n \sum_{i=1}^k p_i \left(1 + \left| \left\{ j : (1 - \eta)p_i^{C_j} > p_i^{C_s} + \eta(p_i - p_i^{C_s}) \right\} \right| \right)^{-1} - cost(\eta\text{-spammer}). \quad (8.6)$$

To continue the analysis, we must next assume a cost per spam host. Intuitively, compromising different hosts on the Internet has different costs. Some fraction of hosts can be compromised cheaply (for example, old systems with out-of-date software) while others (for example, well maintained hosts behind a firewall) would be expensive to compromise. This suggests that the cost of obtaining spam-hosts should be superlinear in the number of hosts required. We model the cost of compromising a fraction η of all n hosts (equating each user to one host) with a simple quadratic function:

$$\text{cost}(\eta) = a\eta^2 n, \quad (8.7)$$

where a is a constant. To estimate a conservative value of a , suppose that 20% of hosts on the Internet can become spam hosts for an average cost of 1 per machine, i.e. the utility of being ranked at the top rather than second in a ranking for one user. In this case, $a = 25$.

III. The Economics of Click-Spam

We can now ask the following questions: Given a partitioning P , when will it be profitable for a competitor to hire a spammer? What will be the impact on user utility? For the equations to be manageable, in this section we limit ourselves to the two competitor case, calling the first competitor A and the second B .

Say we are given a partitioning P . Let the partitions P_i be ordered such that $\forall i \in \{1, \dots, k-1\}$, $p_i^A/p_i \leq p_{i+1}^A/p_{i+1}$. In words, the first partition has the largest fractional vote for B , followed by the second partition and so forth. This means that as η increases, an η -spammer would take over partitions in order, and guarantees that there is some t such that $i \leq t \Leftrightarrow p_i^B > p_i^A$.

Given this ordering of partitions, it will be in B 's interest to hire a spammer if and only if the utility of taking over the j next partitions after P_t minus the cost of taking over the j^{th} partition is positive. The fraction η_j of machines that must be compromised to flip the outcome in partition P_j is simply half the margin by which A wins divided by the fraction of spam-hosts that were voting for A when taken over:

$$\eta_j = \frac{1}{2} (p_j^A/p_j - p_j^B/p_j) \frac{1}{p_j^A/p_j} = 1 - \frac{p_j}{2p_j^A}. \quad (8.8)$$

Thus B will hire a spammer if and only if

$$\exists j \text{ s.t. } \left(\sum_{i=t+1}^{t+j} p_i \right) - a \left(1 - \frac{p_{t+j}}{2p_{t+j}^A} \right)^2 > 0. \quad (8.9)$$

The first term is the value to the spammer of taking over partitions $t + 1$ through $t + j$ (note that A initially has more votes in each of them). The second term is the cost of taking over the $t + j^{\text{th}}$ partition. Since this partition has the largest imbalance, when it is taken over by the spammer, the spammer has also taken over all partitions with lower index.

IV. Practical Partitioning

The above condition tells us if spam is in the economic interest of competitor B in the case of two competitors and any number of partitions. We now turn to the question of whether partitioning would be useful in practice, restricting ourselves to one simple setting.

Assume that we can find a feature $f(u_i)$ for each user u_i that is weakly indicative of the preferences of the user. Say that users who prefer competitor A have $f(u_i)$ normally distributed with $f(u_i) \sim \mathcal{N}(\mu_A, \sigma_A^2)$, and similarly users

