Master’s thesis Computer Science
(Chair: Databases, Track: Information System Engineering)
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Scaling Learning to Rank to Big Data

A study concerning the parallelisation of Learning to Rank algorithms using the MapReduce computing model

Niek Tax (s0166197)

Assessment Committee:
Dr. ir. Djoerd Hiemstra (University of Twente / Databases)
Dr. ir. Dolf Trieschnigg (University of Twente / Human Media Interaction)
Sander Bockting Msc (Avanade Netherlands B.V.)

Supervisors:
Dr. ir. Djoerd Hiemstra (University of Twente / Databases)
Sander Bockting Msc (Avanade Netherlands B.V.)

9th November 2014

UNIVERSITY OF TWENTE.

avanade®
Results Realized

Niek Tax: *Scaling Learning to Rank to Big Data,* A study concerning the parallelisation of Learning to Rank algorithms using the MapReduce computing model, © Februari 2014

PREFACE

Dear reader,

Thank you for taking interest in this thesis, which I have written as final project as part of the Master programme in Computer Science at the University of Twente. This research was conducted at Avanade Netherlands B.V. under the primary supervision of Sander Bockting Msc of Avanade and Dr. ir. Djoerd Hiemstra from the University of Twente. I would like to use this page to express my gratitude to everyone who supported me throughout this project in any way.

Many thanks go to Dr. ir. Djoerd Hiemstra from the University of Twente and to Sander Bockting Msc of Avanade Netherlands B.V. for their great supervision throughout the project. Even though we kept face-to-face meetings to a very minimum, you both provided me with very insightful and valuable feedback either in those meetings or per e-mail. Also, I would like to thank Dr. ir. Dolf Trieschnigg, together with Djoerd and Sander a member of the assessment committee of this graduation project, for still being available for the role as second assessor of this work, while having joined the process rather late.

In addition I would like to thank all fellow graduate interns at Avanade as well as all the Avanade employees for the great talks at the coffee machine, during the Friday afternoon drink, or elsewhere. In particular I would like to mention fellow graduate interns Fayaz Kallan, Casper Veldhuijzen, Peter Mein, and Jurjen Nienhuis for the very good time that we had together at the office as well as during the numerous drinks and dinners that we had together outside office hours.

I finish this section by thanking everyone that helped improving the quality of my work by providing me with valuable feedback. I would like to thank my good friends and former fellow boards members at study association I.C.T.S.V. Inter-*Actief* Rick van Galen and Jurriën Wagenaar, who provided me with feedback in the early stages of the process. In particular I would like to thank Jurjen Nienhuis (yes, again), for the numerous mutual feedback sessions that we held, which most certainly helped raising the quality of this thesis to a higher level.

– Niek Tax

ABSTRACT

Learning to rank is an increasingly important scientific field that comprises the use of machine learning for the ranking task. New learning to rank methods are generally evaluated in terms of ranking accuracy on benchmark test collections. However, comparison of learning to rank methods based on evaluation results is hindered by non-existence of a standard set of evaluation benchmark collections. Furthermore, little research is done in the field of scalability of the training procedure of Learning to Rank methods, to prepare us for input data sets that are getting larger and larger. This paper concerns both the comparison of Learning to Rank methods using a sparse set of evaluation results on benchmark data sets, as well as the speed-up that can be achieved by parallelising Learning to Rank methods using MapReduce.

In the first part of this thesis we propose a way to compare learning to rank methods based on a sparse set of evaluation results on a set of benchmark datasets. Our comparison methodology consists of two components: 1) Normalized Winning Number, which gives insight in the ranking accuracy of the learning to rank method, and 2) Ideal Winning Number, which gives insight in the degree of certainty concerning its ranking accuracy. Evaluation results of 87 learning to rank methods on 20 well-known benchmark datasets are collected through a structured literature search. ListNet, SmoothRank, FenchelRank, FSMRank, LRUF and LARF were found to be the best performing learning to rank methods in increasing order of Normalized Winning Number and decreasing order of Ideal Winning Number. Of these ranking algorithms, FenchelRank and FSMRank are pairwise ranking algorithms and the others are listwise ranking algorithms.

In the second part of this thesis we analysed the speed-up of the ListNet training algorithm when implemented in the MapReduce computing model. We found that running ListNet on MapReduce comes with a job scheduling overhead in the range of 150-200 seconds per training iteration. This makes MapReduce very inefficient to process small data sets with ListNet, compared to a single-machine implementation of the algorithm. The MapReduce implementation of ListNet was found to be able to offer improvements in processing time for data sets that are larger than the physical memory of the single machine otherwise available for computation. In addition we showed that ListNet tends to converge faster when a normalisation preprocessing procedure is applied to the input data. The training time of our cluster version of ListNet was found to grow better sublinearly in terms of data size increase. This shows that the cluster implementation of ListNet can be used to scale the ListNet training procedure to arbitrarily large data sets, given that enough data nodes are available for computation.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

ADMM	Alternating Direction Method of Multipliers
AP	Average Precision
CART	Classification and Regression Trees
CC	Cooperative Coevolution
CE	Cross Entropy
CRF	Conditional Random Field
CUDA	Computing Unified Device Architecture
DCG	Discounted Cumulative Gain
DSN	Deep Stacking Network
EA	Evolutionary Algorithm
ERR	Expected Reciprocal Rank
ET	Extremely Randomised Trees
FPGA	Field-Programmable Gate Array
GA	Genetic Algorithm
GBDT	Gradient Boosted Decision Tree
GP	Genetic Programming
GPGPU	General-Purpose computing on Graphical Processing Units
GPU	Graphical Processing Unit
HDFS	Hadoop Distributed File System
IDF	Inverse Document Frequency
IP	Immune Programming
IR	Information Retrieval
IWN	Ideal Winning Number
KL DIVERGENCE	Kullback-Leibler divergence
KNN	K-Nearest Neighbours
MAP	Mean Average Precision
MHR	Multiple Hyperplane Ranker

MLE	Maximum Likelihood Estimator
MPI	Message Passing Interface
MSE	Mean Squared Error
NDCG	Normalized Discounted Cumulative Gain
NWN	Normalised Winning Number
PCA	Principal Component Analysis
RLS	Regularised Least-Squares
SGD	Stochastic Gradient Descent
SIMD	Single Instruction Multiple Data
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
TREC	Text REtrieval Conference
UDF	User Defined Function
URL	Uniform Resource Locator
WASB	Windows Azure Storage Blob

1 INTRODUCTION

1.1 MOTIVATION AND PROBLEM STATEMENT

Ranking is a core problem in the field of information retrieval. The ranking task in information retrieval entails the ranking of candidate documents according to their relevance for a given query. Ranking has become a vital part of web search, where commercial search engines help users find their need in the extremely large document collection of the World Wide Web.

One can find useful applications of ranking in many application domains outside web search as well. For example, it plays a vital role in automatic document summarisation, where it can be used to rank sentences in a document according to their contribution to a summary of that document [27]. Learning to Rank also plays a role in the fields of machine translation [104], automatic drug discovery [6], the prediction of chemical reactions in the field of chemistry [113], and it is used to determine the ideal order in a sequence of maintenance operations [181]. In addition, Learning to Rank has been found to be a better fit as an underlying technique compared to continuous scale regression-based prediction for applications in recommender systems [4, 141], like those found in Netflix or Amazon.

Luhn [139] was the first to propose a model that assigned relevance scores to documents given a query back in 1957. This started a transformation of the Information Retrieval field from a focus on the binary classification task of labelling documents as either *relevant* or *not relevant* into a *ranked retrieval* task that aims at ranking the documents from most to least relevant. Research in the field of ranking models has long been based on manually designed ranking functions, such as the well-known BM25 model [180], that simply rank documents based on the appearance of the query terms in these documents. The increasing amounts of potential training data have recently made it possible to leverage machine learning methods to obtain more effective models. Learning to Rank is the relatively new research area that covers the use of machine learning models for the ranking task.

In recent years several Learning to Rank benchmark data sets have been proposed that enable comparison of the performance of different Learning to Rank methods. Well-known benchmark data sets include the *Yahoo! Learning to Rank Challenge* data set [44], the *Yandex Internet Mathematics competition*[1], and the *LETOR* data sets [168] that are published by Microsoft Research.

1 http://imat-relpred.yandex.ru/en/

One of the concluding observations of the Yahoo! Learning to Rank Challenge was that almost all work in the Learning to Rank field focuses on ranking accuracy. Meanwhile, efficiency and scalability of Learning to Rank algorithms is still an underexposed research area that is likely to become more important in the near future as available data sets are rapidly increasing in size [45]. Liu [135], one of the members of the LETOR team at Microsoft, confirms the observation that efficiency and scalability of Learning to Rank methods has so far been an overlooked research area in his influential book on Learning to Rank.

Some research has been done in the area of parallel or distributed machine learning [53, 42]. However, almost none of these studies include the Learning to Rank sub-field of machine learning. The field of efficient Learning to Rank has been getting some attention lately [15, 16, 37, 194, 188], since Liu [135] first stated its growing importance back in 2007. Only a few of these studies [194, 188] have explored the possibilities of efficient Learning to Rank through the use of parallel programming paradigms.

MapReduce [68] is a parallel computing model that is inspired by the *Map* and *Reduce* functions that are commonly used in the field of functional programming. Since Google developed the MapReduce parallel programming framework back in 2004, it has grown to be the industry standard model for parallel programming. The release of Hadoop, an open-source version of MapReduce system that was already in use at Google, contributed greatly to MapReduce becoming the industry standard way of doing parallel computation.

Lin [129] observed that algorithms that are of iterative nature, which most Learning to Rank algorithms are, are not amenable to the MapReduce framework. Lin argued that as a solution to the non-amenability of iterative algorithms to the MapReduce framework, iterative algorithms can often be replaced with non-iterative alternatives or by iterative alternatives that need fewer iterations, in such a way that its performance in a MapReduce setting is good enough. Alternative programming models are argued against by Lin, as they lack the critical mass as the data processing framework of choice and are as a result not worth their integration costs.

The appearance of benchmark data sets for Learning to Rank gave insight in the ranking accuracy of different Learning to Rank methods. As observed by Liu [135] and the Yahoo! Learning to Rank Challenge team [45], scalability of these Learning to Rank methods to large chunks of data is still an underexposed area of research. Up to now it remains unknown whether the Learning to Rank methods that perform well in terms of ranking accuracy also perform well in terms of scalability when they are used in a parallel manner using the MapReduce framework. This thesis aims to be an exploratory start in this little researched area of parallel Learning to Rank.

1.2 RESEARCH GOALS

The objective of this thesis is to explore the speed-up in execution time of Learning to Rank algorithms through parallelisation using the MapReduce computational model. The set of Learning to Rank models described in literature is of such size that it is infeasible to conduct exhaustive experiments on all Learning to Rank models. Therefore, we set the scope of our scalability experiment to include those Learning to Rank algorithms that have shown leading performance on relevant benchmark data sets.

The existence of multiple benchmark data sets for Learning to Rank makes it non-trivial to determine the best Learning to Rank methods in terms of ranking accuracy. Given two ranking methods, there might be non-agreement between evaluation results on different benchmarks on which ranking method is more accurate. Furthermore, given two benchmark data sets, the sets of Learning to Rank methods that are evaluated on these benchmark data sets might not be identical. To be able to scope our experiment on the speed-up of the most accurate Learning to Rank methods we first need to identify what the most accurate Learning to Rank algorithms are. This brings us the the first research question:

RQ1 What are the best performing Learning to Rank algorithms in terms of ranking accuracy on relevant benchmark data sets?

Ranking accuracy is an ambiguous concept, as there exists several metrics that can be used to express the accuracy of a ranking. We will explore several metrics for ranking accuracy in section 2.2.

