Learning to Rank with Incomplete Judgements

by

Hikmet Bahadır Şahin

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Sciences and Engineering

Koç University

July 25, 2015

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Hikmet Bahadır Şahin

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Emine Yılmaz

_____

Assoc. Prof. Deniz Yüret

_____

Prof. Attila Gürsoy

_____

Assist. Prof. Didem Unat

_____

Assist. Prof. Arzucan Özgür

Date: _____

To my parents

ABSTRACT

Research in learning to rank has been placed on developing sophisticated learning algorithms mainly, assuming the training set as a given. However, the quality of the training set directly affects the quality of the learned ranking systems. Considering the expense of obtaining relevance judgements and the budget constraint, one can get limited number of judgements to construct a training set. Much research has been devoted to distribute the judgement effort across different queries, and efficient and effective evaluation of retrieval systems given a limited judgement budget. However, little research has been done regarding the effect of incomplete judgements in learning-to-rank and available studies do not propose a solution to the problem that can be applicable given *any* objective metric and *any* sampling distribution that used to generate the incomplete judgements.

In this work, we focus on how to better utilize training sets with shallow (less) judgements per query and obtain better ranking performance using such training sets. For this aim, we generate the incomplete judgements using stratified sampling strategy and use StatAP method to compute unbiased estimates of the objective evaluation metric, i.e. mean average precision (MAP), given these judgements. Then, we use LambdaMART algorithm to train a ranking model by using the estimated and actual objective metrics. By using two different datasets, we show that estimated MAP performs significantly better than actual MAP as the training objective in learning to rank when judgements are incomplete.

ÖZETÇE

Sıralamayı öğrenme alanındaki araştırmalar eğitim veri kümesinin verildiğini farzeden gelişmiş öğrenme algoritmalarına dayanmaktadır. Ancak eğitim veri kümesinin kalitesi sıralamayı öğrenme başarımını direk olarak etkiler. Sorgu ile sonucun arasındaki ilişki kararının elde etmenin maliyeti ve bütçe kısıtı göz önüne alındığında, bir eğitim veri kümesi ancak sınırlı sayıda ilişki kararı içerebilir. Bütçenin sınırlı olduğu bir durumda, kararlama için sarfedilen çabanın çeşitli sorgular için önceliklendirilmesi ve bilgi erişim sistemlerinin etkili ve verimli bir şekilde değerlendirilmesi konusunda birçok araştırma vardır. Ancak, tamamlanmamış kararların sıralamayı öğrenmedeki etkileri hakkında çok az araştırma yapılmıştır. Varolan araştırmalarda, herhangi bir amaç metriği ve tamamlanmamış kararları oluşturan örnekleme dağılımı için uygulanabilir çözüm bulunmamaktadır.

Bu çalışmada, her sorgu için az sayıda karar barındıran eğitim veri kümelerinin daha iyi nasıl kullanılabileceğine ve bu kümeleri kullanarak nasıl daha iyi sıralama başarımı elde edebileceğine odaklandık. Bu amaçla, katmanlama örnekleme yönmeti kullanarak tamamlanmamış kararlar oluşturduk. Bu kararları kullanarak, StatAP yöntemi ile amaç fonksiyonu olan ortalama kesinlik değerlendirme metriğinin tarafsız tahminini hesapladık. Ardından, gerçek ve tahmini amaç metriklerinin LambdaMART algoritması ile eniyi değerlerini bulduk. Tahmini ortalama kesinlik modelinin tamamlanmamış kararlar kullandığımızda, gerçek ortalama kesinlik modeline göre daha iyi başarım elde ettiğini gösterdik. Ortaya sunduğumuz yönetimin herhangi bir veri kümesi için uygulanabilir olduğunu gösterebilmek amacıyla, yöntemimizi iki farklı veri kümesinde test ettik.

ACKNOWLEDGMENTS

First I would like to thank my supervisors Asst. Prof. Emine Yılmaz and Assoc. Prof. Deniz Yüret for accepting me as an Ms. student, their encouragement, critical comments and corrections of this thesis.

Besides my advisors, I would like to thank the rest of my thesis committee, Prof. Attila Gürsoy, Asst. Prof. Didem Unat and Asst. Prof. Arzucan Güngör for critical reading of my thesis and their insightful comments.

Many friends have helped me stay sane through these difficult years. I am grateful to Nermin Kurt for providing me a morale support that helps me in hard days of my research. I cannot think a world without her.

Most importantly, none of this would have been possible without the unconditional love and patience of my family. My caring, supportive and encouraging parents, Ayşen and Selçuk, have sacrificed their lives for myself and give me a opportunity to have a great educational life. I love them so much.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

There are about 46 billion pages indexed by two major search engines, Google and Bing, as of May 2015[1]. Moreover, Google has crawled 1 trillion unique links on the web at once that not all of them lead to unique web pages in 2008[2]. It would be an accurate estimate that they are crawling several trillion pages recently. According to this information, we can see that we are being overwhelmed by the excessive amount of information and finding the desired information by browsing the Web is looking for a needle in a haystack for common Web users. Thus, Importance of efficient and effective information retrieval has increased and information retrieval systems (such as search engines) have become indispensable tools for common users.

Main components of a search engine are shown in Figure 1.1. The crawler travels through Web and collects webpages (documents). Collected documents are filtered and stored in a corpus without further process. The parser analyses these stored documents and determines the terms which are going to be indexed. Then, indexer orders the terms to search documents fast. Query processor is the interface between the user and the search engine. It takes the input query from the user and process the query (e.g. stop word elimination, stemming, spell check) and transform to terms that search engine can understand. The ranker is the main component of the search engine. It is responsible to match processed queries and indexed documents according to a matching score. This score can be calculated by some heuristic formulas, ranking

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

[2]http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

models or machine learning models. Finally, a ranked list of documents related to the input query is displayed to the user.

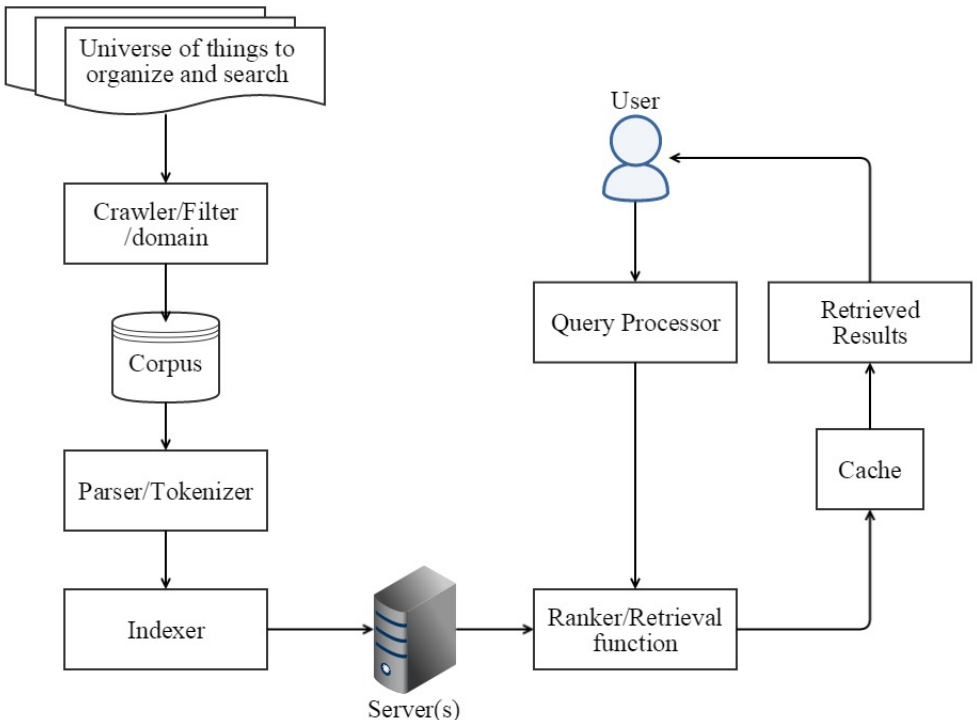


Figure 1.1: Components of a Search Engine

Since rankers are important components for search engines, ranking becomes a central problem in information retrieval. It has been paid more attention to research and development of ranking technologies. Even though there are several different approaches for this problem, most modern search engines are based on learning-to-rank which are specialized machine learning algorithms: Given a training set composed of queries and associated judged documents, a machine learning procedure learns how to combine the query and the document features to effectively find the relevance of any document to any query and thus rank a collection in response to a user input.

Studies on learning to rank has been placed on mainly feature extraction and more complex learning algorithms. However, relatively few studies have been conducted on the choice of queries and documents for constructing training dataset while utiliz-

ing the limited amount of relevance judgements available better. Related research examples include [1], [7] and we will also review several selection techniques in the following chapters. On the other hand, these selection methodologies, that are applied to a complete set of judgements, introduces incomplete and biased judgements in the constructed dataset. In addition, there are even fewer research has been done regarding the negative effects of the incompleteness of relevance judgements per query in the training set for learning-to-rank algorithms. Currently, learning to rank studies do not propose any solution to the performance problem when judgements are incomplete. The main contribution of this research is that, we study learning-to-rank performance problem when partially judged documents are used to train learning algorithms. Furthermore, we focus on a generalized solution that is applicable to *any* objective evaluation metric, *any* sampling distribution to generate incomplete judgements and *any* learning-to-rank algorithm that can optimize evaluation metrics.

In the first part of this study, we construct training sets with incomplete judgements by using the Aslam et al.[2]'s method. This method guarantees to compute unbiased estimates of any evaluation metric given incomplete judgements and any sampling distribution that is used to generate these incomplete judgements. Then, we show that the sampling distribution that is used to generate incomplete relevance judgements can be also used to compute unbiased estimates of objective evaluation measures to use in learning-to-rank algorithm, or unbiased estimates of gradients (for gradient descent based learning algorithms). In the second part, we use LambdaMART [9] algorithm to train a ranking model when the objective cost function is estimated metrics. Moreover, we compare the ranking performance when estimated and actual metrics are used in the training process. We aim to obtain an improved test set performance using estimates of evaluation measures compared to actual measures as the training objective in learning to rank when judgements are incomplete.

The rest of this thesis is organized as follows: A comprehensive literature review about learning to rank, evaluation metrics, selection methodologies to create training

datasets and learning to rank with incomplete judgements is presented in Chapter 2. In Chapter 3, we give the background information about the learning to rank algorithm and selection methodology we leveraged in our research, and we present our proposed solution. Baselines and preliminary results for two different datasets are provided in Chapter 4. Finally, the research is concluded in Chapter 5.

Chapter 2

LITERATURE REVIEW

This section briefly reviews the representative ranking models and learning-to-rank algorithms in the literature of information retrieval, and introduces how these models are evaluated, in the first half. In the second part, we will investigate the query and document selection methodologies to construct training sets and how these sets are used in learning-to-rank. The final section reviews the effects of selection methodologies on learning-to-rank-algorithms.

2.1 Ranking Models

Many ranking models have been proposed to solve ranking problem in the information retrieval literature. We can separate these models as relevance ranking models and importance ranking models.

Relevance-based ranking models ranks documents based on the occurrences of the query terms in the documents. *Boolean model* is the basic relevance-based model which predicts the relevance according to the existence of query terms inside the document [4]. These models can predict the relevance of document to a given query; however, they cannot predict the level of relevance. *Vector Space model* (VSM) is proposed to model the relevance and its level [4]. Both documents and queries are represented as vectors in the space and their similarities are measured as vector products. The most known VSM is *TF-IDF*. One of the major problem of VSM is that it assumes terms are independent. Latent Semantic Indexing tries to avoid this assumption [23]. It transforms original feature space to a latent semantic space by using singular value decomposition and measures similarity in this new space. There

are also probabilistic ranking approaches have been proposed. *BM25 model* uses the log-odds of documents' relevance to rank them [46]. It is not a single model but a combination of ranking models with small differences and parameters. In addition to the previous models, the statistical language models are also applied to information retrieval to rank documents [44]. With the query as input, documents are ranked with respect to the query likelihood, or the probability that the document's language model will generate the terms in the query.