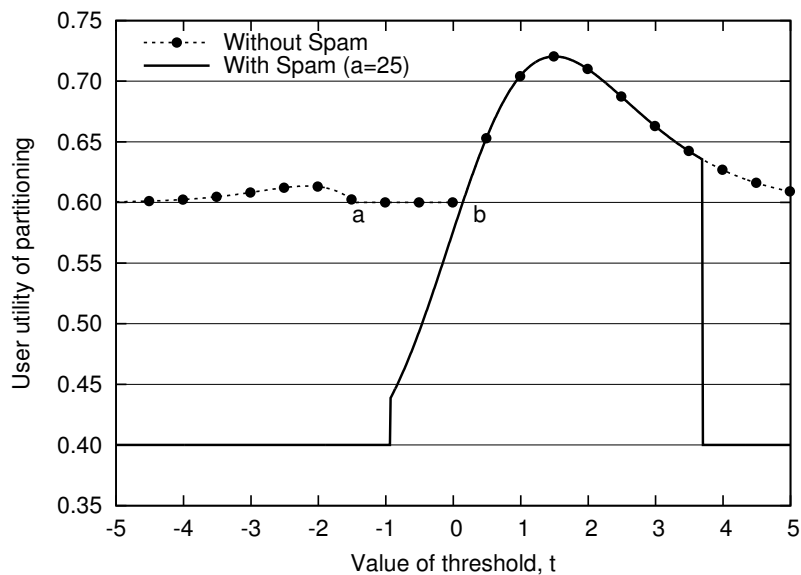


Figure 8.2: User utility as a function of threshold when $\mu_B = 1$ and $\sigma_B = 2$. $\mu_A = 0$, $\sigma_A = 1$ and $a = 25$.

who prefer competitor B have $f(u_i) \sim \mathcal{N}(\mu_B, \sigma_B^2)$. Then at any $x = f(u)$, the probability that a user prefers A is

$$p^A(x) = \frac{p^A \mathcal{N}(x; \mu_A, \sigma_A^2)}{p^A \mathcal{N}(x; \mu_A, \sigma_A^2) + p^B \mathcal{N}(x; \mu_B, \sigma_B^2)}, \quad (8.10)$$

where $\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-(x - \mu)^2/2\sigma^2)$. Say we restrict ourselves to partitionings that pick a threshold t and assign all users with $f(u_i) \leq t$ to partition P_1 and all others to partition P_2 . We would like to find an optimal threshold t that maximizes user utility even when there is click spam. In this preliminary work, we study the problem empirically. In the following, we take $p^A = 0.6$, $p^B = 0.4$ and pick a scale such that $\mu_A = 0$ and $\sigma_A = 1$. We also restrict ourselves to the case where any click-spam favors competitor B .

Figure 8.2 shows the form of the user utility as the threshold t changes when the larger population (A) has $f(u)$ values with smaller variance than the smaller population (B). First consider the dotted line. This is the user utility as function of threshold t when click-spam is not present. Note that without partitioning,

the user utility would be 0.6, since 60% of the population belong to the majority class (A). By partitioning, we can increase the user utility. Since $\sigma_B > \sigma_A$, B has a heavier tail than A . Hence there exists a threshold b above which more users prefer B than A . This point is marked on the figure. If $t > b$, the ranking shown to users in partition P_2 ranks competitor B above A . The form of the plot comes about since as t gets large, the absolute number of users in P_2 becomes small so the gain in user utility becomes small. On the other hand, if t is near b , there are still many users who prefer A in P_2 and this reduces the overall user utility. We see a similar effect at $t < a$.

The solid line in Figure 8.2 shows the effect of click-spam, when there is a spammer who is hired by B . Note that in the unpartitioned case, the cost of spam is such that it is in B 's interest to hire the spammer, thus reducing the overall user utility to 0.4 (since 40% of the user population actually prefers B). We see that with the partitioning, the choice of threshold t has a dramatic impact on the user utility when click-spam is present. At the extremes, when the threshold is either very small or very large, we essentially have one small partition dominated by users who prefer B and one large partition dominated by users who prefer A . When the large partition is large enough, despite the fraction of users who prefer A being somewhat increased, click-spam is still economical. In the third regime, when $a \leq t \leq b$, the partitioning creates two partitions where A dominates in both but by different margins. For some thresholds it is only in B 's interest to dominate just one of the partitions, resulting in a user utility between 0.4 and 0.6, while at others B dominates both.