After determining the most accurate ranking methods, we perform speed-up experiment on distributed MapReduce implementations of those algorithms. We formulate this in the following research question:

RQ2 What is the speed-up of those Learning to Rank algorithms when executed using the MapReduce framework?

With multiple existing definitions of speed-up, we will use the speed-up definition known as *relative speed-up* [197], which is formulated as follows:

$$S_N = \frac{\text{execution time using one core}}{\text{execution time using } N \text{ cores}}$$

The single core execution time in this formula is defined as the time that the fastest known single-machine implementation of the algorithm takes to perform the execution.

1.3 APPROACH

We will describe our research methodology on a per Research Question basis. Prior to describing the methodologies for answering the Research Questions, we will describe the characteristics of our search for related work.

1.3.1 *Literature Study Methodology*

A literature study will be performed to get insight in relevant existing techniques for large scale Learning to Rank. The literature study will be performed by using the following query:

("learning to rank" OR *"learning-to-rank"* OR *"machine learned ranking")* AND *("parallel"* OR *"distributed")*

and the following bibliographic databases:

- Scopus
- Web of Science
- Google Scholar

The query incorporates different ways of writing of Learning to Rank, with and without hyphens, and the synonymous term *machine learned ranking* to increase search recall, i.e. to make sure that no relevant studies are missed. For the same reason the terms *parallel* and *distributed* are included in the search query. Even though *parallel* and *distributed* are not always synonymous, we are interested in both approaches in non-sequential data processing.

A one-level forward and backward reference search is used to find relevant papers missed so far. To handle the large volume of studies involved in the backward and forward reference search, relevance of the studies will be evaluated solely on the title of the study.

1.3.2 *Methodology for Research Question I*

To answer our first research question we will identify the Learning to Rank benchmark data sets that are used in literature to report the ranking accuracy of new Learning to Rank methods. These benchmark data sets will be identified by observing the data sets used in the papers found in the previously described literature study. Based on the benchmark data sets found, a literature search for papers will be performed and a cross-benchmark comparison method will be formulated. This literature search and cross-benchmark comparison procedure will be described in detail in section 5.

1.3.3 *Methodology for Research Question II*

To find an answer to the second research question, the Learning to Rank methods determined in the first research question will be implemented in the MapReduce framework and training time will be measured as a factor of the number of cluster nodes used to perform the computation. The HDInsight cloud-based MapReduce platform from Microsoft will be used to run the Learning to Rank algorithms on. HDInsight is based on the popular open source MapReduce implementation Hadoop[2].

To research the speed-up's dependence on the amount of data that is being processed, the training computations will be performed on data sets of varying sizes. We use the well-known benchmark collections LETOR 3.0, LETOR 4.0 and MSLR-WEB30/40K as a starting set of data sets for our experiments. Table 1 shows the data sizes of these data sets. The data sizes reported are not the total on-disk sizes of the data sets, but instead the size of the largest training set of all data folds (for an explanation of the concept of data folds, see 2.4).

Data set	Collection	Size
OHSUMED	LETOR 3.0	4.55 MB
MQ2008	LETOR 4.0	5.93 MB
MQ2007	LETOR 4.0	25.52 MB
MSLR-WEB10K	MSLR-WEB10K	938.01 MB
MSLR-WEB30K	MSLR-WEB30K	2.62 GB

Table 1: The LETOR 3.0, LETOR 4.0 and MSLR30/40K data sets and their data sizes

MSLR-WEB30K is the largest in data size of the benchmark data sets used in practice, but 2.62GB is still relatively small for MapReduce data processing. To test the how the computational performance of Learning to Rank algorithms both on cluster and on single-node computation scales to large quantities of data, larger data sets will be created by cloning the MSLR-WEB30K data set such that the cloned queries will hold new distinct query ID's.

1.4 THESIS OVERVIEW

CHAPTER 2: BACKGROUND introduces the reader to the basic principles and recent work in the fields of Learning to Rank and the MapReduce computing model. Aims at bringing those unfamiliar with those topics up to speed to be able to understand the succeeding chapters of this thesis.

CHAPTER 3: RELATED WORK concisely describes existing work in the field of parallel and distributed Learning to Rank.

CHAPTER 4: BENCHMARK DATA SETS described the characteristics of the existing benchmark data sets in the Learning to Rank field.

2 http://hadoop.apache.org/

CHAPTER 5: CROSS BENCHMARK COMPARISON described the methodology of a comparison of ranking accuracy of Learning to Rank methods across benchmark data sets and described the results of this comparison.

CHAPTER 6: SELECTED LEARNING TO RANK METHODS describes the algorithms and details of the Learning to Rank methods selected in Chapter V.

CHAPTER 7: IMPLEMENTATION describes implementation details of the Learning to Rank algorithms in the Hadoop framework.

CHAPTER 8: MAPREDUCE EXPERIMENTS presents and discusses speed-up results for the implemented Learning to Rank methods.

CHAPTER 9: CONCLUSIONS summarizes the results and answers our research questions based on the results. The limitations of our research as well as future research directions in the field are mentioned here.

CHAPTER 10: FUTURE WORK describes several directions of research worthy follow-up research based on our findings.

2 TECHNICAL BACKGROUND

This chapter is written with the goal to provide the reader with a brief introduction in the Learning to Rank field and the MapReduce computational model. Models and theories explained in this chapter can be regarded as prior knowledge needed to understand the subsequent chapters of this thesis.

2.1 A BASIC INTRODUCTION TO LEARNING TO RANK

Different definitions of Learning to Rank exist. In general, all ranking methods that use machine learning technologies to solve the problem of ranking are called Learning to Rank methods. Figure 1 describes the general process of machine learning. Input space X consists of input objects x. A hypothesis h defines a mapping of input objects from X into the output space Y, resulting in prediction $\hat{y}$. The loss of an hypothesis is the difference between the predictions made by the hypothesis and the correct values mapped from the input space into the output space, called the *ground truth labels*. The task of machine learning is to find the best fitting hypothesis h from the set of all possible hypotheses H, called the hypothesis space.

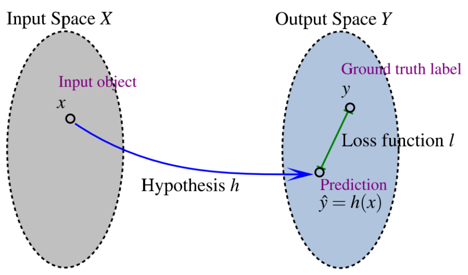

Figure 1: Machine learning framework for Learning to Rank, obtained from Liu [135]

Liu [135] proposes a more narrow definition and only considers ranking methods to be a Learning to Rank method when it is *feature based* and uses *discriminative training*, in which the concepts *feature based* and *discriminative training* are itself defined as:

FEATURE BASED means that all documents under investigation are represented by feature vectors. Those feature vectors can be used to predict the relevance of the documents to the query, or the importance of the document in itself.

DISCRIMINATIVE TRAINING means that the learning process can be well described by the four components of discriminative learning. That is, a Learning to Rank method has its own *input space*, *output space*, *hypothesis space*, and *loss function*, like the machine learning process described by Figure 1. *Input space*, *output space*, *hypothesis space*, and *loss function* are hereby defined as follows:

INPUT SPACE contains the objects under investigation. Usually objects are represented by feature vectors, extracted according to different applications.

OUTPUT SPACE contains the learning target with respect to the input objects.

HYPOTHESIS SPACE defines the class of functions mapping the input space to the output space. The functions operate on the feature vectors of the input object, and make predictions according to the format of the output space.

LOSS FUNCTION in order to learn the optimal hypothesis, a training set is usually used, which contains a number of objects and their ground truth labels, sampled from the product of the input and output spaces. A loss function calculates the difference between the predictions $\hat{y}$ and the ground truth labels on a given set of data.

Figure 2 shows how the machine learning process as described in Figure 1 typically takes place in a ranking scenario. Let q_i with $1 \leqslant i \leqslant n$ be a set of queries of size n. Let x_j^i with $1 \leqslant j \leqslant m$ be the sets of documents of size m that are associated with query i, in which each document is represented by a feature vector. The queries, the associated documents and the relevance judgements y_i are jointly used to train a model h. Model h can after training be used to predict a ranking of the documents for a given query, such the difference between the document rankings predicted by h and the actual optimal rankings based on y_i is minimal in terms of a certain loss function.

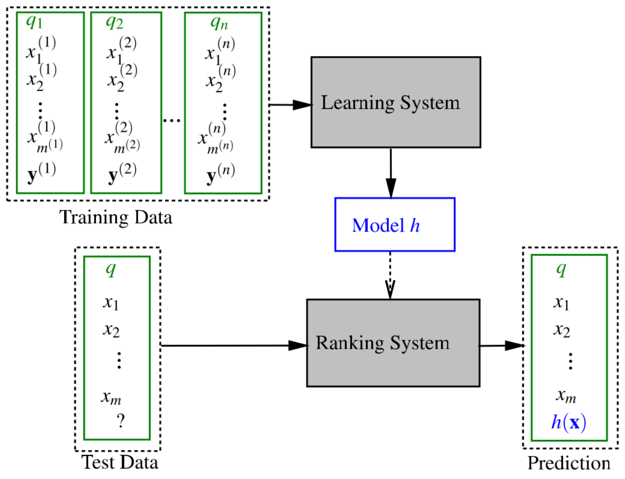

Figure 2: A typical Learning to Rank setting, obtained from Liu [135]

Learning to Rank algorithms can be divided into three groups: the pointwise approach, the pairwise approach and the listwise approach. The approaches are explained in more detail in section 2.3. The main difference between the three approaches is in the way in which they define the input space and the output space.

POINTWISE the relevance of each associated document

PAIRWISE the classification of the most relevant document out for each pair of documents in the set of associated documents

LISTWISE the relevance ranking of the associated documents

2.2 HOW TO EVALUATE A RANKING

Evaluation metrics have long been studied in the field of information retrieval. First in the form of evaluation of unranked retrieval sets and later, when the information retrieval field started focussing more on ranked retrieval, in the form of ranked retrieval evaluation. In this section several frequently used evaluation metrics for ranked results will be described.

No single evaluation metric that we are going to describe is indisputably better or worse than any of the other metrics. Different benchmarking settings have used different evaluation metrics. Metrics introduced in this section will be used in chapters 4 and 5 of this thesis to compare Learning to Rank methods in terms of ranking accuracy.

2.2.1 *Normalized Discounted Cumulative Gain*

Cumulative gain and its successors discounted cumulative gain and normalized discounted cumulative gain are arguably the most widely used measures for effectiveness of ranking methods.

2.2.1.1 *Discounted Cumulative Gain*

Discounted Cumulative Gain

There are two definitions of Discounted Cumulative Gain (DCG) used in practice. DCG for a predicted ranking of length p was originally defined by Järvelin and Kekäläinen [109] as

$$DCG_{JK} = \sum_{i=1}^{p} \frac{rel_i - 1}{\log_2(i+1)}$$

with rel_i the graded relevance of the result at position i. The idea is that highly relevant documents that appear lower in a search result should be penalized (discounted). This discounting is done by reducing the graded relevance logarithmically proportional to the position of the result.

Burges et al. [32] proposed an alternative definition of DCG that puts stronger emphasis on document relevance

$$DCG_B = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

2.2.1.2 *Normalized Discounted Cumulative Gain*

Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) normalizes the DCG metric to a value in the [0,1] interval by dividing by the DCG value of the optimal rank. This optimal rank is obtained by sorting documents on relevance for a given query. The definition of NDCG can be written down mathematically as

$$\mathrm{NDCG} = \frac{\mathrm{DCG}}{\mathrm{IDCG}}$$

Often it is the case that queries in the data set differ in the number of documents that are associated with them. For queries with a large number of associated documents it might not always be needed to rank the complete set of associated documents, since the lower sections of this ranking might never be examined. Normalized Discounted Cumulative Gain is often used with a fixed set size for the result set to mitigate this problem. NDCG with a fixed set size is often called NDCG@k, where k represents the set size.