Importance-based models rank documents based on their importance. The most popular model in this category is PageRank. It uses the probability that a surfer randomly clicking on web links will arrive a particular webpage to rank the webpages. There are many algorithms have been developed to improve PageRank in terms of accuracy, efficiency, speed, and robustness [43], [32], [37], [29].

2.2 Learning to Rank

Most modern search engines use machine learning technologies, which are also named learning-to-rank methods, to solve the problem of ranking. Since learning to rank is a kind of supervised learning, it needs a training set. A typical training set consists of queries, their associated documents and the corresponding relevance judgement. Then, a learning-to-rank procedure learns a ranking model to rank the documents in correct order for the given query. When the user enters a new query, the model can sort the documents according to their relevance and return corresponding ranked list of documents to the user.

Learning to rank algorithms can be categorized into three approaches: the point-wise, the pairwise and the listwise approaches. Even though all approaches' purpose is to learn a ranking model, they may express different input and output spaces, use different hypotheses, and optimize different objective functions.

2.2.1 Pointwise Approach

The pointwise approach is the most straightforward way to solve ranking problem since it directly uses already existing learning methods. The *input space* of this approach contains a feature vector of each document and the *output space* contains the relevance level of each document. The *hypothesis* space is the scoring function which takes feature vectors of documents from the input space and predict the relevance level of each document. According to the output of the scoring function, we can sort all documents and present the final ranked list. The *loss function* provides the accurate prediction for every document in the input space. Based on the loss function, pointwise algorithms separated into three subcategories in the literature of learning to rank: regression, classification and ordinal regression.

Regression is one of the supervised learning problems in which one tries to predict a continuous value as an output. Predicting a product's price is a possible application of this method. When one applies regression to ranking, relevance level of a document is assumed to be continuous. The ranking model is learnt by minimizing the error on the training dataset.

Cossock and Zhang [20] proposed a ranking solution by regression. Their algorithm uses a scoring function to rank documents associated with training queries. For learning purpose, they use a square loss function. Due to the nature of square loss function, there is no loss, if and only if the output of the scoring function and actual document's relevance equals. Otherwise, the loss increases in quadratic order. Since such behaviour can hurt the performance of the algorithm, they extended their method using an importance weighted model. With the help of weights, the algorithm focuses on the regression error on the relevant documents. They also added regularization terms to make the method more generalizable and avoid overfitting.

In classification problems, one tries to classify the data. Unlike regression, the output is discrete. When we formalize classification to ranking, we regard the relevance level of a document as a category level. There are two types of classification: binary

and multi-class. In binary classification, all documents are assumed to be relevant or irrelevant. However, documents associated with a query can vary according to the user. Thus, almost all common search engines have more than two relevance level to classify documents.

P. Li et al. [38] developed a multi-class classification algorithm, named McRank, for ranking problem. They used a logistic function to indicate a probability of a document belongs to a relevance level. Using this probability, they used a surrogate loss function with the boosting tree algorithm to minimize the loss in the training phase. In the test process, they converted classification probabilities into ranking scores with a weighted combination to determine the final ranked list.

Ordinal regression is a mixture of previous regression and classification ideas. The aim of ordinal regression is to find a hypothesis which classifies the results of the hypothesis into different ordered categories by using thresholds.

K. Krammer and Y. Singer [21] developed the most known algorithm on ordinal regression named PRanking. The main idea of PRanking is to find a weight vector direction, which can be used to separate the documents into ordered categories by thresholds when they are projected onto that direction. PRanking uses a general, iterative perceptron training algorithm. In 2003, Harrington [31] proposed Online Aggregate Prank-Bayes Point Machine as an extension of PRanking. First, they subsampled the training data. Then, they used a PRanking model to learn each sample. Finally, they averaged the weights and thresholds associated with the models and produced the final ranked list. This extension improves the ranking performance as well as it achieves better generalization, which means less overfitting of data.

In all pointwise ranking algorithms, document relevance is assumed to be absolute and query-independent. Because of this assumption, documents of different queries are put into the same category, if their labels are predicted as *relevant*. However, in practice, relevance is query-dependant. For instance, an irrelevant document for a particular query may be relevant to another query. Putting all documents associated

with different queries together can negatively hurt the training process. Moreover, the pointwise algorithms do not use the positions of documents in the ranked list in the training process. Thus, the pointwise loss function can be affected by too many irrelevant documents which are ranked low in the final ranked list and not necessary for users.

As a result, in terms of the performance results in the literature related to pointwise approaches, ordinal regression based algorithms have the best results followed by classification and then regression algorithms. However, due to its limitations, the pointwise approach can solve a small part of ranking problem. To find a better solution, researchers have made experiments using document pairs or entire list of documents associated with the same query as the input space. These experiments have created pairwise and listwise approaches to learning to rank.

2.2.2 Pairwise Approach

Unlike pointwise algorithms, the pairwise approach does not focus on predicting the relevance level of each document. Since it also does not assume relevance is query-independent, it tries to learn the relative order between two documents by reducing ranking into pairwise classification with respect to same query. Pairwise algorithms are likely to *rank* documents than the pointwise algorithms. The *input space* of this approach contains pairs of documents that represented by feature vectors. The *output space* contains the whether the pair of documents should be swapped or not, which takes values from $\{-1, 1\}$. The *hypothesis* space contains functions that take pairs of document as input and give the relative order between them as an output. The *loss function* measures the pairwise classification error and the main goal is to minimize the number of miss-classified pairs. This classification is different from pointwise approach, since we investigate all pairs of documents.

There are vast amount of different algorithms belonging to this approach. The main learning methods are based on neural networks, perceptrons and support vector

machines (SVM).

RankBoost [25] is an extension of AdaBoost [26] which defines the distribution on document pairs instead of individual documents. This algorithm is widely used in the literature [39] as a performance baseline for other algorithms. It combines several weak rankers and performs training by using pairwise distribution. Weak learners try to distinguish relevant and irrelevant documents with a threshold; however, they can be complicated ranking functions.

RankNet [8] is a two layered, neural network based ranking algorithm which was also used as one of the learning-to-rank algorithms by a commercial search engine. It is trained on documents pairs of same query, where both documents have different relevance levels. The RankNet optimizes a pair-based, cross-entropy cost function using gradient descent. Even though this algorithm has good scores in terms of IR metrics, its training time increases quadratically while the number of pairs increases.

RankingSVM [35] uses the SVM algorithm to do pairwise classification. The main difference between the RankingSVM and a regular SVM is their constraints, which are obtained from document pairs. Moreover, it inherits the good parts of SVM such as good generalization ability and the ability to use kernels to handle non-linearity. First RankingSVM implementations are slow in training; however, there are several speeding up extensions to solve this problem [36, 17].

The pairwise approach exceeds the limitations of the pointwise approach, since pairwise algorithms learn the relative order between documents instead of the relevance level of individual documents. Obviously, it is a much closer solution to the ranking problem. However, the pairwise approach also has its own disadvantages. Firstly, while most of the pairwise algorithms define their loss functions based on document pairs, they do not consider the position of documents in the ranked list. Since IR metrics measures the performance by considering document positions, such characteristic of pairwise approaches hurts the final score of the output ranked list. Secondly, when we have multiple relevance levels, the algorithms can not identify the

different magnitudes of levels. For instance, a document pair with *more relevant* and *irrelevant* judgements can equal to *relevant* and *irrelevant* document pairs. Finally, number of pairs can vary according to queries and this may cause imbalanced distribution across queries. This kind of data hurts learning process as well as the overall performance of the algorithm.

There are improved pairwise algorithms have been proposed in the literature to overcome these issues, such as IR-SVM [12], as an extension to RankingSVM, Multiple Hyperlane Ranker [45] and Robust Sparse Ranker [49]. Balancing the document pair distribution across queries and using top-ranked documents are also two of these improvements. Studies show that better performance is achieved by these modifications. Nevertheless, the listwise approach tries to solve the disadvantages of the pairwise algorithms within a better and more likely to rank.

2.2.3 Listwise Approach

Pointwise and pairwise approaches try to solve ranking problem by reducing it to regression or classification. They optimize their loss functions according to the absolute or relative relevance of documents and provide a ranked list at the end. However, their performance is not good enough according to IR evaluation metrics. On the other hand, listwise ranking algorithms learns directly on document lists associated with queries which makes these lists our *input space*. The *output space* contains the final ranked list (or permutations) of the documents and *hypothesis* contains a function which gives a score to each document first and then sorts them to obtain the desired permutation. The *loss functions* differ with respect to the used listwise type. The first type of loss function directly optimizes the IR evaluation metrics. The second type minimizes a listwise loss function which is defined on permutations with respect to ranking properties of IR.

The ranking performance of an algorithm is evaluated by IR metrics and optimizing these to learn the ranking model is the most straight choice. Nevertheless,

optimizing these metrics is not an easy task because they are positioned-based, i.e. they measure the quality according to the positions of the correctly ranked documents. Evaluation metrics are non-smooth functions which means they are discontinuous and cannot be differentiated, since in the learning process, documents are re-ranked each iteration of training [55]. However, gradient-based optimization techniques we reviewed until now were developed to handle continuous and differentiable cases. The literature provides several different approaches to overcome this difficulty.

LambdaRank [10] uses the approach of defining the gradient of the target evaluation metric only at the positions needed. LambdaRank uses the λ functions, as its gradients, which are calculated by RankNet's cost function with the amount of change in the metric's value by swapping two documents. Using a gradient descent method, it optimizes evaluation metric and re-ranks the documents. The corresponding metrics can be MAP, NDCG or MRR and LambdaRank has been proven to find the local optima for these target metrics [24].

LambdaMART [9] is further expansion of LambdaRank, which uses the same λ function combined with another learning algorithm, named Multiple Additive Regression Trees(MART) . In terms of ranking performance, there is a slight difference between LambdaRank and LambdaMART; however, in terms of performance and speed, LambdaMART outperforms since it can be easily parallelized and distributed. We will give detailed information about this method and its predecessors in the following chapter.

SoftRank [51] approximates the evaluation metrics to make them smooth and differentiable. Unlike λ-based algorithms, SoftRank does not have an internal ranking part which sorts documents at each iteration according to new scores. Instead, SoftRank constructs a score distribution by using a Gaussian distribution and uses them as rank distribution. According to this approximation, SoftRank computes the expected value of the evaluation metric, which is smooth and differentiable, and optimizes the metric by gradient descent.

RankGP [56], on the other hand, uses a genetic algorithm to optimize non-smooth cost functions directly. The ranking model is defined as a tree, whose leaf nodes are features and non-leaf nodes are basic mathematical operators, such as summation, subtraction, multiplication and division. In the training phase, it uses single population genetic programming which is widely used to optimize non-smooth and non-differentiable loss functions. By using the evaluation measure as the fitness function, the equivalent of loss function, the algorithm uses several evolution mechanism for optimization.

We examined three different algorithms which directly optimize the IR metrics in different ways. The second category of the listwise approach defines listwise loss functions based on the understanding on the properties of ranking for information retrieval. These types of algorithms also optimize evaluation metrics; however, they do this indirectly. They represent a ranked list as the permutation probability distribution. By using the Luce model, probability of each possible permutation of ranked lists is defined. Probabilities of this model are continuous, differentiable and convex with respect to the documents score.

The ListNet [13] algorithm uses a KL-divergence between two permutation probability distributions. It leverages a neural network to learn the ranking model by gradient descent. This algorithm has better metric performance than most of the known pairwise algorithms, which were also explained previously. However, its complexity is also higher than pairwise algorithms since it assumes all permutations as the scores of the documents and calculates them.