To summarize, these first results suggest that partitioning by thresholding on a weak feature may make learning from implicit feedback less prone to

click-spam. This suggests further research along this line of reasoning may be worthwhile. Separately, the sensitivity of practical learning algorithms using practical interpretations of implicit feedback need to be addressed. While we have also published a first study of the sensitivity of Ranking SVMs to synthetically generated click noise when learning from inferred pairwise relevance judgments in (Radlinski & Joachims, 2005a), we did not consider the possibility of malicious noise. In all, malicious noise is likely to become a future component of implicit feedback, and its effects as well as countermeasures against it need to be studied.

8.2.3 Scalability and Generalizability

A third important question to address is how to permit algorithms such as those discussed in this thesis to generalize and be practical at larger scales. In particular, two problems need to be addressed.

First, the amount of implicit feedback collected by a commercial scale search engine is typically too large to be practically usable by standard machine learning algorithms. For instance, the preferences generated from observing user behavior on the Cornell University library search engine over a few months would likely be collected by a large commercial search engine in seconds. Data collected even over one day would probably be too massive to provide to a Ranking SVM. This raises the question whether alternative algorithms should be used, whether the data should be subsampled, whether specific ranking functions should be learned for different segments of the user population, or whether some other approach is necessary.

Second, a number of the theoretically appealing algorithms presented earlier in this thesis estimate model parameters that scale as the number of documents times the number of distinct queries. In particular, the strategies for collecting more useful training data (presented in Chapter 6) and algorithms for learning diverse rankings (presented in Chapter 7) both have this property. On large scales, this means that the models would be less practical. The common approach to avoid learning too many parameters when simply learning to predict document relevance, is to encode document relevance as a function of a relatively small number of features. The parameters of this relevance function can then be learned, instead of learning a relevance score per document. One possible approach to improve the scalability of the aforementioned algorithms would be to also learn a second function parameterized with features that encode the uncertainty in current relevance estimates, or the redundancy between documents. This would allow us to use implicit feedback to learn the parameters of this function instead of learning parameters for each (query, document) pair.

BIBLIOGRAPHY

- Agichtein, E., Brill, E., Dumais, S., & Ragno, R. (2006). Learning user interaction models for prediction web search results preferences. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 3–10). ACM.
- Agichtein, E., & Zheng, Z. (2006). Identifying “best bet” web search results by mining past user behavior. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 902–908). ACM.
- Aldous, D. J. (1985). Exchangeability and related topics. *École d’Été de Probabilités de Saint-Flour XIII* (pp. 1–198).
- Allan, J., Carterette, B., & Lewis, J. (2005). When will information retrieval be “good enough”? *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 433–440).
- Anagnostopoulos, A., Broder, A. Z., & Carmel, D. (2005). Sampling search engine results. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 245–256). Chiba, Japan.
- Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47, 235–256.
- Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002b). The non-stochastic multi-armed bandit problem. *SIAM Journal of Computing*, 32, 48–77.
- Bartell, B., & Cottrell, G. W. (1995). Learning to retrieve information. *Proceedings of the Swedish Conference on Connectionism*.
- Bartell, B. T., Cottrell, G. W., & Belew, R. K. (1994). Automatic combination of multiple ranked retrieval systems. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 173–181). Springer-Verlag.