Table 2 shows an example calculation for NDCG@k with $k = 10$ for both the Järvelin and Kekäläinen [109] and Burges et al. [32] version of DCG.

	Rank										
	1	2	3	4	5	6	7	8	9	10	Sum
rel_i	10	7	6	8	9	5	1	3	2	4	
$\frac{2^{rel_i - 1}}{\log_2(i+1)}$	512	40.4	16	55.1	99.0	5.7	0.3	1.3	0.6	2.3	732.7
$\frac{rel_i}{\log_2(i+1)}$	10	4.42	3	3.45	3.48	1.78	0.33	0.95	0.6	1.16	29.17
optimal rank	10	9	8	7	6	5	4	3	2	1	
$\frac{2^{rel_i - 1}}{\log_2(i+1)}$	512	161.5	64	27.6	12.4	5.7	2.7	1.3	0.6	0.2	788.0
$\frac{rel_i}{\log_2(i+1)}$	10	5.68	4	3.01	2.32	1.78	1.33	0.95	0.6	0.29	29.96

$$\mathrm{NDCG_B@10} = \frac{732.7}{788.0} = 0.9298$$
$$\mathrm{NDCG_{JK}@10} = \frac{29.17}{29.96} = 0.9736$$

Table 2: Example calculation of NDCG@10

2.2.2 *Expected Reciprocal Rank*

Expected Reciprocal Rank

Expected Reciprocal Rank (ERR) [46] was designed based on the observation that NDCG is based on the false assumption that the usefulness of a document at rank i is independent of the usefulness of the documents at rank less than i. ERR is based on the reasoning that a user examines search results from top to bottom and at each position has a certain probability of being satisfied in his information need, at which point he stops examining the remainder of the list. The ERR metric is defined as the expected reciprocal length of time that the user will take to find a relevant document. ERR is formally defined as

$$\mathrm{ERR} = \sum_{r=1}^{n} \frac{1}{r} \prod_{i=1}^{r-1} (1 - R_i) R_r$$

where the product sequence part of the formula represents the chance that the user will stop at position r. R_i in this formula represents the probability of the user being satisfied in his information need after assessing the document at position i in the ranking.

The algorithm to compute ERR is shown in Listing 1. The algorithm requires relevance grades g_i, $1 \leqslant i \leqslant n$ and mapping function R that maps relevance grades to probability of relevance.

```
p ← 1, ERR ← 0
for $r \leftarrow 1$ to $n$ do
    $R \leftarrow \mathcal{R}(rel_r)$
    $ERR \leftarrow ERR + p * \frac{R}{r}$
    $p \leftarrow p * (1 - R)$
end
Output ERR
```

Algorithm 1: The algorithm for computation of the ERR metric, obtained from Chapelle et al. [46]

In this algorithm $\mathcal{R}$ is a mapping from relevance grades to the probability of the document satisfying the information need of the user. Chapelle et al. [46] state that there are different ways to define this mapping, but they describe one possible mapping that is based on the Burges version [32] of the gain function for DCG:

$$\mathcal{R}(r) = \frac{2^r - 1}{2^{max\_rel}}$$

where max_rel is the maximum relevance value present in the data set.

2.2.3 *Mean Average Precision*

Average Precision (AP) [257] is an often used binary relevance judgement based metric that can be seen as a trade-off between precision and recall that is defined as *Average Precision*

$$AP(q) = \frac{\sum_{k=1}^{n} Precision(k) * rel_k}{\text{number of relevant docs}}$$

where n is the number of documents in query q. Since AP is a binary relevance judgement metric, rel_k is either 1 (relevant) or 0 (not relevant). Table 3 provides an example calculation of average precision where de documents at positions 1, 5, 6 and 8 in the ranking are relevant. The total number of available relevant documents in the document set R is assumed to be seven. Mean Average Precision (MAP) is the average AP for a set of queries. *Mean Average Precision*

	Rank										Sum
	1	2	3	4	5	6	7	8	9	10	
r_i	1	0	0	0	1	1	0	1	0	0	
P@i	1				0.4	0.5		0.5			2.4
									# of relevant docs	=	7
									AP@10	=	0.34

Table 3: Average Precision example calculation.

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

2.3 APPROACHES TO LEARNING TO RANK

2.3.1 *Pointwise Approach*

The pointwise approach can be seen as the most straightforward way of using machine learning for ranking. Pointwise Learning to Rank methods directly apply machine learning methods to the ranking problem by observing each document in isolation. They can be subdivided in the following approaches:

1. regression-based, which estimate the relevance of a considered document using a regression model.
2. classification-based, which classify the relevance category of the document using a classification model.
3. ordinal regression-based, which classify the relevance category of the document using a classification model in such a way that the order of relevance categories is taken into account.

Well-known algorithms that belong to the pointwise approach include McRank [127] and PRank [58].

2.3.2 *Pairwise Approach*

Pointwise Learning to Rank methods have the drawback that they optimise real valued expected relevance, while evaluation metrics like NDCG and ERR are only impacted by a change in expected relevance when that change impacts a pairwise preference. The pairwise approach solves this drawback of the pointwise approach by regarding ranking as pairwise classification.

Aggregating a set of predicted pairwise preferences into the corresponding optimal rank is shown to be a NP-Hard problem [79]. An often used solution to this problem is to upper bound the number of classification mistakes by an easy to optimise function [19].

Well-known pairwise Learning to Rank algorithms include FRank [210], GBRank [253], LambdaRank [34], RankBoost [81], RankNet [32], Ranking SVM [100, 110], and SortNet [178].

2.3.3 *Listwise Approach*

Listwise ranking optimises the actual evaluation metric. The learner learns to predict an actual ranking itself without using an intermediate step like in pointwise or pairwise Learning to Rank. The main challenge in this approach is that most evaluation metrics are not differentiable. MAP, ERR and NDCG are non-differentiable, non-convex and discontinuous functions, what makes them very hard to optimize.

Although the properties of MAP, ERR and NDCG are not ideal for direct optimisation, some listwise approaches do focus on direct metric optimisation [249, 203, 47]. Most listwise approaches work around optimisation of the non-differentiable, non-convex and discontinuous metrics by optimising surrogate cost functions that mimic the behaviour of MAP, ERR or NDCG, but have nicer properties for optimisation.

Well-known algorithms that belong to the listwise approach include AdaRank [236], BoltzRank [217], ListMLE [235], ListNet [39], RankCosine [173], SmoothRank [47], SoftRank [203], SVM^{map} [249].

2.4 CROSS-VALIDATION EXPERIMENTS

A cross-validation experiment [116], sometimes called rotation estimation, is an experimental set-up for model evaluation where the data is split into k chunks of approximately equal size, called *folds*. One of the folds is used as validation set, one of the folds is used as test set, and the rest of the $k-2$ folds are used as training data. This procedure is repeated k times, such that each fold is once used for validation, once as test set, and $k-2$ times as training data. The performance can be measured in any model evaluation metric, and is averaged over the model performances on each of the folds. The goal of cross-validation is to define a data set to test the model in the training phase, in order to limit the problem of overfitting.

Cross-validation is one of the most frequently used model evaluations methods in the field of Machine Learning, including the Learning to Rank subfield. Often, folds in a cross-validation are created in a *stratified* manner, meaning that the folds are created in such a way that the distributions of the target variable are approximately identical between the folds.

2.5 AN INTRODUCTION TO THE MAPREDUCE PROGRAMMING MODEL

MapReduce [68] is a programming model invented at Google, where users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. This model draws its inspiration from the field of functional programming, where *map* and *reduce* (in some functional languages called *fold*) are commonly used functions.

This combination of the *map* and *reduce* functions allows for parallel computation. In the *map* phase parallel computation can be performed by simply splitting the input data after a certain number of bytes, where each worker nodes performs the user-specified *map*-function on its share of the data. Before the *reduce* phase these intermediate answers of the different worker nodes are transformed in such a way that they are grouped by key value, this is called the shuffle-phase. After the shuffle-phase, the user-defined *reduce*-function is applied to each group of key/value pairs in the reduce phase. Since the key/-value pairs are already grouped by key in the shuffle phase, this *reduce*-function can be applied to a group of key/value pairs on any of the worker nodes.

3 RELATED WORK

3.1 LITERATURE STUDY CHARACTERISTICS

The literature research is performed by using the bibliographic databases Scopus and Web of Science with the following search query:

("learning to rank" OR *"learning-to-rank"* OR *"machine learned ranking")* AND *("parallel"* OR *"distributed")*

An abstract-based manual filtering step is applied where those results are filtered that use the terms *parallel* or *distributed* in context to *learning to rank*, *learning-to-rank* or *machine learned ranking*. As a last step we will filter out studies based on the whole document that only focus on efficient query evaluation and not on parallel or distributed learning of ranking functions, as those studies are likely to meet listed search terms.

On Scopus, the defined search query resulted in 65 documents. Only 14 of those documents used *large scale*, *parallel* or *distributed* terms in context to the *learning to rank*, *learning-to-rank* or *machine learned ranking*. 10 out of those 14 documents focussed on parallel or distributed learning of ranking functions.

The defined search query resulted in 16 documents on Web of Science. Four of those documents were part of the 10 relevant documents found using Scopus, leaving 12 new potentially relevant documents to consider. Four of those 12 documents used *large scale*, *parallel* or *distributed* terms in context to the *learning to rank*, *learning-to-rank* or *machine learned ranking*, none of them focused on parallel or distributed learning of ranking functions.

On Google Scholar, the defined search query resulted in 3300 documents. Because it infeasible to evaluate all 3300 studies we focus on the first 300 search results.

Backward reference search resulted in 10 studies regarded as potentially relevant based on the title, of which four were actually relevant and included in the literature description. Forward reference search resulted in 10 potentially relevant titles, of which seven studies turned out to be relevant.

Research in scaling up the training phase of Learning to Rank models can be categorised according to the approach in scaling up. Figure 3 illustrates the categories of scalable training approaches in Learning to Rank. The numbers in Figure 3 correspond to the sections that describe the related work belonging to these categories.