ListMLE [54] algorithm is proposed to reduce ListNet's complexity. This method uses the negative log-likelihood of the true permutation as the loss function, which is also called the likelihood loss. Since we need to compute the probability of a single permutation in this way, the training complexity is greatly reduced. This loss function is also convex; therefore, gradient descent method can be used for optimizing it.

We examined different kinds of listwise algorithms that are given in the literature.

They model the ranking problem in a more *ranking-based* manner than the pointwise and pairwise approaches. Thus, the listwise approach can also solve some problems that we observed in the previous two approaches. It takes the entire set of documents associated with a query as the input and gives their ranked list as the output. This property provides a separation of documents from different queries and use the documents' positions in the ranked list in the learning phase. Some studies also present that the performances of the listwise ranking algorithms are generally better than the other two approaches.

On the other hand, most of listwise algorithms are more complex than pointwise and pairwise approaches, which makes their training time to take longer. It is a major problem for algorithms that cannot be parallelized or distributed, e.g. neural network based methods. Moreover, listwise approach cannot handle cases with incomplete relevance judgements since most of evaluation metrics are not robust with this case.

2.3 Information Retrieval Evaluation Metrics

Given the amount of ranking algorithms introduced previously, evaluation methods are needed to select the most effective algorithm.

Evaluation metrics are generally defined with respect to a collection of documents for a given query. Documents are labelled according to their relevance to the query. This labels can be binary or multilevel. Binary measures assumes that a document is whether relevant or non-relevant $\{0, 1\}$. Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) are commonly used binary measures.

Average precision (AP) is defined as the average precisions at relevant documents [6]. If a system did not retrieve all relevant documents, the remaining documents are assumed to be retrieved at infinity, which causes having 0 valued precisions:

$$AP@R = \frac{\sum_{r=1}^{R} rel(i)P@r}{N} \tag{2.1}$$

where R is the cut-off rank, r is the rank position, N is the total number of

relevant documents, *rel(i)* is the binary relevance label of the document at position r and $P@r$ is the precision at the r, i.e.

$$P@r = \frac{\sum_{i=1}^{r} rel(i)}{r} \tag{2.2}$$

MAP is the average precisions over all Q queries. Since MAP evaluates the whole ranked list of a given query, it does not need a cut-off rank.

$$MAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(q) \tag{2.3}$$

NDCG is one of the most commonly used evaluation metrics in learning to rank. Unlike binary measures such as MAP, it is one of the few metrics that can incorporate multilevel relevance labels. It is defined as the sum of the discounted cumulative gains divided by maximum such value.

$$NDCG@R = \frac{1}{Z} \sum_{r=1}^{R} \frac{2^{rel(r)} - 1}{log(1 + r)} \tag{2.4}$$

where *rel(r)* $\in [0, \infty]$ is the relevance label of the document at position r and R is the cut-off rank. Z is the normalization constant to make sure that the perfect ranking would result in 1.

The learning algorithms with respect to the number of queries and the number of relevance judgements per query may vary depending on the objective metric used. We use average precision and mean average precision as our objective metric and extend LambdaMART with average precision for ranking the documents.

2.4 Selection Methodologies for Constructing Training Set

Machine learning algorithms have a huge place in the ranking problem. More complex methods have been proposed and developed to solve this problem. However, all these ranking algorithms are affected by the quality of the given training set and significantly little research has been conducted for constructing a training set. When

ranking problem is considered, learning-to-rank algorithms requires a huge amount of training data to learn a good ranking model. To construct a training set, paid human assessors judge thousands of documents to determine whether they are related to a given query or not. However, this process is expensive and time-consuming due to human factor. When constructing a training set, the main problem is how to distribute the limited budget for document judgements across queries. Since obtaining better performance depends on the quality of data and the given budget is limited, one must decide to whether judge many queries with fewer documents per query (shallow judgement) or fewer queries with many documents per query (deep judgement).

2.4.1 Query Selection Methodologies (Deep Judgement)

In learning to rank, the performance of a ranking algorithm is directly affected by the number of labelled data in the collection. This emerges a need of selecting the most informative data for the learning task. Guiver et al. [28] argues that a reduced subset of queries can replicate the performance of complete evaluation of a system. There are different methodologies have been proposed in the literature for query selection.

Long et al. [41] propose an active learning framework which is Expected Loss Optimization (ELO) and they derive this framework to Expected DCG Loss Optimization algorithm to select most informative data. It can incorporate both query and document selection; however, it is limited to rankers that predict absolute graded relevance. However, Bilgic and Bennet [5] adapt ELO to work with any ranker.

Cai et al. [11] use another active learning scheme to select queries for ranking. They present a framework which samples informative, domain-specific queries by weighting the source of the training data as their prior knowledge.

Seung et al. [47] propose *query by committee* algorithm for query selection. The training dataset is used to train by a committee of students and the queries are selected according to the principle of maximal disagreement.

Previously explained algorithms assumes that relevance judgements of documents

are given at first; however, it is not clear to select queries if such judgements are not available. Hosseini et al. [34] suggests a mathematical formulation that models the uncertainty in IR metric evaluation effectiveness when relevance judgements are not available. They propose an adaptive selection algorithm which provides a solution to this problem.

Unlike the previous selection algorithms, Dang et al. [22] offer query reformulation techniques increase ranking efficiency. They use a log-based query expansion method using anchor text and re-rank the list of expanded queries with two features.

2.4.2 Document Selection Methodologies (Shallow Judgement)

There are several approaches proposed as document selection methodologies such as pooling, sampling, and active learning. We will investigate five different methods to select documents in this review.

Depth pooling is the most known selection technique that is proposed by TREC [53]. Retrieval systems submit their ranked list to TREC. They keep these lists in a pool. When a query is formed, the union of the top k documents is retrieved and selected to be judged by the human assessors. The main idea of depth-pooling is that the most relevant documents appear at the top of the ranked list. Thus, this method can provide most of the relevant documents in the list.

The pooling technique reduces effort needed to create relevance judgements; however, it is still an expensive approach. As an example, in TREC8 [30], 86830 relevance judgements used to measure the quality of 129 submitted systems in response to 50 queries. Moreover, depth-pooling does not help the re-usability issue of training data. One may want to find which retrieval system is the best for her IR implementation; however, she may not have enough money and time to afford collecting thousands of judgements. Also, the pooling method can not help researchers that want to test a new IR system with minimum number of judgements to realize its performance. If the pool does not have any judgements about the topic that researchers want, they

can not use this method. Finally, if an IR system's collection is dynamic, such as search engines, queries and document judgement become obsolete by time, since new documents are added to the collection. This is also called incomplete judgement relevance in the literature. Such changes in the collection may make some documents less relevant if added ones are highly relevant. It is obviously inefficient to judge new collections since the changes may occur daily or monthly.

There are active learning based algorithms and sampling methods proposed in the literature to overcome such cases. Although they have different ways to solve these problems, their aim is to minimize the judgement effort per query.

The Minimal Test Collection (MTC) [14] is an active learning algorithm which selects document according to their informativeness. It checks the difference in the performance between two systems with and without that document. If the selected document, whether it is relevant or not, affects the performance in a good way, the algorithm selects it to be judged.

The Hedge [3] algorithm is also an active learning method. It treats each retrieval system as an expert and considers the relevance of documents to a given query as advice. Each system has a weight based on its ranking performance. By using the appropriate loss function, it can find the weight of a system according to the retrieved document's relevance to a given query. It finds common documents in different retrieval systems and chooses documents that are most likely to be relevant.

The Inferred Average Precision (InfAP) [57] is actually an IR evaluation metric that handles incomplete relevance information. It selects documents from the collection by uniform random sampling. Sampled documents are representative of the whole collection. They show that infAP is more robust for calculating actual average precision for incomplete relevance judgements. Moreover, infAP and real average precision equals when collection has the complete judgement property.

Statistical AP (StatAP) [2] has similar purpose to infAp. However, it adopts a statistical method, which is called stratified sampling [48]. It uses a prior distribution

of relevance and samples documents with probability proportional to its likelihood of relevance. This method can compute unbiased estimated of any evaluation metric given incomplete judgements and any sampling distribution that is used to generate incomplete judgements. We will examine this method in detail in the following chapter.

The trend in the literature is decreasing the amount of judgement efforts as much as possible. For the evaluation of Millions Query Track [15], Carterette et al. propose that they reduce the total assessor effort significantly while they have better search performance by using training sets with more queries but shallow judgements. Moreover, Yilmaz and Robertson [58] compare the performance of deep and shallow judgements in learning-to-rank. They conclude that training sets with more queries but less judged documents per query have better test performance and are more cost-effective.

2.5 Learning to Rank with Incomplete Judgements

Evaluating the quality of the IR system results requires designed around the Cranfield paradigm [18]. According to this paradigm, the cross comparison process requires the use of a test collection which consists of a set of queries (or topics), documents and relevance judgements that indicates which documents are relevant and should be retrieved for which queries. The Cranfield paradigm assumes that every document is judged for every topic (i.e., complete relevance judgements) [19]. For small collections this assumption can be applied easily; however, for larger datasets, such as TREC, only enough documents are judged to approximately hold completeness assumptions. Furthermore, this paradigm requires unbiased relevance judgements for reliable comparison of different retrieval strategies without favouring any particular system over another.

Applying query or document selection methodology on a collection can reduce the need of human effort; however, it also introduces incompleteness and bias in the

selected or sampled new collection. Bucklet and Voorhees [7] show that evaluation metrics are not robust to incomplete judgements and propose the bpref measure to evaluate retrieval systems with incomplete judgements. Yilmaz and Aslam [57] propose infAp metric that based on uniform random sampling and demonstrate that it is more robust than the bpref. Aslam et al. [2] also propose StatAP which is based on a stratified sampling strategy and guarantees to compute unbiased estimates of any evaluation metric given incomplete judgements and any sampling distribution that is used to generate incomplete judgements. Three proposed methods aim to evaluate IR systems with incomplete relevance judgements more accurately without any bias.

Most of the state-of-the-art learning to rank algorithms aim to optimize an evaluation metric which is a quality measure of a ranked list of documents [56, 51, 9]. Hence, the quality of the learning to rank algorithms can also be negatively affected by the bias in the value of the objective metric introduced by incomplete judgements. While several research has been devoted to efficient and effective evaluation of IR systems with incomplete judgements, little research has been done regarding the effect of incomplete judgements in learning-to-rank.

He et al. [33] examine how the quality of learning to rank algorithms are affected by the incompleteness of relevance judgements per query in the training set. Given different evaluation metrics, they show that at high levels of incompleteness, optimizing for a metric that is more robust to incomplete judgements result in better test performance than optimizing for a less robust metric. They suggest that learning to rank methods should account for the incompleteness in the training data.

Even though He et al. [33] were one of the first researchers to explore the effect of incompleteness in learning to rank, they do not propose a solution to the problem that can be applicable to any given objective metric and sampling distribution that was used to generate the incomplete judgements. They assume that the incomplete judgements are a random subset of complete judgements and they use the bpref and infAp metrics that were previously explained.

In the following chapter, we show that the sampling distribution that is used to generate the incomplete relevance judgements can also be used to compute unbiased estimates of objective evaluation measures for use in learning-to-rank, or unbiased estimates of gradients (for gradient based learning algorithms).

Chapter 3

METHODOLOGY

This section gives a brief background information about LambdaMART and StatAP algorithms and presents the combination of these two algorithms.

3.1 Learning to Rank Algorithms

In this study, we focus on LambdaMART method which combines Multiple Additive Regression Tree (MART) and LambdaRank algorithms. We decide to build our technique's learning part on this algorithm since LambdaMART has proven to be a successful ranking algorithm. For instance, ensemble of LambdaMART rankers won the first place in Yahoo! Learning to Rank Challenge 2010 [16] and Yandex's Personalized Web Search Challenge in Kaggle [42]. LambdaMART developed as in Figure 3.1, therefore, we cover each of these algorithms separately.