- Basilico, J., & Hofmann, T. (2004). Unifying collaborative and content-based filtering. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 221–228). ACM.
- Beeferman, D., & Berger, A. (2000). Agglomerative clustering of a search engine query log. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 407–416).
- Boyan, J., Freitag, D., & Joachims, T. (1996). A machine learning architecture for optimizing web search engines. *AAAI Workshop on Internet Based Information Systems*.
- Bradley, R. A. (1976). Science, statistics, and paired comparisons. *Biometrics*, 32, 213–232.
- Bradley, R. A., & Terry, M. E. (1952). The rank analysis of incomplete block designs. 1. the method of paired comparisons. *Biometrika*, 39, 324–345.
- Branting, L. K., & Broos, P. S. (1997). Automated acquisition of user preferences. *International Journal of Human-Computer Studies*, 46, 55–77.
- Brinker, K. (2004). Active learning of label ranking functions. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 129–136).
- Broder, A. (2002). A taxonomy of web search. *SIGIR Forum*, 26, 3–10.
- Brumby, D. P., & Howes, A. (2004). Good enough but i'll just check: Web-page search as attentional refocusing. *Proceedings of 6th International Conference on Cognitive Modeling* (pp. 46–51).
- Buckley, C., & Voorhees, E. M. (2004). Retrieval evaluation with incomplete information. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 25–32).

- Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. (2005). Learning to rank using gradient descent. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 89–96).
- Burges, C. J. C., Ragno, R., & Le, Q. V. (2006). Learning to rank with nonsmooth cost functions. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 193–200). MIT Press.
- Carbonell, J., & Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 335–336).
- Carterette, B., Bennett, P. N., Chickering, D. M., & Dumais, S. T. (2008). Here or there: Preference judgements for relevance. *Proceedings of European Conference on Information Retrieval (ECIR)* (pp. 16–27).
- Carterette, B., & Jones, R. (2007). Evaluating search engines by modeling the relationship between relevance and clicks. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 217–224).
- Caruana, R., Baluja, S., & Mitchell, T. (1995). Using the future to “sort out” the present: Rankprop and multitask learning for medical risk prediction. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 959–965).
- Chajewska, U., Koller, D., & Parr, R. (2000). Making rational decisions using adaptive utility estimation. *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (pp. 363–369).

- Chen, H., & Karger, D. R. (2006). Less is more: Probabilistic models for retrieving fewer relevant documents. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 429–436).
- Chu, W., & Ghahramani, Z. (2005a). Extensions of gaussian processes for ranking: Semi-supervised and active learning. *Proceedings of the NIPS 2005 Workshop on Learning to Rank* (pp. 29–34).
- Chu, W., & Ghahramani, Z. (2005b). Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6, 1019–1041.
- Chu, W., & Ghahramani, Z. (2005c). Preference learning with gaussian processes. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 137–144).
- Chu, W., & Keerthi, S. S. (2005). New approaches to support vector ordinal regression. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 145–152).
- Cohen, W. W., Shapire, R. E., & Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, 10, 243–270.
- Cooper, W. S. (1968). Expected search length: A single measure of retrieval effectiveness based on the weak ordering action of retrieval systems. *Journal of the American Society for Information Science*, 19, 30–41.
- Coupey, E., Irwin, J. R., & Payne, J. W. (1998). Product category familiarity and preference construction. *Journal of Consumer Research*, 24, 459–468.
- Crammer, K., & Singer, Y. (2001). Pranking with ranking. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 641–647).

- Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.
- Cucerzan, S., & Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 293–300).
- Cui, H., Wen, J.-R., Nie, J.-Y., & Ma, W.-Y. (2002). Probabilistic query expansion using query logs. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 325–332).
- Currim, I. S., & Sarin, R. K. (1984). A comparative evaluation of multiattribute consumer preference model. *Management Science*, 30, 543–561.
- Davidson, R. R., & Beaver, R. J. (1977). On extending the bradley-terry model to incorporate within-pair order effects. *Biometrics*, 33, 693–702.
- De Beer, J., & Moens, M.-F. (2006). Rpref - a generalization of bpref towards graded relevance judgments. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 637–638).
- Dekel, O., Manning, C. D., & Singer, Y. (2003). Log-linear models for label ranking. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 497–504).
- Diaconis, P., & Graham, R. (1977). Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B*, 39, 262–268.
- Dittrich, R., Hatzinger, R., & Katzenbeisser, W. (1998). Modelling the effect of subject-specific covariances in paired comparison studies with an application to university rankings. *Applied Statistics*, 47, 511–525.