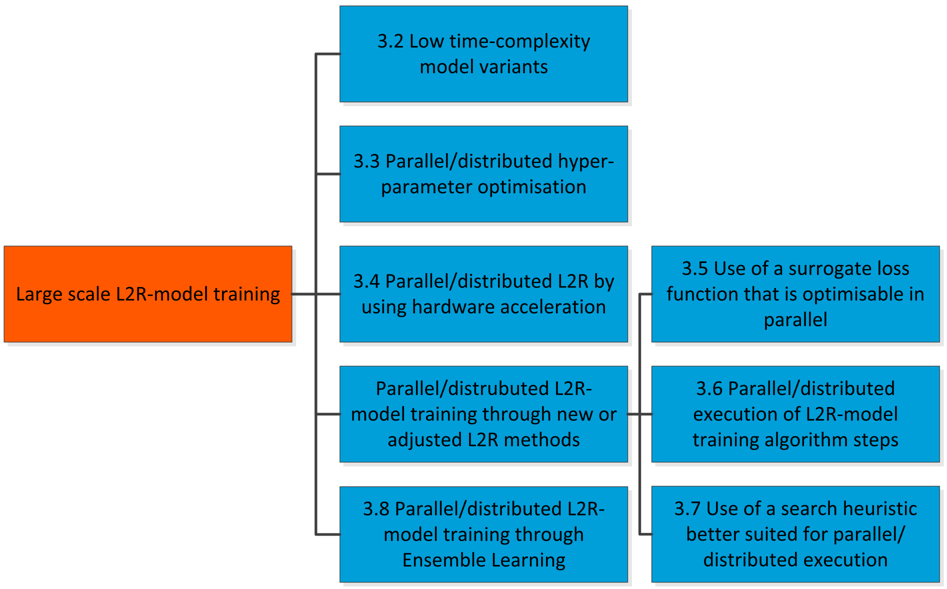
Large scale L2R-model training
3.2 Low time-complexity model variants
3.3 Parallel/distributed hyper-parameter optimisation
3.4 Parallel/distributed L2R by using hardware acceleration
Parallel/distrubuted L2R-model training through new or adjusted L2R methods
3.8 Parallel/distributed L2R-model training through Ensemble Learning
3.5 Use of a surrogate loss function that is optimisable in parallel
3.6 Parallel/distributed execution of L2R-model training algorithm steps
3.7 Use of a search heuristic better suited for parallel/ distributed execution

Figure 3: Categorisation of research on large scale training of Learning to Rank models

3.2 LOW COMPUTATIONAL COMPLEXITY LEARNING TO RANK

Regularised Least-Squares

One approach for handling large volumes of training data for Learning to Rank is through design of low time complexity Learning to Rank methods. Pahikkala et al. [154] described a pairwise Regularised Least-Squares (RLS) type of ranking function, RankRLS, with low time complexity. Airola et al. [8] further improved the training time complexity of RankRLS to $\mathcal{O}(tms)$, where t is the number of needed iterations, m the number of training documents and s the number of features. The RankRLS ranking function showed ranking performance similar to RankSVM [101, 110] on the BioInfer corpus [166], a corpus for information extraction in the biomedical domain.

Support Vector Machine

Gradient Boosted Decision Tree

Airola et al. [9] and Lee and Lin [125] both described lower time complexity methods to train a linear kernel ranking Support Vector Machine (SVM) [101, 110]. Lee and Lin [125] observed that linear kernel RankSVMs are inferior in accuracy compared to nonlinear kernel RankSVMs and Gradient Boosted Decision Tree (GBDT)s and are mainly useful to quickly produce a baseline model. Details of the lower time complexity version of the linear kernel RankSVM will not be discussed as it is shown to be an inferior Learning to Rank method in terms of accuracy.

Learning to Rank methods that are specifically designed for their low computational complexity, like RankRLS and the linear kernel RankSVM methods described in this section, are generally not among the top achieving models in terms of accuracy. From results on benchmarks and competitions it can be observed that the best generalisation accuracy are often more complex ones. This makes low time complexity models as a solution for large scale Learning to Rank less applicable and increases the relevance of the search for efficient training of more complex Learning to Rank models.

3.3 DISTRIBUTED HYPERPARAMETER TUNING OF LEARNING TO RANK MODELS

Hyperparameter optimisation is the task of selecting the combination of hyperparameters such that the Learning to Rank model shows optimal generalisation accuracy. Ganjisaffar et al. [87, 85] observed that long training times are often a result of hyperparameter optimisation, because it results in training multiple Learning to Rank models. Grid search is the *de facto* standard of hyperparameter optimisation and is simply an exhaustive search through a manually specified subset of hyperparameter combinations. The authors show how to perform parallel grid search on MapReduce clusters, which is easy because grid search is an embarrassingly parallel method as hyperparameter combinations are mutually independent. They apply their grid search on MapReduce approach in a Learning to Rank setting to train a LambdaMART [234] ranking model, which uses the Gradient Boosting [84] ensemble method combined with regression tree weak learners. Experiments showed that the solution scales linearly in number of hyperparameter combinations. However, the risk of overfitting grows overwhelmingly as the number of hyperparameter combinations grow, even when validation sets grow large.

Burges et al. [35] described their Yahoo! Learning to Rank Challenge submission that was built by performing an extensive hyperparameter search on a 122-node Message Passing Interface (MPI) cluster, running Microsoft HPC Server 2008. The hyperparameter optimisation was performed on a linear combination ensemble of eight LambdaMART models, two LambdaRank models and two MART models using a logistic regression cost. This submission achieved the highest Expected Reciprocal Rank (ERR) score of all Yahoo! Learning to Rank Challenge submissions.

Message Passing Interface

Expected Reciprocal Rank

Notice that methods described in this section train multiple Learning to Rank models at the same time to find the optimal set of parameters for a model, but that the Learning to Rank models itself are still trained sequentially. In the next sections we will present literature focusing on training Learning to Rank models in such a way that steps in this training process can be executed simultaneously.

3.4 HARDWARE ACCELERATED LEARNING TO RANK

Hardware accelerators are special purpose processors designed to speed up compute-intensive tasks. An Field-Programmable Gate Array (FPGA) and a Graphical Processing Unit (GPU) are two different types of hardware that can achieve better performance on some tasks though parallel computing. In general, FPGAs provide better performance while GPUs tend to be easier to program

Field-Programmable Gate Array

Graphical Processing Unit

[48]. Some research has been done in parallelising Learning to Rank methods using hardware accelerators.

3.4.1 *FPGA-based parallel Learning to Rank*

Single Instruction Multiple Data

Yan et al. [242, 243, 244, 245] described the development and incremental improvement of a Single Instruction Multiple Data (SIMD) architecture FPGA designed to run, the Neural Network based LambdaRank Learning to Rank algorithm. This architecture achieved a 29.3X speed-up compared to the software implementation, when evaluated on data from a commercial search engine. The exploration of FPGA for Learning to Rank showed additional benefits other than the speed-up originally aimed for. In their latest publication [245] the FPGA based LambdaRank implementation showed it could achieve up to 19.52X power efficiency and 7.17X price efficiency for query processing compared to Intel Xeon servers currently used at the commercial search engine.

Xu et al. [238, 239] designed an FPGA-based accelerator to reduce the training time of the RankBoost algorithm [81], a pairwise ranking function based on Freund and Schapire's AdaBoost ensemble learning method [82]. Xu et al. [239] state that RankBoost is a Learning to Rank method that is not widely used in practice because of its long training time. Experiments on MSN search engine data showed the implementation on a FPGA with SIMD architecture to be 170.6x faster than the original software implementation [238]. In a second experiment in which a much more powerful FPGA accelerator board was used, the speed-up even increased to 1800x compared to the original software implementation [239].

3.4.2 *GPGPU for parallel Learning to Rank*

General-Purpose computing on Graphical Processing Units

Computing Unified Device Architecture

Wang et al. [221] experimented with a General-Purpose computing on Graphical Processing Units (GPGPU) approach for parallelising RankBoost. Nvidia Computing Unified Device Architecture (CUDA) and ATI Stream are the two main GPGPU computing platform and are released by the two main GPU vendors Nvidia and AMD. Experiments show a 22.9x speed-up on Nvidia CUDA and a 9.2x speed-up on ATI Stream.

De Sousa et al. [67] proposed a GPGPU approach to improve both training time and query evaluation through GPU use. An association rule based Learning to Rank approach, proposed by Veloso et al. [215], has been implemented using the GPU in such a way that the set of rules van be computed simultaneously for each document. A speed-up of 127X in query processing time is reported based on evaluation on the LETOR data set. The speed-up achieved at learning the ranking function was unfortunately not stated.

3.5 PARALLEL EXECUTION OF LEARNING TO RANK ALGORITHM STEPS

Some research focused on parallelising the steps Learning to Rank algorithms that can be characterised as strong learners. Tyree et al. [211] described a way of parallelising GBDT models for Learning to Rank where the boosting step is still executed sequentially, but instead the construction of the regression trees themselves is parallelised. The parallel decision tree building is based on Ben-Haim and Yom-Tov's work on parallel construction of decision trees for classification [20], which are build layer-by-layer. The calculations needed for building each new layer in the tree are divided among the nodes, using a master-worker paradigm. The data is partitioned and the data parts are divided between the workers, who compress their share into histograms and send these to the master. The master uses those histograms to approximate the split and build the next layer. The master then communicates this new layer to the workers who can use this new layer to compute new histograms. This process is repeated until the tree depth limit is reached. The tree construction algorithm parallelised with this master-worker approach is the well-known Classification and Regression Trees (CART) [28] algorithm. Speed-up experiments on the LETOR and the Yahoo! Learning to Rank challenge data sets were performed. This parallel CART-tree building approach showed speed-up of up to 42x on shared memory machines and up to 25x on distributed memory machines.

Classification and Regression Trees

3.5.1 *Parallel ListNet using Spark*

Shukla et al. [188] explored the parallelisation of the well-known ListNet Learning to Rank method using Spark, which is a parallel computing model that is designed for cyclic data flows which makes it more suitable for iterative algorithms. Spark is incorporated into Hadoop since Hadoop 2.0. The Spark implementation of ListNet showed near linear training time reduction.

3.6 PARALLELISABLE SEARCH HEURISTICS FOR LISTWISE RANKING

Direct minimisation of ranking metrics is a hard problem due to the non-continuous, non-differentiable and non-convex nature of the Normalized Discounted Cumulative Gain (NDCG), ERR and Mean Average Precision (MAP) evaluation metrics. This optimisation problem is generally addressed either by replacing the ranking metric with a convex surrogate, or by heuristic optimisation methods such as Simulated Annealing or a Evolutionary Algorithm (EA). One EA heuristic optimisation method that is successfully used in direct rank evaluation functions optimisation is the Genetic Algorithm (GA) [247]. GAs are search heuristic functions that mimic the process of natural selection, consisting of mutation and cross-over steps [103]. The following subsection describe related work that uses search heuristics for parallel/distributed training.

Normalized Discounted Cumulative Gain
Mean Average Precision
Evolutionary Algorithm
Genetic Algorithm

3.6.1 *Immune Programming*

Immune Programming
Genetic Programming

Wang et al. [228] proposed a Immune Programming (IP) solution to direct ranking metric optimisation. IP [146] is, like Genetic Programming (GP) [117], a paradigm in the field of evolutionary computing, but where GP draws its inspiration from the principles of biological evolution, IP draws its inspiration from the principles of the adaptive immune system. Wang et al. [228] observed that all EAs, including GP and IP are generally easy to implement in a distributed manner. However, no statements on the possible speed-up of a distributed implementation of the IP solution has been made and no speed-up experiments have been conducted.

3.6.2 *CCRank*

Cooperative Coevolution

Wang et al. [225, 227] proposed a parallel evolutionary algorithm based on Cooperative Coevolution (CC) [165], which is, like GP and IP, another paradigm in the field of evolutionary computing. The CC algorithm is capable of directly optimizing non-differentiable functions, as NDCG, in contrary to many optimization algorithms. the divide-and-conquer nature of the CC algorithm enables parallelisation. CCRank showed an increase in both accuracy and efficiency on the LETOR 4.0 benchmark data set compared to its baselines. However, the increased efficiency was achieved through speed-up and not scale-up. Two reasons have been identified for not achieving linear scale-up with CCRank: 1) parallel execution is suspended after each generation to perform combination in order to produce the candidate solution, 2) Combination has to wait until all parallel tasks have finished, which may spend different running time.

3.6.3 *NDCG-Annealing*

Karimzadeghan et al. [112] proposed a method using Simulated Annealing along with the Simplex method for its parameter search. This method directly optimises the often non-differentiable Learning to Rank evaluation metrics like NDCG and MAP. The authors successfully parallelised their method in the MapReduce paradigm using Hadoop. The approach showed to be effective on both the LETOR 3.0 data set and their own data set with contextual advertising data. Unfortunately their work does not directly report on the speed-up obtained by parallelising with Hadoop, but it is mentioned that further work needs to be done to effectively leverage parallel execution.