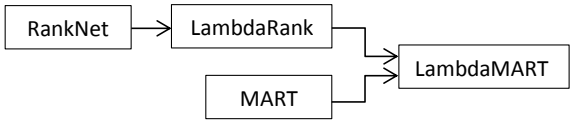


Figure 3.1: Evolution of LambdaMART algorithm

Firstly, we will briefly introduce the ideas and methods of the predecessor algorithms of LamdaMART . Then, we will explain LambdaMART and its two different distributed extensions.

3.1.1 RankNet

RankNet is a neural network based learning-to-rank algorithm which outputs a differentiable function of the model parameters. The training data is separated by query and the algorithm maps input feature vector $x \in R^n$ to a score function $f(x)$. For a given query, pair of documents D_i and D_j with different labels is selected with feature vectors x_i and x_j. Each such pair is presented to the model which computes the scores $s_i = f(x_i)$ and $s_j = f(x_j)$. The cost function is a pairwise cross-entropy loss applied to the logistic of the difference of the model scores:

$$C_{ij} = C(o_{ij}) = -S_{ij}o_{ij} + log(1 + e^{S_{ij}o_{ij}}) \tag{3.1}$$

where $o_{ij} = s_i - s_j$ and:

$$S_{ij} = \begin{cases} 1, & \text{if } rel(D_i) > rel(D_j) \\ -1, & \text{otherwise} \end{cases} \tag{3.2}$$

The gradient of the cost according to score difference is:

$$\frac{\delta C_{ij}}{\delta o_{ij}} = -\frac{S_{ij}}{1 + e^{S_{ij}o_{ij}}} \tag{3.3}$$

This gradient is used to update the weights of the neural network so as to minimize the cost via gradient descent optimization.

3.1.2 LambdaRank

RankNet is a pairwise algorithm which optimizes a smooth and differentiable cost function. However, it cannot be used to optimize IR evaluation metrics, such as MAP, NDCG, since they are non-smooth and non-differentiable. LambdaRank handles this problem by defining gradients only at the points needed. The gradient (λ functions) is formed by observing how swapping two documents changes the value of the IR measure at the end of one training iteration by sorting them. Finally, the gradient is scaled by the derivative of the RankNet costs.

Although, Burges et al. defined LambdaRank gradients for NDCG originally, the definition can be generalized and can be applied for all IR measures. In our research, we will use λ functions to optimize the average precision.

Average precision is computed as average of the precisions at relevant documents (See Equation (2.1)). Since, the precision of a document depends on the number of relevant documents above, swapping two documents may change the precision of all documents between.

Consider two documents i and j at rank positions r_i and r_j and with relevance levels $rel(i), rel(j) \in \{0, 1\}$. Assume these documents are ranked incorrectly, i.e., $r_i > r_j$ but $rel(i) > rel(j)$, where better ranked document must have lower rank index. Obviously, swapping these two documents affects the average precision and the change in the value can be written as:

$$\Delta AP = \frac{1}{R}\left[\Delta P(r_j) + \sum_{r=r_j}^{r_i} \Delta P(r) + \Delta P(r_i)\right] \tag{3.4}$$

where $\Delta P(r)$ is the change of precision value at given rank when the documents are swapped and R is the total number of relevant documents. Changes at rank r_j and r_i can be written as:

$$\Delta P(r_j) = \frac{rel(i)}{r_j}\left(rel(i) + \sum_{r=1}^{r_j-1} rel(r)\right) - \frac{rel(j)}{r_j}\left(rel(j) + \sum_{r=1}^{r_j-1} rel(r)\right) \tag{3.5}$$

$$\Delta P(r_i) = \frac{(rel(j) - rel(i))\left(\sum_{r=1, r\neq r_j}^{r_i-1} rel(r) + 1\right)}{r_i} \tag{3.6}$$

Note that equations (3.5) and (3.6) define the change of precision values where documents i and j ranked at r_i and r_j respectively. For all documents that ranked between, the change in precision value is:

$$\Delta P(r) = \frac{rel(r)}{r}\left(rel(i) - rel(j)\right) \tag{3.7}$$

where $r_j < r < r_i$.

After ΔAP computed, the lambda values can be computed as:

$$\lambda_{ij} = \Delta AP \frac{1}{1 + e^{s_i - s_j}} \tag{3.8}$$

3.1.3 Multiple Additive Regression Trees (MART)

MART [27] is a boosting tree model which performs gradient descent using a set of the least squares regression trees. It models an input feature vector $x \in R^d$ to a score function $F(x) \in R$. $F(x)$ can be written as:

$$f(x, N) = \sum_{n=1}^{N} \alpha_n f_n(x) \tag{3.9}$$

where N is the number of trees, $f_n(x)$ is a function modelled by a regression tree and the a_n are weights. Both $f_n(x)$ and a_n are learned during training. f_n maps the input feature x to a real value by passing it down to the leaves of the tree where the path is determined by a particular feature. The real valued output of the tree, v_{ln}, is associated with each leaf, where l is a particular leave and n is a particular tree. The decision functions at each node of each tree are chosen to minimize a least-squares loss.

Given n trees have been trained, MART uses gradient descent to decrease the loss. Each regression tree models the m derivatives with respect to the current model score at each training point: $\frac{\delta C}{\delta F_n}(x_i), i = 1, ...m$. Hence, we can model the derivative:

$$\delta C \approx \frac{\delta C(F_n)}{\delta F_n} \delta F \tag{3.10}$$

where $\delta C < 0$ if $\delta F = -\eta \frac{\delta C}{\delta F_n}$. Thus, every tree models the gradient with respect to the score function and new trees added with a step size ηv_{ln}. The step sizes v_{ln} can be computed exactly or approximated using Newton's step and each value multiplied by a learning rate η.

3.1.4 LambdaMART

LambdaMART is a boosted decision tree ranking algorithm that combines LambdaRank, MART and the Newton step. The gradients we use in LambdaMART is λ functions that we presented previously, and MART determines the best feature and computes the value of tree splits according least squares.

In a detailed view, the costs for each training sample m, $m = 1, ..., M$ is computed and and the λ-gradient λ_m of the given cost function with respect to model scores is calculated. Thus, the gradient of the cost function, that is assigned to training sample by model, is associated to each sample. The set of gradients are modelled by a least-squares regression tree f_n which provides that each leaf in this tree models a single value of the gradient. The overall cost is decreased by computing a Newton step v_{ln} for each leaf and every leaf value is then multiplied by a learning rate η. The LambdaMART algorithm is stated in Algorithm 1. The final model f is an ensemble such that $f(x, N) = \sum_{n=1}^{N} h_n(x)$, where h_n is the weak hypothesis of tree n.

Algorithm 1 Sequential LambdaMART

Input: Training data $\{x_m, y_m\}$ where $m = 1, ..., M$, number of trees N, number of leaves L, learning rate η.
Output: Model $f(x, N)$
$f(x, 0) =$ BaseModel(x) // If BaseModel is empty, $f(x, 0) = 0$
for $n = 1$ to N **do**
  for $m = 1$ to M **do**
    $\lambda_m = G(q, x, y, m)$ //Calculate λ_m for sample m, for the documents **x** and labels **y** associated with query **q**.
    $w_m = \frac{\partial \lambda_m}{\partial f(x_m)}$ // Derivative of λ_m.
    Create a histogram H with λ_m to find the best tree split.
  end for
  $\{R_{lk}\}_{l=1}^{L}$ // Create regression tree with L leaves on $\{x_m, \lambda_m\}_{m=1}^{M}$ and H.
  for $l = 1$ to L **do**
    $v_{ln} = \frac{\sum_{x_m \in R_{ln}} \lambda_m}{\sum_{x_m \in R_{ln}} w_m}$ // Calculate the leaf values based on Newton step approximation.
  end for
  $f(x_m, n) = f(x_m, n-1) + \eta \sum_l v_{ln} I(x_m \in R_{ln})$ // Update model
end for

With the increasing amount of data in the Web, large datasets for learning-to-rank become more available and popular. For instance, a commercial search engine can process billions of search queries and Web search information per day. Obviously, such big data offers better solutions to learning-to-rank problems that could not previously solved. However, the growth of the available data forces the algorithms to change dramatically. Since most of the learning-to-rank algorithms are centralized, they need lots of hours to be able to rank billions of documents.

To decrease the training time of LambdaMART while using large datasets, Svore and Burges [50] extends LambdaMART algorithm with two different approaches. Their first approach distributes the computation of finding the best tree split across the nodes. It is called *feature-distributed* LambdaMART. They use MPI architecture to distribute the computations. The communication cost among compute nodes is constant even if the amount of data increases. This approach gives equivalent training results with respect to centralized model in less time; however, it requires each computation worker to have full copy of training data. Thus, this approach exhausts the memory. For detailed information and algorithm, we refer reader to [50].

The second approach distributes the data to compute nodes. This method reduces the communication overhead since it does not need to know other trees' split decisions and provides the possibility of training more than billions of data. However, this approach does not give the same accuracy with centralized model. The reason is in data-distribution technique, each worker node computes its own weak hypothesis through training phases and the master node selects the best hypothesis that maximizes the target evaluation metric. The data-distributed LambdaMART is defined in Algorithm 2. It is almost similar to the sequential LambdaMART. The only difference is all computational parts are unique to worker nodes and at the end master node decides the best output.

To have a better insight, we compare LambdaRank and LambdaMART. LambdaRank is a neural network algorithm and it updates all weights after each query and

Algorithm 2 Data-distributed LambdaMART

Input: Training data $\{x_m, y_m\}$ where $m = 1, ..., M$, number of trees N, number of leaves L, learning rate η, number of workers K.
Output: Model $f(x, N)$
for $k = 1$ to K **do**
  $f(x, 0) =$ BaseModel(x)
  for $n = 1$ to N **do**
    for $m \in S_k$ **do**
      $\lambda_m = G(q, x, y, m)$ //Calculate λ_m for sample m, for the documents **x** and labels **y** associated with query **q** where m is in training data S_k on worker **k**.
      $w_m = \frac{\partial \lambda_m}{\partial f(x_m)}$ // Derivative of λ_m.
      Create a histogram H with λ_m to find the best tree split faster.
    end for
    $\{R_{lk}\}_{l=1}^{L}$ // Create regression tree $\{R_{lnk}\}_{l=1}^{L}$ on $\{x_m, \lambda_m\}$ and H where $m \in S_k$.
    for $l = 1$ to L **do**
      $v_{ln} = \frac{\sum_{x_m \in R_{ln}} \lambda_m}{\sum_{x_m \in R_{ln}} w_m}$ // Calculate the leaf values based on Newton step approximation.
    end for
    $f(x_m, n) = f(x_m, n-1) + \eta \sum_l v_{ln} I(x_m \in R_{lnk})$ // Update model
    $\{C_k(f_k(x, n))\}_{K/k}$ // Compute costs of candidate weak hypotheses
    $C(f_k(x, n)) = \sum_{i \in V} C_i(f_k(x, n))$ // Evaluate candidates from all workers.
    $f(x, n) = \underset{f_k(x,n)}{\operatorname{argmax}} C(f_k(x, n))$
  end for
end for

their documents are processed. This means that system utilizes the weights even if some queries decreases the overall performance. Furthermore, distributing or paralellizing neural networks is not a trivial task, since we cannot distribute gradients while backpropagating the system. Hence, increasing size of training data causes longer training times. On the other hand, LambdaMART splits the nodes by using all the data that falls to that node. Hence, LambdaMART updates only the split values for the current leaf nodes, but using all the data. This behaviour provides LambdaMART to choose splits and leaf values that may penalize some queries to increase overall performance. Finally, Distributing LambdaMART is easier compared to LambdaRank,

and we explained how to distribute this method above.