- Dou, Z., Song, R., & Wen, J.-R. (2007). A large-scale evaluation and analysis of personalized search strategies. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 581–590).
- Dupret, G., Murdock, V., & Piwowarski, B. (2007). Web search engine evaluation using clickthrough data and a user model. *Proceedings of the Workshop on Query Log Analysis at WWW*.
- Edelman, B., Ostrovsky, M., & Schwarz, M. (2007). Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 9, 242–259.
- Eliashberg, J. (1980). Consumer preference judgments: An exposition with empirical applications. *Management Science*, 26, 60–77.
- Fagin, R., Kumar, R., & Sivakumar, D. (2003). Comparing top k lists. *SIAM Journal of Discrete Math*, 17, 134–160.
- Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Science (TOIS)*, 23, 147–168.
- Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 170–178).
- Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4, 933–969. Originally appears at ICML 1998.
- Fuhr, N. (1989). Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Science (TOIS)*, 7, 183–204.

- Furnas, G. (1985). Experience with an adaptive indexing scheme. *Proceedings of Conference on Human Factors in Computing Systems (CHI)* (pp. 131–135). ACM Press.
- Glickman, M. E. (1999). Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, *48*, 377–394.
- Glickman, M. E. (2008). Bayesian locally-optimal design of knockout tournaments. *Journal of Statistical Planning and Inference*, *138*, 2117–2127.
- Glickman, M. E., & Jensen, S. T. (2005). Adaptive paired comparison design. *Journal of Statistical Planning and Inference*, *127*, 279–293.
- Granka, L., Joachims, T., & Gay, G. (2004). Eye-tracking analysis of user behavior in www search. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 478–479).
- Harrington, E. F. (2003). Online ranking / collaborative filtering using the perceptron algorithm. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 250–257).
- Hawking, D., & Craswell, N. (2001). Overview of the TREC-2001 web track.
- Herbert, S. (1957). A behavioral model of rational choice. In *Models of man, social and rational: Mathematics essays on rational human behavior in a social setting*. Wiley.
- Herbrich, R., & Graepel, T. (2006). *TrueskillTM: A bayesian skill rating system* (Technical Report MSR-TR-2006-80). Microsoft Research.
- Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers* (pp. 115–132).
- Hersh, W., Turpin, A., Price, S., Chan, B., Kraemer, D., Sacherek, L., & Olson, D. (2000). Do batch and user evaluations give the same results? *Proceedings of the*

- ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 17–24).
- Hinkelmann, K., & Kempthorne, O. (1994). *Design and analysis of experiments: Volume 1: Introduction to experimental design*. John Wiley & Sons.
- Huang, T.-K., Lin, C.-J., & Weng, R. C. (2004). A generalized bradley-terry model: From group competition to individual skill. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 601–608).
- Hunter, D. R. (2004). MM algorithms for generalized bradley-terry models. *The Annals of Statistics*, 32, 384–406.
- Järvelin, K., & Kekäläinen, J. (2000). Ir evaluation methods for retrieving highly relevant documents. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 41–48).
- Joachims, T. (1999). Making large-scale SVM learning practical. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods – support vector machines*. MIT Press.
- Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 132–142).
- Joachims, T. (2003). Evaluating retrieval performance using clickthrough data. In J. Franke, G. Nakhaeizadeh and I. Renz (Eds.), *Text mining*, 79–96. Physica/Springer Verlag.
- Joachims, T. (2005). A support vector method for multivariate performance measures. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 377–384). New York, NY, USA: ACM Press.