3.7 PARALELLY OPTIMISABLE SURROGATE LOSS FUNCTIONS

3.7.1 *Alternating Direction Method of Multipliers*

Alternating Direction Method of Multipliers

Duh et al. [77] proposed the use of Alternating Direction Method of Multipliers (ADMM) for the Learning to Rank task. ADMM is a general optimization method that solves problems of the form

$$\begin{aligned}&\text{minimize } f(x) + g(x)\\&\text{subject to } Ax + Bz = c\end{aligned}$$

by updating x and z in an alternating fashion. It holds the nice properties that it can be executed in parallel and that it allows for incremental model updates without full retraining. Duh et al. [77] showed how to use ADMM to train a RankSVM [101, 110] model in parallel. Experiments showed the ADMM implementation to achieve a 13.1x training time speed-up on 72 workers, compared to training on a single worker.

Another ADMM Learning to Rank based approach was proposed by Boyd et al. [26]. They implemented an ADMM based Learning to Rank method in Pregel [140], a parallel computation framework for graph computations. No experimental results on parallelisation speed-up were reported on this Pregel based approach.

3.7.2 *Bregman Divergences and Monotone Retargeting*

Acharyya et al. [2, 1] proposed a Learning to Rank method searches for an order preserving transformation (*monotone retargeting*) of the target scores that is easier for a regressor to fit. This approach is based on the observation that it is not necessary to fit scores exactly, since the evaluation is dependent on the order and not on the pointwise predictions themselves.

Bregman divergences are a family of distance-like functions that do not satisfy symmetry nor the triangle inequality. A well-known member of the class of Bregman divergences is Kullback-Leibler divergence, also known as *information gain*.

Acharyya et al. [2, 1] defined a parallel algorithm that optimises a Bregman divergence function as surrogate of NDCG that is claimed to be well suited for implementation of a GPGPU. No experiments on speed-up have been performed.

3.7.3 *Parallel robust Learning to Rank*

Robust learning [106] is defined as the task to lean a model in the presence of outliers. Yun et al. described a [250] robust Learning to Rank model called RoBiRank that has the additional advantage that it is executable in parallel. RoBiRank uses parameter expansion to linearise a surrogate loss function, after which the elements of the linear combination can be divided over the available nodes. The Parameter expansion trick was proposed by Gopal and Yang [96], who originally proposed it for multinomial logistic regression. Unfortunately, no speed-up experiments were mentioned for the RoBiRank method, since Yun et al. focussed their research more on robust ranking than on parallel ranking. The only reference to performance of RoBiRank in terms of speed is the statement that its training time on a computing cluster is comparable to the more efficient implementation by Lee and Lin [125] of a linear kernel RankSVM [101, 110] described in section 3.2.

3.7.4 *Distributed Stochastic Gradient Descent*

Stochastic Gradient Descent

Long et al. [137] described special case of their pairwise cross-domain factor Learning to Rank method using distributed optimization of Stochastic Gradient Descent (SGD) based on Hadoop MapReduce. Parallelisation of the SGD optimization algorithm was performed using the MapReduce based method described by Zinkevich et al. [258] has been used. Real world data from Yahoo! has been used to show that the model is effective. Unfortunately the speed-up obtained by training their model in parallel is not reported.

3.8 ENSEMBLE LEARNING FOR PARALLEL LEARNING TO RANK

Schapire proved in 1990 that a strong model can be generated by combining weak models through a procedure called boosting [183]. The invention of the boosting method resulted in an increasing focus in machine learning research on methods to combine multiple weak models, which are called ensemble learning methods. Well-known ensemble methods include gradient boosting [83], bagging [29], AdaBoost [82] and stacked generalisation (often called stacking) [231].

Gradient Boosted Decision Tree

Ensemble methods can be used to parallelise learning methods by training the weak models in the ensemble on different nodes. Parallelisation of the training phase of Learning to Rank models is often achieved through ensemble learning, mainly boosting, and combined with decision trees, jointly called Gradient Boosted Decision Tree (GBDT). GBDTs are shown to be able to achieve good accuracy in a Learning to Rank setting when used in a pairwise [253] or listwise [50] setting.

3.8.1 *Gradient Boosting*

Gradient Boosting [83] is an ensemble learning method in which multiple weak learners are iteratively added together into one ensemble model in such a way that new models focus more on those data instances that were misclassified before.

Kocsis et al. [115] proposed a way to train multiple weak learners in parallel by extending those models that are likely to yield a good model when combined through boosting. Authors showed through theoretical analysis that the proposed algorithm asymptotically achieve equal performance to regular gradient boosting. GBDT models could be trained in parallel using their BoostingTree algorithm.

Ye et al. [246] described how to implement the stochastic GBDT model in a parallel manner using both MPI and Hadoop. Stochastic GBDT differs from regular GBDT models by using stochastic gradient boosting instead of regular gradient boosting. Stochastic gradient boosting is a minor alteration to regular gradient boosting that, at each iteration of the algorithm, trains a base learner on a subsample of the training set that is drawn at random without replacement [84]. Experiments showed the Hadoop implementation to resulted in too expensive communication cost to be useful. Authors believed that these high communication costs were a result of the communication intensive implementation that was not well suited for the MapReduce paradigm. The MPI approach proved to be successful and obtained near linear speed-ups.

Svore and Burges [200, 199] designed two approaches to train the LambdaMART [234] model in a distributed manner. Their first approach is applicable in case the whole training set fits in main memory on every node, in that case the tree split computations of the trees in the LambdaMART tree-ensemble are split instead of the data. The second approach distributes the training data and corresponding training computations and is therefore scalable to high amounts of training data. Both approaches achieves a six times speed-up over LambdaMART on 32 nodes compared to a single node.

3.8.2 *Boosting wrapped in Bagging*

Some approaches combine both boosting and bagging. Bagging [29], also called bootstrap aggregating, is a ensemble learning method in which m training sets $D_1..D_m$ are constructed from the training set D by uniformly sampling data items from D with replacement. The bagging method is parallelisable by training each D_i with $i \in \{1..m\}$ on a different node.

Yandex researchers Pavlov et al. [159] were the first to proposed a boosting-wrapped-in-bagging approach, which they called BagBoo. The boosting step itself is not parallelisable, but the authors state that learning a short boosted sequence on a single node is still a do-able task.

Ganjisaffar et al. [86, 85] proposed a pairwise boosting-wrapped-in-bagging model called BL-MART, contrasting the pointwise BagBoo model. BL-MART adds a bagging step to the LambdaMART [234] ranking model, that uses the Gradient Boosting [84] ensemble method combined with regression tree weak learners. In contract to BagBoo, BL-MART is limited in the number of trees. An experiment on the TD2004 data set resulted in 4500 trees using BL-MART while 1.1 million trees were created with the BagBoo model.

3.8.3 *Stacked generalisation*

Deep Stacking Network

A Deep Stacking Network (DSN) is a processing architecture developed from the field of *Deep Learning*. DSNs are based on the stacked generalization (also known as stacking) ensemble learning method [231], which is a multi-layer approach where multiple learning models are stacked on top of each other in such a way that the outputs of the models are used as the input features for the models in the next layer of the stack. Stacked generalisation models are easy to compute in parallel, as the models within the same layer can be trained independently. Deng et al. [69] used a deep architecture of stacked generalisation (DSN) to make pointwise relevance label predictions such that the Mean Squared Error (MSE) of the predicted relevance labels as compared to the ground truth relevance labels are optimised. No speed-up results of parallel computation are reported, although it is stated repeatedly that the method is scalable and easy to compute in parallel

Mean Squared Error

3.8.4 *Randomisation*

Most ensemble learning based Learning to Rank methods are based on boosting. Although boosting has shown to be an effective method in Learning to Rank, it has the drawback that the models in the ensemble are not completely mutually independent, which makes the process harder to parallelise. Geurts and Louppe [94] proposed a tree-based ensemble method in which the trees are built using random subsets of the available features. The authors called this method Extremely Randomised Trees (ET) and originally proposed this method for classification and regression settings instead of ranking settings [93]. The ET algorithm is similar to the well-known Random Forest algorithm, but with two differences: 1) in ET each tree is build on the complete data set instead of random subsets of the data and 2) ET sets the discretisation thresholds that define the splits in the tree randomly, instead of based on the sample. The randomisations in the process make the models in the ensemble mutually independent, which makes it trivial to parallelise by training each tree on a different node. In contrary to what one might think, ETs show very good performance on benchmark data with a 10th place in the Yahoo! Learning to Rank Challenge.

Extremely Randomised Trees

4 BENCHMARK DATA SETS

Benchmark data sets enable fair comparisons of ranking models over a fixed set of documents and queries. The approach of using fixed benchmark data sets to compare the performance of Information Retrieval systems under equal circumstances has been the standard evaluation methodology in the Information Retrieval field since the release of the Cranfield collection [56].

This chapter describes benchmark characteristics, e.g. collection size and features, of Learning to Rank benchmark collections and data sets and gives an overview of the performance of baseline algorithms on these collections and data sets.

The accuracies of the Learning to Rank methods described in the following sections can only be compared within the benchmark and not between benchmarks for the following reasons:

1. Differences in feature sets between data sets detract from fair comparison
2. Although the Normalized Discounted Cumulative Gain (NDCG) definition is unambiguous (note: there are two existing versions of NDCG: Järvelin and Kekäläinen [109] and Burges et al. [32], but both are in itself defined unambiguously), Busa-Fekete et al. [37] found that NDCG evaluation tools of different benchmark data sets produced different scores

Normalized Discounted Cumulative Gain

4.1 YAHOO! LEARNING TO RANK CHALLENGE

Yahoo!'s observation that all existing benchmark data sets were too small to draw reliable conclusions prompted Yahoo! to release two internal data sets from Yahoo! search. Data sets used at commercial search engines are many times larger than available benchmark data sets. Yahoo! published a subset of their own commercial training data and launched a Learning to Rank competition based on this data. The Yahoo! Learning to Rank Challenge [44] is a public Learning to Rank competition which took place from March to May 2010, with the goal to promote the data sets and encourage the research community to develop new Learning to Rank algorithms.

The Yahoo! Learning to Rank Challenge consists of two tracks that uses the two data sets respectively: a standard Learning to Rank track and a transfer learning track where the goal was to learn a specialized ranking function that can be used for a small country by leveraging a larger training set of another country. For this experiment, we will only look at the standard Learning to Rank data set, because transfer learning is a separate research area that is not included in this thesis.

	Train	Validation	Test
# of queries	19,994	2,994	6,983
# of documents	473,134	71,083	165,660
# of features	519	519	519

Table 4: Yahoo! Learning to Rank Challenge data set characteristics, as described in the overview paper [44]

Expected Reciprocal Rank

Both NDCG and Expected Reciprocal Rank (ERR) are measured as performance metrics, but the final standings of the challenge were based on the ERR values. Model validation on the Learning to Rank methods participating in the challenge is performed using a train/validation/test-set split following the characteristics shown in Table 4. Competitors could train on the training set and get immediate feedback on their performance on the validation set. The test set performance is used to create the final standings and is only measured after the competition has ended to avoid overfitting on the test set. The large number of documents, queries and features compared to other benchmark data sets makes the Yahoo! Learning to Rank Challenge data set interesting. Yahoo! did not provide detailed feature descriptions to prevent competitors to get detailed insight in the characteristics of the Yahoo! data collection and features used at Yahoo!. Instead high level descriptions of feature categories are provided. The following categories of features are described in the challenge overview paper [44]:

WEB GRAPH Quality and popularity metrics of web documents, e.g. PageRank [152].

DOCUMENT STATISTICS Basic document statistics such as the number of words and url characteristics.

DOCUMENT CLASSIFIER Results of various classifiers on the documents. These classifiers amongst others include: spam, adult, language, main topic, and quality classifiers.

QUERY Basic query statistics, such as the number of terms, query frequency, and click-through rate.