3.2 Statistical Average Precision - StatAP

We decide to leverage StatAP for selection and estimation part. One should note that random sampling from a complete collection of judgement treats all documents equally regardless of their relevance level. Such behaviour can cause sample too many non-relevant documents which results an unbalanced and inefficient dataset. However, StatAP selects documents that are commonly retrieved by multiple ranking systems due to the way the sampling prior is computed. Thus, as long as many underlying systems do not rank a non-relevant document at higher ranks, StatAP sampling method unlikely sample this document. Moreover, StatAP is a statistical sampling method which is able to produce unbiased values of evaluation metrics with respect to a partially judged collection of documents given any sampling distribution. In addition, the method is applicable for estimating any evaluation metric that can be defined as an expectation. The algorithm contains two steps: document sampling and estimation of evaluation metric. These two steps are completely independent from each other. The output of sampling step does not assume a particular evaluation metric being used, and the evaluation step does not have knowledge about the sample's content. This provides a flexibility to use different sampling strategies for a given set of documents.

Before starting to sample the documents, we need to decide the design of our sampling distribution. Average precision has been used with many sampling techniques, such as metasearch and pooling. It is a decent prior to dictate the relevance of documents in a ranked list. We choose the AP-prior over documents which is computed according to their ranks in the ranked lists. Since, the documents at top are likely to be relevant, this distribution should be biased towards relevant documents. For given ranked lists of documents, the sampling distribution prior of a document-query pair is:

$$W_d(r) = \frac{1}{2Z} \sum_{s=1}^{S} \frac{1}{r_s} \tag{3.11}$$

where W_d is the prior weight of the document d according to rank r, Z is the size of the ranked list, and S is the number of systems (different ranked lists of same query-document set). In Algorithm 3, the prior calculation is presented. In our research, we use the described prior; however, the sampling strategy work with any prior over documents.

Algorithm 3 Calculating Sampling Prior Distribution Weights

Input: Ranked lists from different systems s L_s, judged document list j with size Z, sample size m.
Output: Sorted list of prior weights.
for $z = 1$ to Z **do**
  if Document of query q (d_q) is **in** j **then**
    for $s = 1$ to L_S **do**
      Calculate prior weight W_{d_q}
    end for
  end if
  Sort documents' prior weights in descending order
end for

The next step is to decide to a sampling strategy which samples the documents given the prior distribution. We have several constraints for an acceptable sampling strategy. Firstly, the selected method must be able to handle both uniform and non-uniform sampling distributions and sample documents without replacement. Since our AP estimator needs *inclusion probabilities* (π_d) for documents, they must be computable. To reduce variance in estimation step, the calculated inclusion probabilities for documents must be proportional with the precision values $P@r_d$. Aslam et al. chose a method that developed by Stevens [48], stratified sampling, that covers all the considerations we described.

The stratified sampling is done in three steps. Firstly, we separate list documents, that are sorted with respect to their prior weights, into the buckets according to the

desired sample size m for each query. The bucket size equals to m; however, the last bucket may have less than m document and we can neglect it. Secondly, we sample these buckets k many times , where $0 \leq k \leq K \leq m$, with replacement. A bucket's probability equals $\sum_{d=1}^{m} W_d$'s inside of it. Finally, k many documents are selected uniformly and without replacement from the sampled bucket. The inclusion probability of a sampled document equals to the bucket's selection probability. The overall algorithm for the sampling strategy is given in Algorithm 4.

Algorithm 4 Sampling Strategy

Input: Sorted prior list of documents L_q for query q, where $q = 1, ..., Q$, sample size m
Output: Sampled document list.
for $q = 1$ to Q **do**
  Create buckets with size m
  $P_b = \sum_{d=1}^{m} W_d$ //Calculate bucket probability (P_b) as weights of its documents.
  Sample buckets m times with replacement. //Assume that sampled buckets are picked K times, where $k = 0, .., m$.
  for $k = 0$ to K **do**
    Sample documents from buckets uniformly and without replacement.
    $\pi_d = P_d$ //Make sure inclusion probability equals to bucket probability
  end for
end for

The sampling strategy of StatAP is easy to understand and implement. Even if we sample buckets with replacement, documents are selected without replacement at the end. Relevant documents are likely to have bigger π values. Hence, they are more probably to be sampled. If we can get a relevant document from a low probability bucket, its effect on the estimation will be higher than the relevant documents from high probability buckets. Also, we do not need to calculate π_d for each document separately since it is same for all documents in same bucket. Thus, calculating each bucket's probability is trivial to compute π_d values.

Obviously, we cannot use usual evaluation metrics on a given sample of documents since we have a partial part of the overall judgements. Thus, we need to make an

estimation of these metrics. As the evaluation step of StatAP algorithm, Aslam et al. provides several IR metrics' estimation formulas. Since we focus on AP in our research, we will explain how to estimate AP while we have a set of sampled documents with inclusion probabilities. Estimated precision values can be considered as mean of a population for precision calculation where one can have unequal probabilities among regarding to population's demographics. In our case, we consider the mean of relevant documents where number of them varies:

$$\widehat{P@r} = \frac{\sum_{i=1,d_i \in S}^{r} \frac{rel(i)}{\pi_i}}{r} \tag{3.12}$$

where S is the set of sampled documents, d_i is the document at rank i in the original ranked list, π_i is the inclusion probability of the document and $rel(i)$ is the relevance level of document which values can be $\{0, 1\}$. We sum the number of relevant documents in actual precision formula (Equation (2.2)); however we cannot rely on the exact number of relevant documents in our sample. To be able to find an estimated value for the relevant document count, we need to divide relevance with π_d of the respective document. If the relevant document is sampled from a high probability bucket, its effect on the precision value will not be too much. Eventually, in a full (%100) sample, all π values equal to 1. On the other hand, if we sample a relevant document with lower π, the precision value will increase more than the first case. Obviously, non-relevant documents do not have effect on the calculation, since their relevance equals 0 in any case.

Furthermore, the generalized ratio estimator, which is also used in election polls or market researches, can be used to estimate average precision [52]. Estimated AP at rank r is:

$$\widehat{AP@r} = \frac{\sum_{r=1,d_r \in S}^{R} \frac{rel(r)\widehat{P@r}}{\pi_r}}{\sum_{r=1,d_r \in S}^{R} \frac{1}{\pi_r}} \tag{3.13}$$

The numerator part of the estimated AP is almost same with the actual AP (Equation (2.1)). The major difference is that due to the generalized ratio estimator,

we divided estimated precision value with the π_r. The denominator of the actual AP is the total number relevant documents that a query has in a given rank list. However, in the estimated AP, we need to estimate this count as we explain for estimated precision in Equation (3.12).

Finally, the estimated MAP or the StatMAP becomes:

$$\widehat{MAP} = \frac{\sum_{q=1}^{Q} \widehat{AP_q}}{Q'} \tag{3.14}$$

where Q is the total number of queries in the ranked list, $\widehat{AP_q}$ is the estimated average precision of query q and Q' if the total number of valid queries which are represented by at least one relevant document in the sample.

3.3 LambdaMART with Incomplete Judgements

In the previous sections, we described a powerful learning-to-rank algorithm LambdaMART that optimizes the IR evaluation metrics in an acceptable training time. Also, we described a document selection methodology, StatAP, which can be used to compute unbiased estimates of evaluation metrics with respect to a *complete* collection given a limited number of relevance judgements and the sampling distribution that was used to generate these relevance judgements. Even though we focus on average precision, StatAP can estimate any metrics as long as they can be defined as an expectation. Because of the advantages of these both methods in their area, we choose to combine them to observe the effect of incomplete judgements in learning-to-rank.

In this section, we will show how StatAP can be used for training purposes using LambdaMART as the learning algorithm and AP as the evaluation metric. Since StatAP provides the estimated formulas of any metric given any sampling distribution, LambdaMART can optimize such estimates, instead of the actual metric values.

In Equation (3.4), the change in the actual average precision in case document i will beat (will be ranked higher) document j can be computed as:

$$\Delta AP = \frac{1}{R}\left[\Delta P(r_j) + \sum_{r=r_j, d_r \in S}^{r_i} \Delta P(r) + \Delta P(r_i)\right]$$

Using StatAP to estimate the average precision, given document collection D, a set of documents S sampled from D and the associated sampling distribution, we can show that ΔAP can be estimated as follows:

$$\widehat{\Delta AP} = \frac{1}{\sum_{d \in S} \pi_d}\left[\widehat{\Delta P(r_j)} + \sum_{r=r_j}^{r_i} \widehat{\Delta P(r)} + \widehat{\Delta P(r_i)}\right] \tag{3.15}$$

where $\widehat{\Delta P(r)})$ corresponds to the *estimated* change of precision at rank i when the two documents are swapped. Then the estimated changes at rank r_j and r_i can be computed as:

$$\widehat{\Delta P(r_j)} = \frac{rel(i)}{r_j \pi_i}\left(rel(i) + \sum_{r=1, d_r \in S}^{r_j - 1} \frac{rel(r)}{\pi_r}\right) - \frac{rel(j)}{r_j \pi_j}\left(rel(j) + \sum_{r=1, d_r \in S}^{r_j - 1} \frac{rel(r)}{\pi_r}\right) \tag{3.16}$$

$$\widehat{\Delta P(r_i)} = \left(\frac{rel(j)}{r_i \pi_j} - \frac{rel(i)}{r_i \pi_i}\right)\left(\sum_{r=1, r \neq r_j, d_r \in S}^{r_i - 1} \frac{rel(r)}{\pi_r} + 1\right) \tag{3.17}$$

Note that equations $\widehat{\Delta P(r_j)}$ and $\widehat{\Delta P(r_i)}$ define the estimated change of precisions where documents i and j are in sample and ranked at r_i and r_j in the *complete* collection, initially. For all documents that ranked between, the estimate change in precision value is:

$$\widehat{\Delta P(r)} = \frac{rel(r)}{r \pi_r}\left(rel(i) - rel(j)\right) \tag{3.18}$$

where $r_j < r < r_i$.

After ΔAP computed, the λ values can be written as:

$$\lambda_{ij} = \widehat{\Delta AP}\frac{1}{1 + e^{s_i - s_j}} \tag{3.19}$$

The rest of the training is follows the original LambdaMART that optimizes MAP scores in training. However, in our case, we optimize StatMAP (See Equation (3.14)) while training the algorithm. The validation and test scores are calculated according to original MAP formula (see Equation (2.3)).

Chapter 4

EXPERIMENTS

In this section, we briefly present the used datasets and experiment settings. Then, we compare the performance of the ranking function trained by LambdaMART over an incomplete judged collection using the StatAP estimation, with the performance of the ranking function when optimized by actual measures over the same partially judged data.

4.1 Dataset

In our experiments we used two different datasets: TREC-8 Ad-hoc retrieval track and MQ2007 [40].

4.1.1 TREC-8 Ad-hoc Retrieval Collection

The TREC-8 collection has the largest judgement collection among publicly available learning to rank datasets. Moreover, there are many documents related to the queries inside the collection and the number of documents varies between 2000 to 4000. In addition to these properties the collection contains 129 different retrieval systems that were run over 50 queries which makes computing sampling prior weights more precisely. All systems ranked 1000 documents for each query and the depth-100 pools of the returned documents were judged as relevant or non-relevant by human annotators. The effectively complete dataset we use in our experiments contains the documents from the depth-100 pools. Thus, after we remove unjudged documents from the collection, the overall number of documents is approximately 86000 left, where the final dataset has between 1000-3000 documents per query. For each query-

	A subset of used features
1	BM25
2	LogBM25_Feature
3	LM_ABS_Feature
4	LM_DIR_Feature
5	LM_JM_Feature
6	LogNormalizedTF_Feature
7	SumLogTF_Feature
8	TF_Feature
9	TF_IDF_Feature
10	LogTF_IDF_V2_Feature

Table 4.1: Feature Set

document pairs, each feature is computed over the document text (without the title) and over document text combined with the title, resulting 22 features in total. Since we used 5-Fold cross-validation for our tests, we separated our dataset into five folds such that each fold has 30 queries for training, 10 queries for validation and 10 queries for test sets. Approximately, %70 of all documents are represented in training and both validation and test sets contain %15 of rest of the documents.