- Joachims, T., Granka, L., Pan, B., Hembrooke, H., & Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 154–161).
- Joachims, T., Granka, L., Pan, B., Hembrooke, H., Radlinski, F., & Gay, G. (2007). Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Science (TOIS)*, 25 (2). Article 7.
- Joachims, T., & Radlinski, F. (2007). Search engines that learn from implicit feedback. *Computer*, 40, 34–40.
- Jones, R., & Fain, D. C. (2003). Query word deletion prediction. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 435–436).
- Jones, R., Rey, B., Madani, O., & Greiner, W. (2006). Generating query substitutions. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 387–396).
- Kammenhuber, N., Luxenburger, J., Feldmann, A., & Weikum, G. (2006). Web search clickstreams. *Proceedings of the ACM Conference on Internet Measurement* (pp. 245–250).
- Kazawa, H., Hirao, T., & Maeda, E. (2005). Order svm: A kernel method for order learning based on generalized order statistics. *Systems and Computers in Japan*, 36, 35–43.
- Kellar, M., & Watters, C. (2006). Using web browser interactions to predict task. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 843–844).

- Kelly, D., & Teevan, J. (2003). Implicit feedback for inferring user preference: A bibliography. *SIGIR Forum* (pp. 18–28).
- Kemp, C., & Ramamohanarao, K. (2002). Long-term learning for web search engines. *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)* (pp. 263–274).
- Kendall, M., & Gibbons, J. D. (1990). *Rank correlation methods*. Edward Arnold, London.
- Khuller, S., Moss, A., & Naor, J. (1997). The budgeted maximum coverage problem. *Information Processing Letters*, 70, 39–45.
- Kleinberg, R. (2005). *Online decision problems with large strategy sets*. Doctoral dissertation, MIT.
- Lancaster, J. F., & Quade, D. (1983). Random effects in paired-comparison experiments using the bradley-terry model. *Biometrics*, 39, 245–249.
- Lau, T., & Horvitz, E. (1999). Patterns of search: Analyzing and modeling web query refinement. *Proceedings of the International Conference on User Modeling* (pp. 119–128).
- Lebanon, G. (2007). Non-parameteric modeling of partially ranked data. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 857–864). MIT Press.
- Lebanon, G., & Lafferty, J. (2002). Cranking: Combining rankings using conditional probability models on permutations. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 363–370).
- Lin, R., Louis, T. A., Paddock, S. M., & Ridgeway, G. (2006). Loss function based ranking in two-stage hierarchical models. *Bayesian Analysis*, 1, 915–946.

- Loken, E., Radlinski, F., Crespi, V., Cushing, L., & Millet, J. (2004). Online study behavior of 100,000 students studying for the SAT, ACT and GRE. *Journal of Educational Computing Research*, 30, 255–262.
- Loken, E., Radlinski, F., Crespi, V. H., & Millet, J. (2005). New SAT is to old SAT as ... *APS Observer*, 18, 15–16.
- Luaces, O., Bayón, G. F., Quevedo, J. R., Díez, J., del Coz, J. J., & Bahamonde, A. (2004). Analyzing sensory data using non-linear preference learning with feature subset selection. *Proceedings of the European Conference on Machine Learning (ECML)* (pp. 286–297).
- Manning, C. D., Raghavan, P., & Schuetze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.
- Mantell, S. P., & Kardes, F. R. (1999). The role of direction of comparison, attribute-based processing, and attitude-based processing in consumer preference. *Journal of Consumer Research*, 25, 335–352.
- McKenzie, B., & Cockburn, A. (2001). An empirical analysis of web page revisitation. *Proceedings of the International Conference on System Science (HICSS)* (p. 5019).
- Metzler, D., & Croft, W. B. (2005). A markov random field model for term dependencies. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 472–479).
- Miller, C. S., & Remington, R. W. (2004). Modeling information navigation: Implications for information architecture. *Human-Computer Interaction*, 19, 225–271.
- Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