Term Frequency - Inverse Document Frequency

TEXT MATCH Textual similarity metrics between query and document. Includes Term Frequency - Inverse Document Frequency (TF-IDF), BM25 [179] and other metrics for different sections of the document.

TOPICAL MATCHING These features go beyond similarity at word level and compute similarity on topic level. For example by classifying both the document and the query in a large topical taxonomy.

CLICK Click-based user feedback.

EXTERNAL REFERENCES Document meta-information such as Delicious[1] tags

1 https://delicious.com/

	Authors	ERR
1	Burges et al. (Microsoft Research)	0.46861
2	Gottschalk (Activision Blizzard) & Vogel (Data Mining Solutions)	0.46786
3	Parakhin (Microsoft)	0.46695
4	Pavlov & Brunk (Yandex Labs)	0.46678
5	Sorokina (Yandex Labs)	0.46616

Table 5: Final standings of the Yahoo! Learning to Rank Challenge, as presented in the challenge overview paper [44]

TIME Document age and historical in- and outlink data that might help for time sensitive queries.

4.1.1 *Results*

1055 teams send in at least one submission to the Yahoo! Learning to Rank challenge. The top eight participants of the Yahoo! Learning to Rank challenge all used decision trees combined with ensemble methods. The mainly used ensemble method within these top performers is boosting. The combination of boosting with decision tree learners is often called Gradient Boosted Decision Tree (GBDT). Figure 5 shows the top five participants in the Yahoo! Learning to Rank Challenge in terms of ERR score. Burges [35] created a linear combination ensemble of eight LambdaMART [33], two LambdaRank and two Logistic Regression models. Gottschalk and Vogel used a combination of RandomForest models and GBDT models. Pavlov and Brunk used a regression based model using the BagBoo [159] ensemble technique, which combines bagging and boosting. Sorokina used a similar combination of bagging and boosting that is called Additive Groves [193].

Gradient Boosted Decision Tree

The challenge overview paper [44] states as one of the lessons of the challenge that the simple baseline GBDT model performed very well with a small performance gap to the complex ensemble submissions at the top of the table.

Although the winning GBDT models performs very well in terms of NDCG their high complexity makes them unsuitable to use in production. The winning models take weeks to train and are very slow during query evaluation. An exception in training time is the BagBoo [159] method used by Pavlov & Brunk. The bagging component of the BagBoo method enables it to achieve high scalability through parallelism. Pavlov et al. [159] managed to train half a million trees on 200 nodes in 2 hours.

4.2 LETOR

The LETOR benchmark set was first released by Microsoft Research Asia in April 2007 [136] to solve the absence of a experimental platform for Learning to Rank at that time. LETOR has been updated several times since: LETOR 2.0 was released at the end of 2007, LETOR 3.0 [168] in December 2008 and LETOR 4.0 [169] in July 2009.

4.2.1 *LETOR 2.0*

LETOR 2.0 consists of three data sets: the OHSUMED data set and two data sets of the on the .gov collection. The OHSUMED collection is a subset of the medical publication database MEDLINE and contains medical publications from 270 journals that were published between 1987 and 1991. The .gov collection is a web crawl obtained in January 2002, which was used for the Text REtrieval Conference (TREC) web track in 2003 and 2004.

Text REtrieval Conference

Different query sets exists for the .gov corpus. Those query sets are categorized in the following categories [59]:

TOPIC DISTILLATION (TD) these queries involve finding relevant pages given a broad query. E.g., the query 'cotton industry' which is entered with the aim to find pages that provide information about the cotton industry.

NAMED PAGE FINDING (NP) in these queries the user asks for one specific page by name. E.g., the user queries for 'ADA Enforcement' to find the page $http://www.usdoj.gov/crt/ada/enforce.htm$.

HOME PAGE FINDING (HP) these queries are similar to NP queries in that the user is looking for one specific page, but now this specific page is a homepage. E.g., the user queries for 'Tennessee Valley Authority' to find the homepage $http://www.tva.gov$.

LETOR 2.0 only uses the topic distillation queries. Baseline Learning to Rank algorithms evaluated on LETOR 2.0 by the organisation are:

PAIRWISE

- RankSVM [101, 110]
- RankBoost [81]
- Multiple Hyperplane Ranker (MHR) [174]
- FRank [210]

Multiple Hyperplane Ranker

LISTWISE

- ListNet [39]
- AdaRank-MAP [236]
- AdaRank-NDCG [236]

4.2.1.1 *Results*

Table 6 shows the performance of the baseline methods on the data sets in the LETOR 2.0 collection. ListNet can be regarded as the winner of the comparison as it outperformed the other methods on two of the three data sets and ranked third on the third data set. MHR was only evaluated on the OHSUMED data set, on which it performed second best, after ListNet.

	OHSUMED	TD2003	TD2004
RankBoost	0.4356	0.2851	0.4716
RankSVM	0.4411	0.3410	0.4201
FRank	0.4423	0.3357	0.4708
ListNet	0.4489	0.3743	0.4579
AdaRank-MAP	0.4385	0.1940	0.4063
AdaRank-NDCG	0.4369	0.2702	0.3878
MHR	0.4423	-	-

Table 6: NDCG@10 results of the baseline methods on LETOR 2.0

4.2.2 *LETOR 3.0*

The LETOR 3.0 benchmark collection [168] as released in 2007 contained two data sets: the OHSUMED collection and the .gov collection. Tables 23 and 24 in Appendix A provide the descriptions of the features of the OHSUMED and the .gov collections for LETOR 3.0 respectively. Where LETOR 2.0 only used topic distillation (TD) queries on the .gov corpus, LETOR 3.0 also uses named page finding (NP) and home page finding (HP) queries. With query sets available from both the TREC 2003 and 2004 conferences there are six query sets in total: TD2003, TD2004, NP2003, NP2004, HP2003 and HP2004. It is noteworthy that the OHSUMED, TD2003 and TD2004 data sets of LETOR 2.0 and LETOR 3.0 are not identical and therefore not comparable due to differences in sampling method.

The evaluation metrics used are NDCG and Mean Average Precision (MAP). The *winning number* metric is defined as the number of other algorithms that it can beat over all of the seven data sets (six .gov sets + OHSUMED). The LETOR organisation implemented and evaluated several well-known Learning to Rank algorithms themselves and in addition gathers and publications and results of new algorithms evaluated on the LETOR benchmark. The baseline Learning to Rank algorithms evaluated by the LETOR team are:

Mean Average Precision

POINTWISE

- Linear regression

PAIRWISE

- RankSupport Vector Machine (SVM) [101, 110]
- RankBoost [81]
- FRank [210]

Support Vector Machine

LISTWISE

- ListNet [39]
- AdaRank-MAP [236]
- AdaRank-NDCG [236]
- SVM^{MAP} [249]

4.2.2.1 *Results*

The LETOR paper [168] describes the performance of the LETOR baseline methods. Figures 4 and 5 show ListNet to be the best performing baseline in terms of winning number on both NDCG and MAP. The LETOR website lists[2] a few algorithms that have since been evaluated on the LETOR benchmark collection.

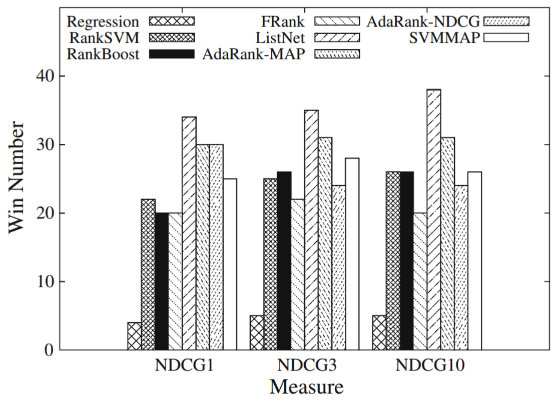

Figure 4: Comparison of ranking accuracy across the seven data sets in LETOR by NDCG, obtained from Qin et al. [168]

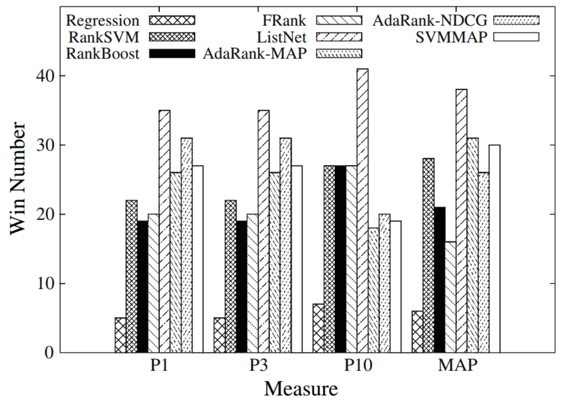

Figure 5: Comparison across the seven data sets in LETOR by MAP, obtained from Qin et al. [168]

We will describe the performance of the algorithms that were evaluated by the LETOR team on LETOR 3.0 after publication of the LETOR paper by Qin et al. [168], as listed on the LETOR website[2] by comparing their performance with the ListNet baseline. We will consider those methods to be better than ListNet when they beat the ListNet baseline in at least four of the seven data sets in NDCG value. Note that this does not necessarily imply that these methods would have scored a higher NDCG winning number than ListNet. Table 7 shows the ListNet performance on the seven LETOR data sets in terms of NDCG@10 and MAP.

2 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor3baseline.aspx

	NDCG@10	MAP
TD2003	0.348	0.2753
TD2004	0.317	0.2231
NP2003	0.801	0.6895
NP2004	0.812	0.6720
HP2003	0.837	0.7659
HP2004	0.784	0.6899
OHSUMED	0.441	0.4457

Table 7: Performance of ListNet on LETOR 3.0

Since LETOR is arguably the most well-known benchmark collection in the field, it is conceivable that creators of new Learning to Rank methods evaluate their new method on the LETOR collection to show how well their new method works.

	Ridge Regression	RankSVM-Primal	RankSVM-Struct	SmoothRank	ListNet
TD2003	0.3297	0.3571	0.3467	0.3367	0.348
TD2004	0.2832	0.2913	0.3090	0.3343	0.317
NP2003	0.8025	0.7894	0.7955	0.7986	0.801
NP2004	0.8040	0.7950	0.7977	0.8075	0.812
HP2003	0.8216	0.8180	0.8162	0.8325	0.837
HP2004	0.7188	0.7720	0.7666	0.8221	0.784
OHSUMED	0.4436	0.4504	0.4523	0.4568	0.441
# winning data sets	2	2	1	3	-

Table 8: NDCG@10 comparison of algorithms recently evaluated on LETOR 3.0 with the ListNet baselines

4.2.3 *LETOR 4.0*

The LETOR 4.0 benchmark collection[3] consists of the Gov-2 document collection and two query data sets from the Million Query Track at TREC 2007 (MQ2007) and TREC 2008 (MQ2008). LETOR 4.0 consists of a semi-supervised ranking task, a rank aggregation task and a listwise ranking task next to the supervised ranking task. Table 9 shows the collection characteristics in number of queries, documents and features for LETOR 4.0. Evaluation on this collection is performed using a five-fold cross-validation, where partitioning of the

3 http://http://research.microsoft.com/en-us/um/beijing/projects/letor/

Data set	MQ2007	MQ2008
Queries	1692	784
Documents	69,622	15,211
Features	46	46

Table 9: Characteristics of the LETOR 4.0 collection

Model	Data set	Mean NDCG	NDCG@10
RankSVM-Struct	MQ2007	0.4966	0.4439
	MQ2008	0.4832	0.2279
ListNet	MQ2007	0.4988	0.4440
	MQ2008	0.4914	0.2303
AdaRank-NDCG	MQ2007	0.4914	0.4369
	MQ2008	0.4950	0.2307
AdaRank-MAP	MQ2007	0.4891	0.4335
	MQ2008	0.4915	0.2288
RankBoost	MQ2007	0.5003	0.4464
	MQ2008	0.4850	0.2255

Table 10: Comparison of the LETOR 4.0 baseline models

data into folds was performed beforehand by the creators of the MQ2007 and MQ2008 data sets. Documents in the data set were

4.2.3.1 *Results*

RankSVM-Struct, ListNet, AdaRank-NDCG, AdaRank-MAP and RankBoost were used as baseline models on the LETOR 4.0 data set and were implemented and evaluated by the publishers of LETOR 4.0. Table 10 shows the performance of those baseline models on the LETOR 4.0 benchmark collection.