4.1.2 LETOR 4.0: Millions Query 2007 Collection

We choose Millions Query 2007 (MQ2007) Collection of LETOR 4.0 [40] as our second dataset since it has different properties compared to TREC collection that can effect the ranking performance which we want to observe. This collection is obtained from the TREC's Millions Query 2007 track originally which has 30 different retrieval systems provided their runs. LETOR provides four settings for this collection: supervised ranking, semi-supervised ranking, rank aggregation and listwise ranking. Supervised ranking setting contains only the judged query-documents pairs. It has 1692 queries and approximately 70K documents. Semi-supervised setting has both judged and unjudged documents. It has the same amount of queries; however, there

are more query-documents pairs compared to supervised task. Rank aggregation settings has different features than the LETOR feature set (see Table 4.1). The features are the ranks of the documents in the respective retrieval system. Listwise ranking setting contains is similar to semi-supervised setting; however, the relevance levels in this dataset are a permutation for a query instead of multi-level relevance label.

We conducted our experiments on MQ2007 with supervised setting collection, since all documents are judged in this dataset. LETOR provides its own 5-folded version of dataset; thus, we used it directly.

4.1.3 Feature Set for the Collections

Both TREC-8 and MQ2007 datasets uses the same set of features; however, TREC-8 dataset contains only a subset of 22 features while the MQ2007 uses all of the features. The features used are shown in Table 4.1 and they are a subset of the LETOR documentation [40]. The features can be categorized under four categories. The first set of features are considered as the **low-level content features**. These set includes term frequency (tf), inverse document frequency (idf), $tf*idf$ and document length. Each of these features have four different values with respect to the four fields of a webpage: body, anchor, title and URL. The second set is **high-level content features**. These features are the outputs of BM25 and LMIR (Language Model for IR) algorithms which we review in Chapter 2. For BM25, there are four feature values which are computed using the whole document, anchor text, title, and extracted title. On the other hand LMIR has 9 different feature values with respect to three smoothing methods (DIR, JM, and ABS) and three fields (anchor, title, and extracted title). The third feature set is seven **hyperlink features** which include PageRank, HITS (both authority and hub scores), HOSTRank,, topical PageRank and topic HITS(both authority and hub scores). The final features are the **combinations of the previous feature sets** that contain both content and hyperlink information such as hyperlink-based relevance propagation and sitemap-based relevance propagation.

In total, LETOR provides 44 features for each query-document pair. The complete list of features can be found in [40].

4.2 Experiment Setting

We ran all experiments on a machine with the following hardware and operating system: Windows 8(64bit) , Intel Core i7-3639QM CPU@2.40GH (8 cores) and 16GB RAM. The maximum required memory during the experiments were between 2GB to 4GB depending on the sample size.

In our experiments, we sampled the judged documents using the sampling strategy of StatAP. TREC-8 has approximately 2000 documents per query in the dataset. Thus, we sampled documents by nine different sizes $p \in \{$ 0.5%, 1%, 1.5%, 2%, 2.5%, 3%, 4%, 5%, 10%$\}$ which were selected intuitively. Since this dataset has enough documents per query, we ensure that each query has exact same number of sampled documents for all percentages. On the other hand, MQ2007 has approximately 42 documents per query and some queries have less than 10 documents, there are even queries with only 1 documents exist, in whole dataset. Therefore, we decided to create samples by six different percentages $p \in \{$5%, 10%, 20%, 30%, 40%, 50%$\}$ which are selected intuitively, too. However, we cannot assure we have same number of documents for all queries. To reduce the variation in results, we created 30 different samples for each sample size. In total, we have 270 samples for TREC-8 and 180 samples for MQ2007.

The LambdaMART algorithm has three parameters that need to be tuned to achieve the best results initially: "Number of leaves", "Learning Rate" and "Minimum Leaf Support". We fixed the minimum leaf support such that each leaf node will take at least one document in training. We used grid search to test 30 different combinations of LambdaMART parameters for both datasets. Table 4.2 shows the values we tried for each of the parameters. Each combination of parameters is tested on 5 folds of each data set. This requires 30 x 5 x 30 x 9 = 40500 experiments on

Parameter	TREC8	MQ2007
Sample Sizes	0.5%, 1%, 1.5%, 2%, 2.5%, 3%, 4%, 5%, 10%	5%, 10%, 20%, 30%, 40%, 50%
Maximum Number of Leaves	10, 20, 40, 80, 100	2, 4, 7, 10, 20
Sampling runs for each size	30	Same
Learning Rate	0.5, 0.4, 0.3, 0.2, 0.1 0.05	Same
Minimum Leaf Support	1	Same
Number of Trees	1000	Same
Early Stop Condition	100 Iteration	Same
Feature Subsampling	1.0	Same
Cross-Validation Fold	5	Same

Table 4.2: Parameters and properties for experiments on TREC-8 and MQ2007

TREC-8 and 30 x 5 x 30 x 6 = 27000 experiments on MQ2007.

In some articles about LambdaMART, there are also feature sub-sampling and data sub-sampling parameters exist. Since we are already sampling the data, we did not use data sub-sampling as parameter. As another design choice, we decided to use all features in the training phase. Thus, we did not add this parameter to our grid search.

Huge amount of experiments is a problem. Thus, we used data-distributed LambdaMART approach with *8* threads. We also added several other optimization tricks to reduce training times. The final implementation of the algorithm takes 60 hours to run all experiments on MQ2007 and 90 hours to run whole TREC-8 experiments (time for constructing samples excluded).

4.3 Results

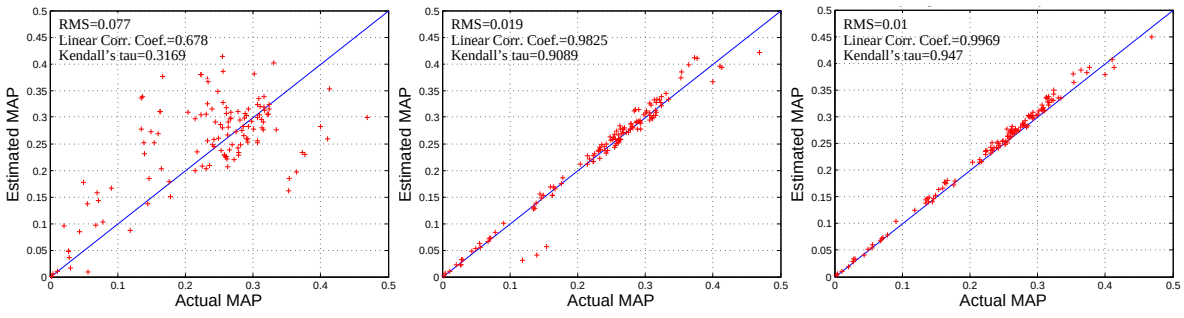
Firstly, we generated samples for TREC-8 and MQ2007, and compared the evaluation of samples with the evaluation of the original sets. Then, we compared the quality of the resulting ranking function trained by LambdaMART over two partially judged

collections using the previously described approach of measure estimation, with the quality of the ranking function when optimized by the mere use of the incomplete judged data.

For both TREC-8 and MQ2007, we used the sampling distribution described in Chapter 3 to generate samples. Then, we employ above technique for training using the partially judged collections by utilizing the estimated value of the AP with respect to the complete collection as the objective metric given in Equation (3.4) (referred to as *Estimated AP*). We compare *Estimated AP* with directly using the sampled judgements to compute the value of the objective metric, Equation (3.4), without considering the sampling distribution (referred to as Sampled Pool AP).

4.3.1 Generating Incomplete Judgements

We generate 30 different samples for experiments of TREC-8 whose sizes vary from 10 to 200 documents per query. To assure that we replicate the results of Aslam et al. [2], we compare our estimates with the actual evaluations of 129 systems of TREC-8 that are obtained by depth-100 TREC-style pooling. We use mean average precision (MAP) as evaluation metric.

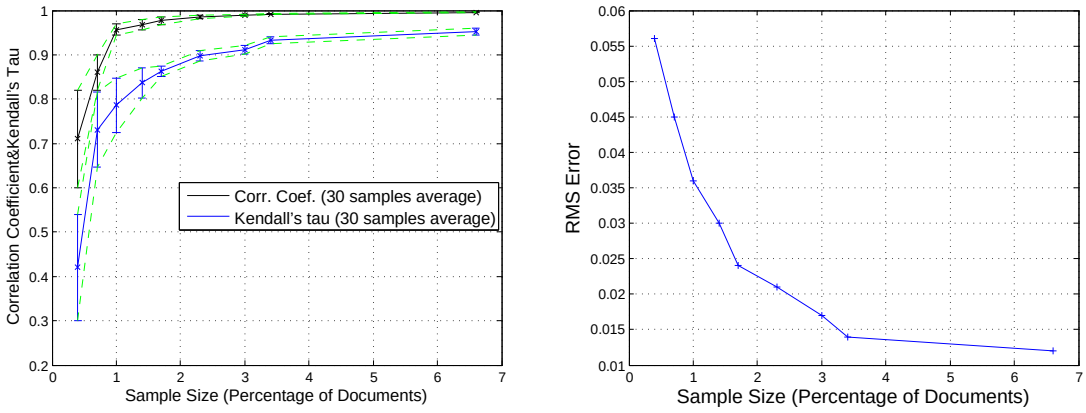


(a) (b) (c)

Figure 4.1: Sampling estimates for 129 systems of TREC-8. Sample sizes are (a) 0.5% (10 documents) (b) 4% (80 documents), (c) 10% (200 documents) respectively.

For quality evaluation of estimates, we calculate *root mean squared* (RMS) error,

linear correlation coefficient ρ and *Kendall's* τ. ρ means how well the actual and estimated values fit to a straight line, and τ shows how well the estimated measures rank the systems compared to the actual rankings. Values of ρ and τ are between $[-1, +1]$. For both, -1 indicates perfectly negatively correlated values and $+1$ indicates perfectly correlated values. However, these values do not provide any information about the difference between estimated and actual values. Thus, we use RMS error which is used to show how different the estimates values are from the actual values. Note that higher values of ρ and τ, smaller values for RMS error is what we desire in our sample comparisons. Moreover, while achieving smaller RMS error also provides both small bias and small variance.



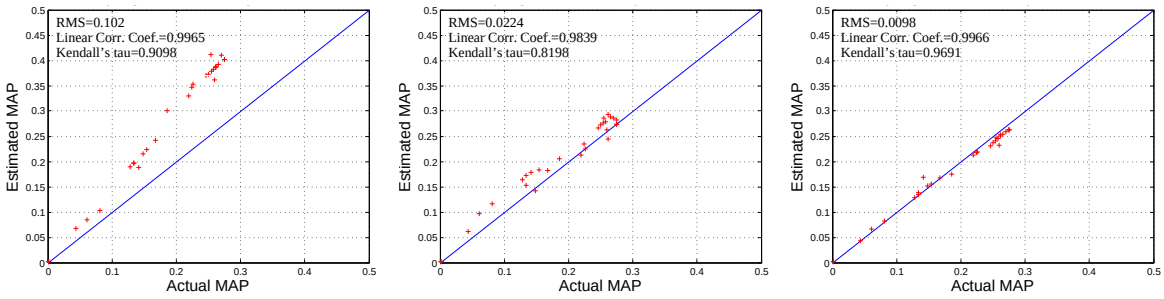
(a) (b)

Figure 4.2: Linear coefficient correlation, Kendall's τ and RMS Error changes for mean average precision in TREC8.