- Moore, D. A. (1999). Order effects in preference judgments: Evidence for context dependence in the generation of preferences. *Organizational Behavior and Human Decision Processes*, 78, 146–165.
- Nemhauser, G. L., Wolsey, L. A., & Fisher, M. L. (1978). An analysis of approximation for maximizing submodular set functions. *Mathematical Programming*, 14, 265–294.
- Oztekin, B. U., Karypis, G., & Kumar, V. (2002). Expert agreement and content based reranking in a meta search environment using Mearf. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 333–344). Honolulu, Hawaii.
- Pandey, S., Roy, S., Olston, C., Cho, J., & Chakrabarti, S. (2005). Shuffling a stacked deck: The case for partially randomized ranking of search engine results. *Proceedings of the International Conference on Very Large Data Bases (VLDB)* (pp. 781–792).
- Pohl, S., Radlinski, F., & Joachims, T. (2007). Recommending related papers based on digital library access records. *Proceedings of the Joint Conference on Digital Libraries (JCDL)* (pp. 417–418).
- Provost, F., & Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 43–48).
- Radlinski, F. (2007). Addressing malicious noise in clickthrough data. *SIGIR Workshop on Learning to Rank for Information Retrieval*.
- Radlinski, F., Broder, A., Ciccolo, P., Gabrilovich, E., Josifovski, V., & Riedel, L. (2008a). Optimizing relevance and revenue in ad search: A query substitution

- approach. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*.
- Radlinski, F., & Dumais, S. (2006). Improving personalized web search using result diversification. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 691–692).
- Radlinski, F., & Joachims, T. (2005a). Evaluating the robustness of learning from implicit feedback. *ICML Workshop on Learning in Web Search*.
- Radlinski, F., & Joachims, T. (2005b). Query chains: Learning to rank from implicit feedback. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 239–248).
- Radlinski, F., & Joachims, T. (2006). Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (pp. 1406–1412).
- Radlinski, F., & Joachims, T. (2007). Active exploration for learning rankings from clickthrough data. *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 570–579).
- Radlinski, F., Kleinberg, R., & Joachims, T. (2008b). Learning diverse rankings with multi-armed bandits. *Proceedings of the International Conference on Machine Learning (ICML)*.
- Rajaram, S., Garg, A., Zhou, Z. S., & Huang, T. S. (2003). Classification approach towards ranking and sorting problems. *Lecture Notes in Artificial Intelligence* (pp. 301–312).
- Reid, J. (2000). A task-oriented non-interactive evaluation methodology for information retrieval systems. *Information Retrieval*, 2, 115–129.

- Robertson, S. E. (1977). The probability ranking principle in IR. *Journal of Documentation*, 33, 294–304.
- Rudin, C., Cortes, C., Mohri, M., & Schapire, R. E. (2005). Margin-based ranking meets boosting in the middle. *Annual Conference on Learning Theory* (pp. 63–78).
- Ryvkin, D. (2005). The predictive power of noisy elimination tournaments. The Center for Economic Research and Graduate Education – Economic Institute, Working Papers.
- Saar-Tsechansky, M., & Provost, F. (2004). Active sampling for class probability estimation and ranking. *Machine Learning*, 54, 153–178.
- Samuelson, P. (1948). Consumption theory in terms of revealed preferences. *Econometrica*, 15, 243–253.
- Scholer, F., & Williams, H. E. (2002). Query association for effective retrieval. *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)* (pp. 324–331).
- Selten, R. (1998). Aspiration adaptation theory. *Journal of Mathematical Psychology*, 42, 191–214.
- Shashua, A., & Levin, A. (2002). Ranking with large margin principle: Two approaches. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)* (pp. 937–944). MIT Press.
- Silverstein, C., Henzinger, M., Marais, H., & Moricz, M. (1998). Analysis of a very large altavista query log. *Digital SRC Technical Note #1998-014*. October 26, 1998.
- Spink, A., Wolfram, D., Jansen, M. B. J., & Saracevic, T. (2001). Searching to web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52, 226–234.