BoostedTree model [115] showed an NDCG of 0.5071 on the MQ2007 data set and thereby beat all baseline models.

4.3 OTHER DATA SETS

LETOR and the Yahoo! Learning to Rank Challenge data sets are the most used benchmark collections in the field. Several other benchmark collections for the Learning to Rank task have been proposed.

4.3.1 *MSLR-WEB10/30k*

MSLR-WEB30k and the smaller subset MSLR WEB10k are two large data sets with 30,000 and 10,000 queries respectively. The data was obtained from the

Microsoft Bing[4] commercial web search engine. A feature list and feature descriptions are available. The data set only includes features that are commonly used in the research community, as proprietary features were excluded from the data set. Microsoft published these data sets as unofficial successor to LETOR in June 2010, but no baseline evaluations were described by the MSLR-WEB10/30k team. The Microsoft Research website from which the data set is obtainable[5] also offers an evaluation script for NDCG and MAP. The presence of an official evaluation script enables fair comparison of evaluation results from other researchers comparable.

4.3.2 *WCL2R*

The WCL2R collection, released by Alcântara et al. [10], contains of two data sets originating from the Chilean search engine TodoCL[6]. Both data sets contain approximately 3 million documents. WCL2R is the only benchmark collection known that provides click-through data on user level. The collection contains 277,095 queries and logs of in total 1,5 million clicks by 16,829 users. The WCL2R paper provides evaluation results for the well-known RankSVM [101, 110] and RankBoost [81] algorithms. In addition two ranking methods developed at the same university of the WCL2R paper were evaluated: LAC [215] and a ranking algorithm based on Genetic Programming (GP) [66]. Table 11 shows the performance of those algorithms on the WCL2R collection. No evaluations of other algorithms on the WCL2R collection are known.

Genetic Programming

	FS			NC		
Algorithm	@1	@3	@10	@1	@3	@10
RankSVM	0.314	0.353	0.395	0.265	0.301	0.339
LAC	0.296	0.360	0.403	0.244	0.266	0.315
GP	0.288	0.344	0.396	0.221	0.262	0.318
RankBoost	0.295	0.328	0.375	0.247	0.264	0.305

Table 11: NDCG results of the baseline methods on the WCL2R collection, obtained from Alcântara et al. [10]

4.3.3 *AOL*

Pass et al. [158] describe an America Online Learning to Rank data set which they published in 2006. This data set is unique in that it is the only large commercial web search data set that contains user session and user behaviour information. The data set contains 20 million search keywords for over 650,000 users over a three month time period. AOL later realised the publication of the data set to be a mistake, as personally identifiable information turned out

4 http://www.bing.com/
5 http://research.microsoft.com/en-us/projects/mslr/download.aspx
6 www.todocl.cl

to be presented in some of the queries[7]. AOL acknowledged the mistake of publishing the data and removed the data from its website. The removal of the data was too late, as the data was already downloadable from several mirror sites[8]. This controversial AOL data set is to date still used for Learning to Rank research. No official baseline evaluation is provided by the AOL team.

4.3.4 *Yandex Internet Mathematics contest*

Russian commercial web search engine Yandex dedicated their yearly Internet Mathematics contest to the task of Learning to Rank in the 2009 edition of the contest. Features (245 in total) are only numbered and their semantics are not revealed, equivalent to the Yahoo! Learning to Rank Challenge. The set is split into a 45%-training, 10%-validation, and 45%-test data. The training set contains 9124 queries, with on average around 100 assessed documents per query. Yandex IMC 2009 has an online leaderboard[9] showing the best performing teams in the contest, but it is not traceable which methods each teams used. Pavlov et al. [159] in their paper claim to have won the Yandex Internet Mathematics contest 2009 with their BagBoo method.

7 http://select.nytimes.com/gst/abstract.html?res=F10612FC345B0C7A8CDDA10894DE404482
8 http://gregsadetsky.com/aol-data/
9 http://imat2009.yandex.ru/en/results

5 CROSS BENCHMARK COMPARISON

As we have seen in Chapter 4, the evaluation of Learning to Rank methods is spread over several benchmark data sets. However, as the Learning to Rank methods evaluated differs between benchmarks, no single benchmark comparison can be regarded as a conclusive argument on which Learning to Rank method is most accurate.

Several studies make a small start in considering Learning to Rank methods performance over multiple benchmark data sets. Gomes et al. [95] analysed ranking accuracy of a set of models on both LETOR 3.0 and LETOR 4.0. Busa-Fekete et al. [36] compared the accuracy of a small set of models over the LETOR 4.0 data sets, both MSLR data sets, both Yahoo! Learning to Rank Challenge data sets and the OHSUMED dataset from LETOR 3.0. To our knowledge, no structured meta-analysis on ranking accuracy has been conducted where evaluation results on several benchmark collections are taken into account. With a meta-analysis we will compare the performance of Learning to Rank methods across the Learning to Rank benchmark data sets described in foregoing sections.

5.1 COLLECTING EVALUATION RESULTS

With a literature review we will collect evaluation results on the data sets / collections. The following list presents an overview of the benchmark collections taken into account in the meta-analysis:

- LETOR 2.0
- LETOR 3.0
- LETOR 4.0
- Yahoo! Learning to Rank Challenge
- Yandex Internet Mathematics Competition 2009
- MSLR-web10/30k
- WCL2R
- AOL

For the LETOR collections, the evaluation results of the baseline models will be used from LETOR 2.0[1], LETOR 3.0[2] and LETOR 4.0[3] as listed on the LETOR

1 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor2.0/baseline.aspx
2 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor3baseline.aspx
3 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor4baseline.aspx

website.

LETOR 1.0, LETOR 3.0, Yahoo! Learning to Rank Challenge, WCL2R and AOL have accompanying papers which were published together with these benchmark collections. Users of those benchmark collections are encouraged to cite these papers. Therefore, we collect evaluation measurements of Learning to Rank methods on these benchmark collections through forward literature search. Table 12 presents an overview of the results of this forward literature search. Google Scholar will be used to perform the forward reference search.

Benchmark	Paper	# of forward references
LETOR 1.0 & 2.0	Liu et al. [136]	307
LETOR 3.0	Qin et al. [168]	105
Yahoo! Learning to Rank Challenge	Chapelle et al. [44]	102
AOL dataset	Pass et al. [158]	339
WCL2R	Alcântara et al. [10]	2

Table 12: Forward references of Learning to Rank benchmark papers

The LETOR 4.0, MSLR-web10/30k and Yandex Internet Mathematics Competition 2009 benchmark collections were not accompanied with a describing study. To collect measurements of Learning to Rank methods evaluated on these benchmarks, a Google Scholar search is performed on the name of the benchmark. Table 4 shows the results of this literature search.

Benchmark	Google Scholar search results
LETOR 4.0	75 results
MSLR-web10k	16 results
MSLR-web30k	15 results
Yandex Internet Mathematics Competition	1 result

Table 13: Google Scholar search results statistics for Learning to Rank benchmarks

5.2 COMPARISON METHODOLOGY

The LETOR 3.0 paper [168] states that it may differ between data sets what the most accurate ranking methods are. To evaluate the overall performance of Learning to Rank methods over the multiple data sets in the LETOR 3.0 collections, Qin et al. [168] proposed a measure called *winning number* as the number of other algorithms that an algorithm can beat over the set of data sets. Formally the winning number measure is defined as

$$\text{Winning Number}_i(M) = \sum_{j=1}^{n} \sum_{k=1}^{m} I_{\{M_i(j) > M_k(j)\}}$$

where j is the index of a dataset, n the number of data sets in the comparison, i and k are indices of an algorithm, $M_i(j)$ is the performance of the i-th algorithm on the j-th dataset, M is a ranking measure (such as Normalized Discounted Cumulative Gain (NDCG) or Mean Average Precision (MAP)), and $I_{\{M_i(j)>M_k(j)\}}$ is an indicator function such that

Normalized Discounted Cumulative Gain
Mean Average Precision

$$I_{\{M_i(j)>M_k(j)\}} = \begin{cases} 1 & \text{if } M_i(j) > M_k(j), \\ 0 & \text{otherwise} \end{cases}$$

In contrast to the winning number comparison on LETOR 3.0, there will not be accuracy measurements for each algorithm on each dataset in our meta-analysis. To compare algorithms based on a sparse set of evaluation measurements, a normalised version of the Winning Number metric will be used. This Normalised Winning Number (NWN) takes only those data sets into account that an algorithm is evaluated on and divides this by the theoretically highest Winning Number that an algorithm would have had in case it it would have been the most accurate algorithm on all data sets on which it has been evaluated. We will redefine the indicator function I in order to only take into account those data sets that an algorithm is evaluated on, as

Normalised Winning Number

$$I_{\{M_i(j)>M_k(j)\}} = \begin{cases} 1 & \text{if } M_i(j) \text{ and } M_k(j) \text{ are both defined and } M_i(j) > M_k(j), \\ 0 & \text{otherwise} \end{cases}$$

From now on this adjusted version of Winning Number will be references to as NWN. The mathematical definition of NWN is

$$\text{Normalised Winning Number}_i(M) = \frac{\text{Winning Number}_i(M)}{\text{Ideal Winning Number}_i(M)}$$

where Ideal Winning Number (IWN) is defined as

Ideal Winning Number

$$\text{Ideal Winning Number}_i(M) = \sum_{j=1}^{n} \sum_{k=1}^{m} D_{\{M_i(j),M_k(j)\}}$$

where j is the index of a dataset, n the number of data sets in the comparison, i and k are indices of an algorithm, $M_i(j)$ is the performance of the i-th algorithm on the j-th dataset, M is a ranking measure (such as NDCG or MAP), and $D_{\{M_i(j),M_k(j)\}}$ is an evaluation definition function such that

$$D_{\{M_i(j),M_k(j)\}} = \begin{cases} 1 & \text{if } M_i(j) \text{ and } M_k(j) \text{ are both defined,} \\ 0 & \text{otherwise} \end{cases}$$

NDCG@3, NDCG@5, NDCG@10 and MAP are chosen as metrics on which the meta-analysis will be performed. These metrics seem to be the most frequently used evaluation metrics in most of the used benchmark data sets. An exception is the Yahoo! Learning to Rank Challenge data sets on which mainly Expected Reciprocal Rank (ERR) is used as main evaluation metric. The lack of use of the ERR-metric in other benchmarks makes it unsuitable for a cross-benchmark comparison. By making the decision to include NDCG at three cut-off points and only a single MAP entry, we implicitly attain a higher weight for NDCG compared to MAP on an analysis that combines all measurements on the four

Expected Reciprocal Rank

metrics. This implicit weighting is arbitrary and therefore undesirable, but the number of algorithm evaluation results gained by this makes it a pragmatic approach.

5.3 EVALUATION RESULTS FOUND IN LITERATURE

Table 14 gives an overview of the Learning to Rank methods for which evaluation results were found for one or more of the benchmark data sets listed in section 5.1 through the literature review process also described in section 5.1. Occurrences of L2, L3 and L4 in Table 14 imply that these algorithms are evaluated as official LETOR 2.0, LETOR 3.0 and LETOR 4.0 baselines respectively.