In Figure 4.1, we present scatter plots of one sample from sizes with 10, 80 and 200 documents respectively. We choose one example randomly among all generated samples. We report that while sample size is increasing, estimate and actual MAP results almost meet on $x = y$ line and estimate values with size 200 almost gives the same MAP evaluations with complete judgement pool.

Figure 4.2 shows how MAP estimates change in terms of ρ, τ and RMS error while

the sample size is increasing. For each sample size, we repeat the sampling process 30 times and then calculate the average ρ, τ and RMS. For ρ and τ, we also display the standard deviation bar estimated unbiased from 30 values. In general, obtained results are similar to results of Aslam et al. [2]. While the sample size is increasing, both ρ and τ values converges to $+1$ and RMS error is decreasing.



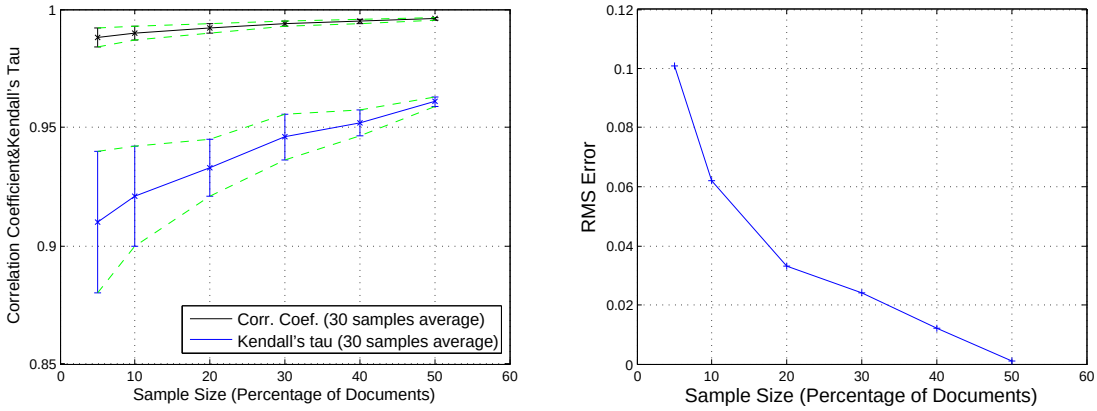
(a) (b) (c)

Figure 4.3: Sampling estimates for 30 systems of MQ2007. (a) 5% (b) 30%, (c) 50% respectively

For MQ2007, we follow the same processes to create 30 different samples for each sizes. We compare our estimates with the actual evaluation of 30 systems of MQ2007 that TREC is providing. We evaluate the actual AP values by excluding the queries that are not in the training dataset but in the original 30 systems. Figure 4.3 illustrates scatter plots of one random sample from sizes with 10, 30, 50 percentages of documents per query. In Figure 4.3 (a), it can be observed both ρ and τ values are high and indicates estimates and actual values are perfectly correlated. However, RMS error is above acceptable range which means that the selected sample has high bias and variance. While sample size is increasing in Figure 4.3 (b) and (c), ρ and τ remain closer to $+1$, and RMS error decreases which means estimate quality improves.

Changes of average ρ, τ and RMS error for MQ2007 are illustrated in Figure 4.4. The behaviour of the changes for all values are similar to the TREC-8 results.

We evaluate one sample from each size and average of 30 samples for both TREC-8



(a) (b)

Figure 4.4: Linear coefficient correlation, Kendall's τ and RMS Error changes for mean average precision in MQ2007.

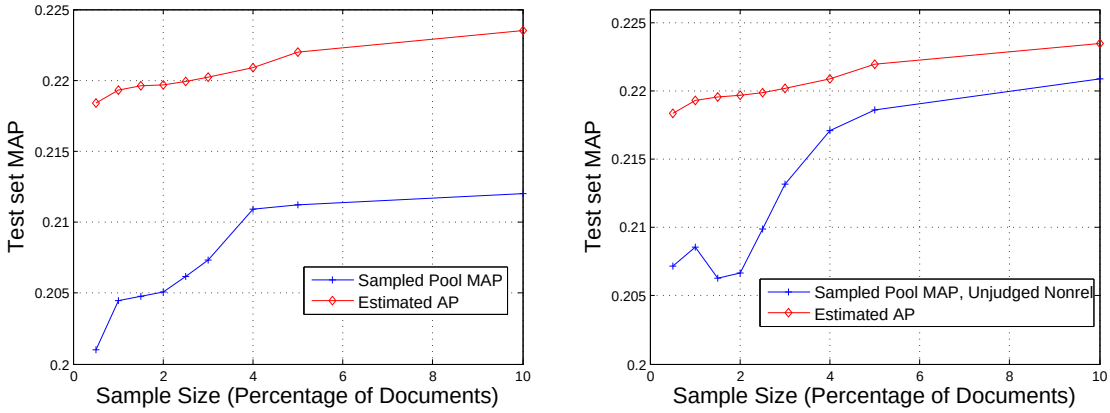
and MQ2007. The results of these evaluations are comparable with the results that Aslam et al. obtained. Thus, we continue with the same set of samples for the further experiments.

4.3.2 Training and Test Results with TREC8 and MQ2007

Figure 4.5 illustrates how the *Estimated AP* and *Sampled Pool AP* methods compare by using TREC-8. (a) corresponds to the case where unjudged documents are not considered in training based on Sampled Pool AP, while (b) plots when unjudged documents are used in training assumed to be nonrelevant. The Estimated AP approach does not need to make any such assumption since it uses the unbiased estimator over the complete pool and thus the curves corresponding to Estimated AP are exactly the same in both plots.

It can be seen that optimizing for Estimated AP consistently outperforms Sampled Pool AP. The difference is statistically significant for all experimented sample sizes with %95 confidence (see Table 4.3).

To get a better insight over the results presented in Figure 4.5 and further demon-



(a) (b)

Figure 4.5: The effect of using the estimated value of average precision given the judged documents and the sampling distribution versus (a) using only the sampled documents, and (b) using only the sampled documents while assuming unjudged document are nonrelevant - TREC-8.

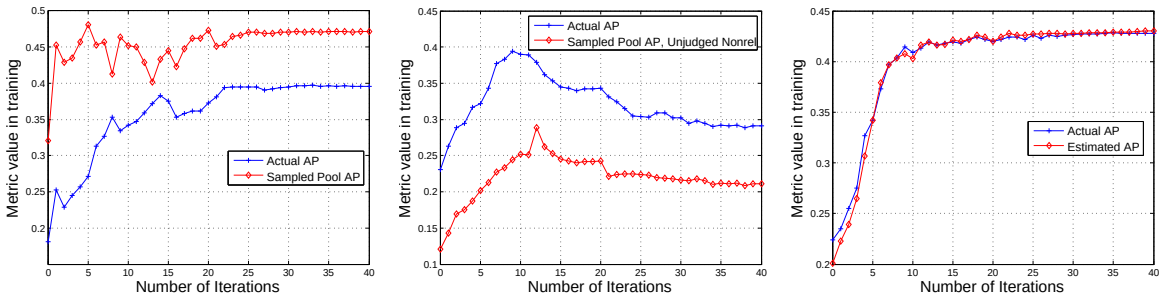
strate the advantages of the Estimated AP during training, we investigate how the value of actual average precision (Actual AP) changes as compared to the value of the objective metric during each training iteration. Actual AP is the average precision value that would be computed if the training was terminated at a specific epoch and the learning algorithm was run on the training set with complete judgements.

Figure 4.6 shows how the value of Actual AP changes as compared to the computed value of the objective metric during each training iteration. The sampled

Percentage	0.5%	1%	1.5%	2%	2.5%	3%	4%	5%	10%
p of Figure 4.5(a)	0.004	0.001	0.0003	0.060	0.039	0.011	0.102	0.235	0.156
p of Figure 4.5(b)	0.003	0.003	0.002	0.024	0.045	0.021	0.203	0.321	0.242

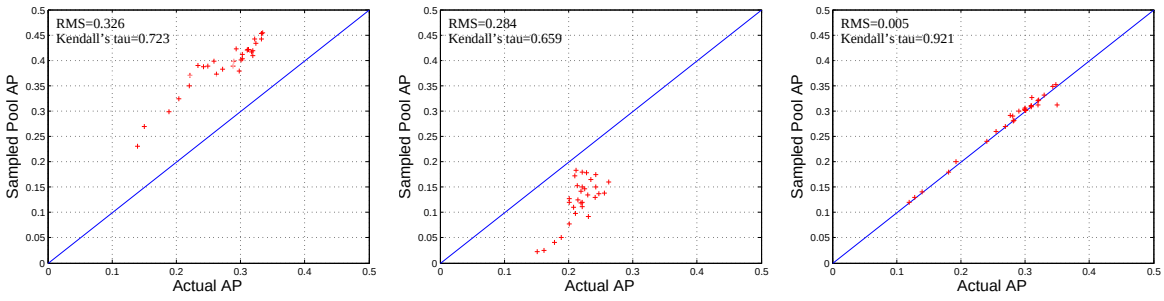
Table 4.3: Significance test results of Estimated AP versus Sampled Pool AP and Sampled Pool AP, Unjudged Nonrelevant - TREC-8.

judgements size is 200 documents. In the figure, the change in the value of Actual AP when (a) Sampled Pool AP, (b) Sampled Pool AP assuming unjudged documents are nonrelevant and (c) Estimated AP are used as the objective metrics, respectively. For comparison purposes, the plots also show the change in the value of the corresponding objective metric. It can be observed that the value of Actual AP and the objective metrics tend to change in a similar way with Estimated AP being the closest approximation to Actual AP.



(a) (b) (c)

Figure 4.6: Change in value of Actual AP when optimizing for (a) Sampled Pool AP, (b) Sampled Pool AP assuming unjudged documents are nonrelevant, and (c) Estimated AP as compared to the value of the corresponding target metric during each training epoch - TREC-8.

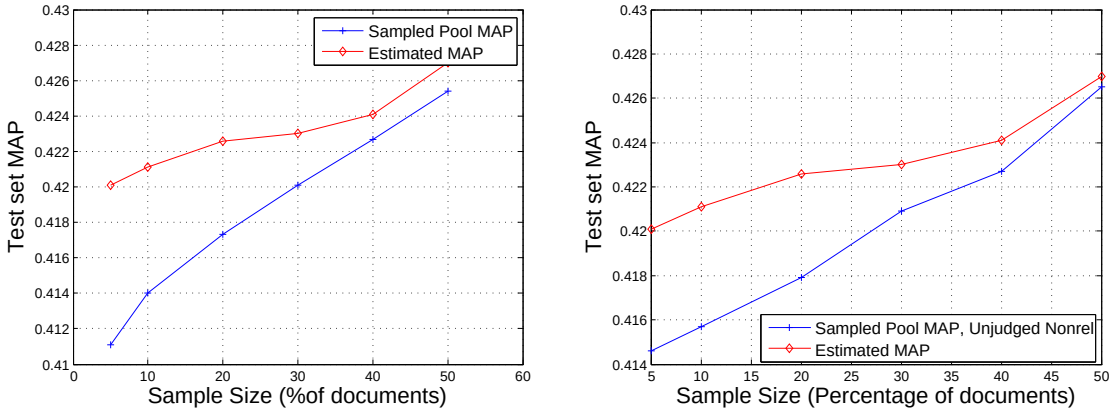


(a) (b) (c)

Figure 4.7: Actual AP vs. the value of the metric used in optimization: (a) Sampled Pool AP, (b) Sampled Pool AP assuming unjudged documents are nonrelevant, and (c) Estimated AP - TREC-8.