- Streeter, M., & Golovin, D. (2007). *An online algorithm for maximizing submodular functions* (Technical Report CMU-CS-07-171). School of Computer Science, Carnegie Mellon University.
- Sugiyama, K., Hatano, K., & Yoshikawa, M. (2004). Adaptive web search based on user profile constructed without any effort from users. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 675–684).
- Sun, Y., & Giles, C. L. (2007). Popularity weighted ranking for academic digital libraries. *Proceedings of European Conference on Information Retrieval (ECIR)* (pp. 605–612).
- Tan, Q., Chai, X., Ng, W., & Lee, D.-L. (2004). Applying co-training to click-through data for search engine adaptation. *Proceedings of the 9th International Conference on Database Systems for Advanced Applications (DASFAA), Lecture Notes in Computer Science* (pp. 519–532).
- Taylor, M. J., Guiver, J., Robertson, S. E., & Minka, T. (2008). Softrank: Optimizing non-smooth ranking metrics. *Proceedings of ACM International Conference on Web Search and Data Mining (WSDM)* (pp. 77–86).
- Teevan, J., Dumais, S. T., & Horvitz, E. (2005a). Beyond the commons: Investigating the value of personalizing web search. *Workshop on New Technologies for Personalized Information Access (PIA 2005)*.
- Teevan, J., Dumais, S. T., & Horvitz, E. (2005b). Personalizing search via automated analysis of interests and activities. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 449–456).
- Teevan, J., Dumais, S. T., & Horvitz, E. (2007). Characterizing the value of personalizing search. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 757–758).

- Turpin, A., & Hersh, W. (2001). Why batch and user evaluations do not give the same results. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 225–231).
- Turpin, A., & Scholer, F. (2006). User performance versus precision measures for simple search tasks. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 11–18).
- Tversky, A., & Kahneman, D. (1981). The framing of decisions and the psychology of choice. *Science, 211 New Series*, 453–458.
- Varian, H. (1992). *Microeconomic analysis*. New York, USA: Norton.
- Voorhees, E. M. (2004). Overview of TREC 2004. *Proceedings of the 13th Text REtrieval Conference*.
- Wang, X., & Zhai, C. (2007). Learn from web search logs to organize search results. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 87–94).
- White, R. W., Ruthven, I., & Jose, J. M. (2005). A study of factors affecting the utility of implicit relevance feedback. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 35–42).
- Wong, S. K. M., Yao, Y. Y., & Bollmann, P. (1988). Linear structure in information retrieval. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 219–232).
- Xue, G.-R., Zeng, H.-J., Chen, Z., Yu, Y., Ma, W.-Y., Xi, W., & Fan, W. (2004). Optimizing web search using web click-through data. *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)* (pp. 118–126).

- Yue, Y., Finley, T., Radlinski, F., & Joachims, T. (2007). A support vector method for optimizing average precision. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 271–278).
- Zhai, C., Cohen, W. W., & Lafferty, J. (2003). Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 10–17).
- Zhang, B., Li, H., Liu, Y., Ji, L., Xi, W., Fan, W., Chen, Z., & Ma, W.-Y. (2005). Improving web search results using affinity graph. *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)* (pp. 504–511).
- Zhang, S., & Markman, A. B. (2001). Processing product unique features: Alignability and involvement in preference construction. *Journal of Consumer Psychology, 11*, 13–27.
- Zhang, Y., Callan, J., & Minka, T. (2002). Novelty and redundancy detection in adaptive filtering. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 81–88).
- Zhang, Y., & Moffat, A. (2006). Some observations on user search behavior. *Proceedings of the Australasia Document Computing Symposium*.
- Zhu, X., Goldberg, A. B., Gael, J. V., & Andrzejewski, D. (2007). Improving diversity in ranking using absorbing random walks. *Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 97–104).
- Ziegler, C.-N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 22–32).