Figure 4.7 shows the scatter plot of the objective metrics and the actual AP values from Figure 4.6 in order to give a better insight for the direction of change and the extend of change between Actual AP and the used objective metrics. Plots also contains the RMS error and the Kendall's τ statistics between the objective metrics and Actual AP. As we previously discussed, the RMS error shows how the values are correlated and Kendall's τ shows how the ranking induced by these values are correlated. Note that achieving a low RMS error with a high Kendall's τ value as compared to Actual AP is important. In this case it shows that a change (increase or decrease) in the value of the objective metric also results in a change in the value of Actual AP.

It can be seen that Estimated AP with 200 documents per document (approximately 2% of judgements) as the objective metrics in optimization results similar to using Actual AP with complete judgements as the objective. Note that the Kendall's τ values are directly correlated with the test set performance in Figure 4.5 in which Estimated AP gives the best performance followed by Sampled Pool AP.



(a) (b)

Figure 4.8: The effect of using the estimated value of average precision given the judged documents and the sampling distribution vs. (a) using only the sampled documents, and (b) using only the sampled documents while assuming unjudged document are nonrelevant - MQ2007.

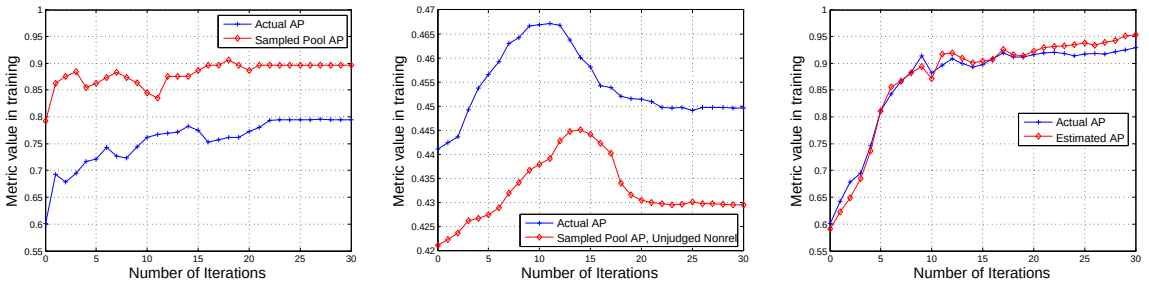
We test our second dataset, MQ2007 with its own set of parameters (see Table 4.2). The results are similar to the TREC-8 experiments. In Figure 4.8, we show the test set performance of the *Estimated AP* versus the two cases of *Sampled Pool AP*. It can be observed that Estimated AP is outperforms its competitors with a statistically significance and while the sample size is increasing Estimated and Sampled Pool AP values come closer as inclusion probabilities of documents converge to 1. Moreover, it can be see that while sample size is increasing, Estimated and Sampled Pool results are converging each other. This observation is hard to see for TREC-8 cases since the largest sample is 10% for those experiments. In Table 4.4, confidences of each sample size are presented. Moreover, Estimated AP is approximates the Actual AP values in training better than other two cases (see Figure 4.9).

Percentage	5%	10%	20%	30%	40%	50%
p of Figure 4.8(a)	0.021	0.026	0.024	0.095	0.121	0.231
p of Figure 4.8(b)	0.034	0.03	0.046	0.076	0.089	0.102

Table 4.4: Significance test results of Estimated AP versus Sampled Pool AP and Sampled Pool AP, Unjudged Nonrelevant - MQ2007.

In Figure 4.9, we present the behaviour of objective metric at each training iteration until it converges while training with MQ2007. We use samples with size 20% for this experiment. In the figure, we show the change in the value of Actual AP when (a) Sampled Pool AP, (b) Sampled Pool AP, assuming unjudged documents are nonrelevant, and (c) Estimated AP are used as the objective metrics. It is shown that Estimated AP is the closest approximation of Actual AP compared to Sampled Pool AP approaches.

Experiments with both TREC-8 and MQ2007 shows that Estimated AP is not just an alternative of the Actual AP but an unbiased estimator that can lead the



(a) (b) (c)

Figure 4.9: Change in value of Actual AP when optimizing for (a) Sampled Pool AP, (b) Sampled Pool AP assuming unjudged documents are nonrelevant, and (c) Estimated AP as compared to the value of the corresponding target metric during each training epoch - MQ2007.

optimization towards the right direction in an efficient and effective way and hence leads to better performance ranking functions.

Chapter 5

CONCLUSION

Learning to rank lies in the core information retrieval. Much research in this area has been focused to developing more complex learning algorithms, assuming training set is fixed and given. Obtaining relevance judgements to construct the training set is an expensive procedure and it is a major problem when budget is limited. When training with a limited budget, we chose training sets that contain queries with very few judged documents and focused on how learning to rank methods cope with the very limited number of judgements available per query.

Using LambdaMART as the learning algorithm, we described a technique based on statistical estimation that can be used to utilize both judged and unjudged documents in learning to rank. The proposed technique is based using statistical estimation techniques to compute unbiased estimates of objective evaluation metrics in learning to rank given the sampling distribution that was used to create the limited relevance judgements. Using this method, we showed that one can obtain significant improvements over utilizing just the judged documents during training.

The approach described in this thesis is very general as it is applicable to *any* sampling distribution that was used to create he limited relevance judgements and any objective evaluation metric. In this thesis, we mainly focused on average precision as the objective metric and LambdaMART as the learning algorithm. In the future, we plan to extend different sampling distributions and learning algorithms for optimizing different evaluation metrics when only partial judgements are available per query.

BIBLIOGRAPHY

[1] Javed A Aslam, Evangelos Kanoulas, Virgil Pavlu, Stefan Savev, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 468–475. ACM, 2009.

[2] Javed A Aslam, Virgil Pavlu, and Emine Yilmaz. A statistical method for system evaluation using incomplete judgments. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 541–548. ACM, 2006.

[3] Javed A Aslam, Virgiliu Pavlu, and Robert Savell. A unified model for metasearch, pooling, and system evaluation. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 484–491. ACM, 2003.

[4] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[5] Mustafa Bilgic and Paul N Bennett. Active query selection for learning rankers. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1033–1034. ACM, 2012.

[6] Chris Buckley and Ellen M Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 33–40. ACM, 2000.

[7] Chris Buckley and Ellen M Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32. ACM, 2004.

[8] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[9] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[10] Christopher JC Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *NIPS*, volume 6, pages 193–200, 2006.

[11] Peng Cai, Wei Gao, Aoying Zhou, and Kam-Fai Wong. Relevant knowledge helps in choosing right teacher: active query selection for ranking adaptation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 115–124. ACM, 2011.

[12] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2006.

[13] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

[14] Ben Carterette, James Allan, and Ramesh Sitaraman. Minimal test collections for retrieval evaluation. In *Proceedings of the 29th annual international ACM*

SIGIR conference on Research and development in information retrieval, pages 268–275. ACM, 2006.

[15] Ben Carterette, Virgil Pavlu, Evangelos Kanoulas, Javed A Aslam, and James Allan. Evaluation over thousands of queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 651–658. ACM, 2008.

[16] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Yahoo! Learning to Rank Challenge*, pages 1–24, 2011.

[17] Olivier Chapelle and S Sathiya Keerthi. Efficient algorithms for ranking with svms. *Information Retrieval*, 13(3):201–215, 2010.

[18] Cyril Cleverdon. The cranfield tests on index language devices. In *Aslib proceedings*, volume 19, pages 173–194. MCB UP Ltd, 1967.

[19] Cyril W Cleverdon. The significance of the cranfield tests on index languages. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. ACM, 1991.

[20] David Cossock and Tong Zhang. Subset ranking using regression. In *Learning Theory*, pages 605–619. Springer, 2006.

[21] Koby Crammer, Yoram Singer, et al. Pranking with ranking. In *NIPS*, volume 14, pages 641–647, 2001.

[22] Van Dang, Michael Bendersky, and W Bruce Croft. Learning to rank query reformulations. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 807–808. ACM, 2010.

[23] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[24] Pinar Donmez, Krysta M Svore, and Christopher JC Burges. On the local optimality of lambdarank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 460–467. ACM, 2009.

[25] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

[26] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[27] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[28] John Guiver, Stefano Mizzaro, and Stephen Robertson. A few good topics: Experiments in topic set reduction for retrieval evaluation. *ACM Transactions on Information Systems (TOIS)*, 27(4):21, 2009.

[29] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 576–587. VLDB Endowment, 2004.

[30] Donna Harman. Overview of the third text retrieval conference (TREC-3). In *The Third Text REtrieval Conference (TREC-3)*, pages 1–19, 1995.

[31] Edward F Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *ICML*, volume 20, pages 250–257, 2003.

[32] Taher H Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

[33] Ben He, Craig Macdonald, and Iadh Ounis. Retrieval sensitivity under training using different measures. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 67–74. ACM, 2008.

[34] Mehdi Hosseini, Ingemar J Cox, Natasa Milic-Frayling, Milad Shokouhi, and Emine Yilmaz. An uncertainty-aware query selection model for evaluation of ir systems. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 901–910. ACM, 2012.

[35] Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384. ACM, 2005.

[36] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[37] Amy N Langville and Carl D Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2004.

[38] Ping Li, Christopher JC Burges, Qiang Wu, JC Platt, D Koller, Y Singer, and S Roweis. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, volume 7, pages 845–852, 2007.

[39] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3–10, 2007.

[40] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3–10, 2007.

[41] Bo Long, Olivier Chapelle, Ya Zhang, Yi Chang, Zhaohui Zheng, and Belle Tseng. Active learning for ranking through expected loss optimization. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 267–274. ACM, 2010.

[42] Paul Masurel, K Lefvre-Hasegawa, Christophe Bourguignat, and Matthieu Scordia. Dataiku's solution to yandexs personalized web search challenge. Technical report, Technical report, Dataiku, 2014.

[43] Frank McSherry. A uniform approach to accelerated pagerank computation. In *Proceedings of the 14th international conference on World Wide Web*, pages 575–582. ACM, 2005.

[44] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[45] Tao Qin, Xu-Dong Zhang, De-Sheng Wang, Tie-Yan Liu, Wei Lai, and Hang Li. Ranking with multiple hyperplanes. In *Proceedings of the 30th annual inter-*

national ACM SIGIR conference on Research and development in information retrieval, pages 279–286. ACM, 2007.

[46] Stephen E Robertson. Overview of the okapi projects. *Journal of Documentation*, 53(1):3–7, 1997.

[47] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992.

[48] WL Stevens. Sampling without replacement with probability proportional to size. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 393–397, 1958.

[49] Zhengya Sun, Tao Qin, Qing Tao, and Jue Wang. Robust sparse rank learning for non-smooth ranking measures. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 259–266. ACM, 2009.

[50] Krysta M Svore and CJ Burges. Large-scale learning to rank using boosted decision trees. *Scaling Up Machine Learning: Parallel and Distributed Approaches*, 2, 2011.

[51] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 77–86. ACM, 2008.

[52] S. K. Thompson. Unequal probability sampling. In *Sampling, 3rd Edition*. Wiley-Interscience, 2012.

[53] Ellen M. Voorhees and Donna Harman. Overview of the eight text retrieval conference (TREC-8). In *The Eight Text REtrieval Conference (TREC-8)*, pages 1–24, 2000.

[54] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM, 2008.

[55] Jun Xu, Tie-Yan Liu, Min Lu, Hang Li, and Wei-Ying Ma. Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 107–114. ACM, 2008.

[56] Jen-Yuan Yeh, Jung-Yi Lin, Hao-Ren Ke, and Wei-Pang Yang. Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007)*, 2007.

[57] Emine Yilmaz and Javed A Aslam. Inferred ap: Estimating average precision with incomplete judgments. In *Fifteenth ACM International Conference on Information and Knowledge Management (CIKM)*, pages 102–111.

[58] Emine Yilmaz and Stephen Robertson. Deep versus shallow judgments in learning to rank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 662–663. ACM, 2009.