Some studies with evaluation results found through in literature review were not usable for the meta-analysis. The following enumeration lists those properties that made one or more studies unusable for the meta-analysis. Between brackets are the studies that these properties apply to.

1. A different evaluation methodology was used in the study compared to what was used in other studies using the same benchmark [92, 130].
2. The study focussed on a different Learning to Rank task (e.g. rank aggregation or transfer ranking) [64, 63, 70, 65, 49, 7, 220, 61, 143, 102, 62, 76, 13, 167, 219, 71, 155, 128, 218, 60].
3. The study used an altered version of a benchmark that contained additional features [24, 73].
4. The study provides no exact data of the evaluation results (e.g. results are only in graphical form) [230, 223, 241, 118, 126, 235, 254, 232, 256, 112, 201, 156, 149, 55, 195, 162, 5, 43, 172, 3, 184, 105, 11, 196, 98, 21, 89, 51, 240, 187].
5. The study reported evaluation results in a different metric than the metrics chosen for this meta-analysis [248, 206, 154, 114, 144].
6. The study reported a higher performance on baseline methods than official benchmark runs [74, 18, 160, 192, 23, 22, 40, 2, 160, 209, 14]. Such cases are not necessarily caused by malicious intent, but might be caused by a mix-up of the two existing versions of the NDCG metric (Järvelin and Kekäläinen [109] and Burges et al. [32]).
7. The study did not report any baseline performance that allowed us to check validity of the results [41, 224, 31].

5.4 RESULTS & DISCUSSION

The following subsections provide the performance of Learning to Rank methods in terms of NWN for NDCG@3, NDCG@5, NDCG@10 and MAP. Performance of the Learning to Rank methods is plotted with NWN on the vertical axis and the number of data sets on which the method has been evaluated on the horizontal

Method	Described	Evaluated
AdaRank-MAP	[236]	L2, L3, L4
AdaRank-NDCG	[236]	L2, L3, L4, [36, 202]
ADMM	[77]	[77]
ApproxAP	[170]	[170]
ApproxNDCG	[170]	[170]
BagBoo	[159]	[86]
Best Single Feature		[95]
BL-MART	[86]	[86]
BoltzRank-Single	[217]	[217, 219]
BoltzRank-Pair	[217]	[217, 86, 219]
BT	[255]	[255]
C-CRF	[171]	[171]
CA	[142]	[36, 202]
CCRank	[226]	[226]
CoList	[88]	[88]
Consistent-RankCosine	[175]	[202]
DCMP	[176]	[176]
DirectRank	[202]	[202]
EnergyNDCG	[80]	[80]
FBPCRank	[121]	[121]
FenchelRank	[119]	[119, 120, 123]
FocusedBoost	[150]	[150]
FocusedNet	[150]	[150]
FocusedSVM	[150]	[150]
FP-Rank	[191]	[191]
FRank	[210]	L2, L3, [222]
FSMRank	[122]	[122, 123]
FSM^{SVM}	[122]	[122]
GAS-E	[91]	[122]
GP	[66]	[10]
GPRank	[190]	[207]
GRankRLS	[153]	[153]
GroupCE	[131]	[131]
GroupMLE	[133]	[131]
IntervalRank	[145]	[145, 80]
IPRank	[228]	[228, 207]
KeepRank	[52]	[52]
Kernel-PCA RankBoost	[75]	[75, 182]
KL-CRF	[216]	[216]
LAC-MR-OR	[215]	[215]
LambdaMART	[33]	[15, 86]
LambdaNeuralRank	[157]	[157]
LambdaRank	[34]	
LARF	[207]	[207]
Linear Regression	[57]	L3, [222, 216]
ListMLE	[235]	[133, 131, 88]
ListNet	[39]	L2, L3, L4
ListReg	[232]	[232]
LRUF	[208]	[208]
MCP	[123]	[123]
MHR	[174]	L2
MultiStageBoost	[111]	[111]
NewLoss	[161]	[161]
OWPC	[212]	[212]
PERF-MAP	[156]	[208]
PermuRank	[237]	[237]
Q.D.KNN	[90]	[229]
RandomForest		[95]
Rank-PMBGP	[182]	[182]
RankAggNDCG	[229]	[229]
RankBoost	[81]	L2, L3, L4, [36, 10]
RankCSA	[99]	[99]
RankDE	[25]	[182]
RankELM (pairwise)	[259]	[259]
RankELM (pointwise)	[259]	[259]
RankMGP	[130]	[130]
RankNet	[32]	[36, 157, 150]
RankRLS	[154]	[153]
RankSVM	[101, 110]	L2, L3, [36, 80, 99, 10]
RankSVM-Primal		L3, [121]
RankSVM-Struct		L3, L4
RCP	[78]	[78]
RE-QR	[214]	[214]
REG-SHF-SDCG	[233]	[233]
Ridge Regression	[57]	L3
RSRank	[198]	[119]
SmoothGrad	[124]	[202]
SmoothRank	[47]	L3, [47]
SoftRank	[203, 97]	[170]
SortNet	[178]	[178, 80]
SparseRank	[120]	[120]
SVD-RankBoost	[132]	[132]
SVM^{MAP}	[249]	L3, [222, 237, 150]
SwarmRank	[72]	[182]
TGRank	[119]	[119]
TM	[255]	[255, 157, 202]
VFLR	[38]	[38]

Table 14: An overview of Learning to Rank algorithms and their occurrence in evaluation experiments on benchmark data sets

axis. The further to the right, the more certain we can be about the performance of the Learning to Rank method. The methods for which it holds that there is no other method that has 1) a higher NWN and 2) a higher number data sets evaluated, are identified as the best performing methods and are labelled with the name of the method.

5.4.1 *NDCG@3*

Figure 6 shows the performance of Learning to Rank methods for the NDCG@3 metric. Table 25 in Appendix B provides the raw NWN data for the Learning to Rank methods for which NDCG@3 evaluation results were available.

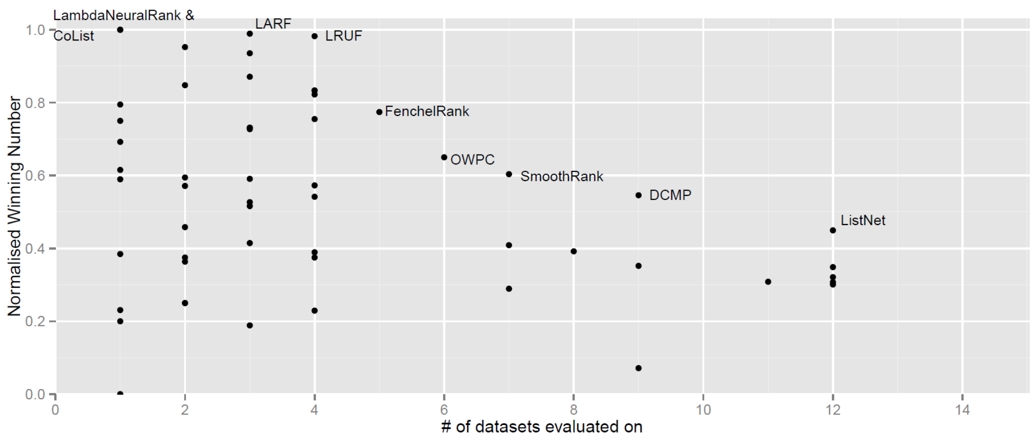

Figure 6: NDCG@3 comparison of Learning to Rank methods

LambdaNeuralRank and CoList both acquired a perfect NWN score of 1.0 by beating all other algorithms on one dataset, with LambdaNeuralRank winning on the AOL dataset and CoList winning on Yahoo! Set 2. LARF and LRUF both scored very high scores of near 1.0 on three of the LETOR 3.0 data sets, which can be said to have a higher degree of certainty on the methods' perform-ance because they are validated on three data sets which in addition are more relevant data sets than AOL and Yahoo! Set 2 because there are more evalu-ation results available for the LETOR 3.0 data sets (see Table 14). FenchelRank, OWPC, SmoothRank, DCMP and ListNet are in that order increasingly lower in NWN, but increasingly higher in number of data sets that they are evaluated on, resulting in a higher degree of certainty on the accuracy of the algorithms.

LambdaNeuralRank, CoList, LARF, LRUF, OWPC and DCMP evaluation res-ults are all based on one study, therefore are subjected to the risk of one overly optimistic study producing those results. FenchelRank evaluation result are based combined result from two studies, although those studies have overlap in authors. SmoothRank and ListNet have the most reliable evaluation result source, as they were official LETOR baseline runs.

5.4.2 *NDCG@5*

Figure 7 shows the performance of Learning to Rank methods for the NDCG@5 metric. Table 25 in Appendix B provides the raw Normalised Winning Number data for the Learning to Rank methods.

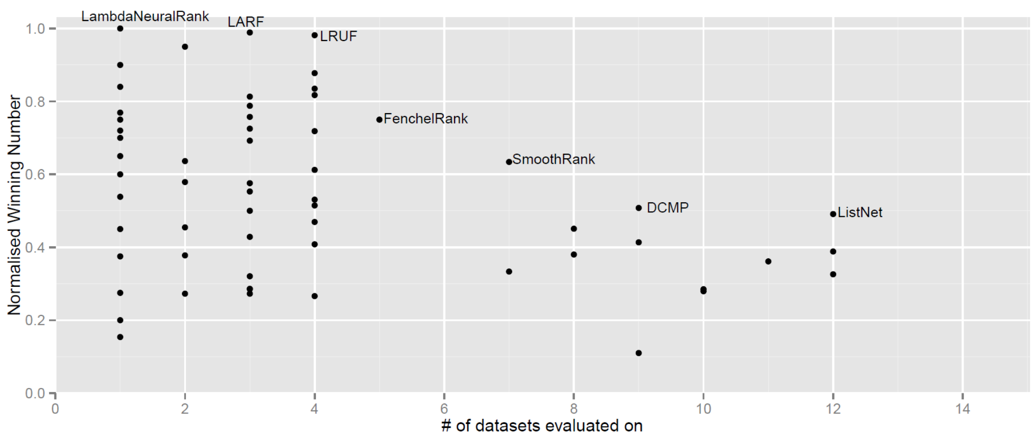

Figure 7: NDCG@5 comparison of Learning to Rank methods

LambdaNeuralRank again beat all other methods solely with results on the AOL dataset scoring a NWN of 1.0. LARF, LRUF, FenchelRank, SmoothRank, DCMP and ListNet are from left to right evaluated on an increasing number of data sets, but score decreasingly well in terms of NWN. These results are highly in agreement with the NDCG@3 comparison. The only modification compared to the NDCG@3 comparison being that OWPC did show to be a method for which there were no methods performing better on both axes in the NDCG@5 comparison, but not in the NDCG@3 comparison. Like in the NDCG@3 comparison, SmoothRank and ListNet can be regarded as most reliable results because the evaluation measurements for these methods are based on LETOR official baselines.

5.4.3 *NDCG@10*

Figure 8 shows the performance of Learning to Rank methods for the NDCG@10 metric. Table 26 in Appendix C provides the raw Normalised Winning Number data for the Learning to Rank methods.

LambdaMART and LambdaNeuralRank score a NWN of 1.0 on the NDCG@10 comparison. For LambdaNeuralRank these results are again based on AOL dataset measurements. LambdaMART showed the highest NDCG@10 performance for the MSLR-WEB10k dataset. The set of algorithms for which there is no other algorithm with both a higher NWN and number of data sets evaluated on is partly in agreement with those for the NDCG@3 and NDCG@5 comparisons: LARF, LRUF, FSMRank, SmoothRank, ListNet, RankSVM. SmoothRank and FSMRank were not present in this set for the NDCG@3 and NDCG@5 comparison,

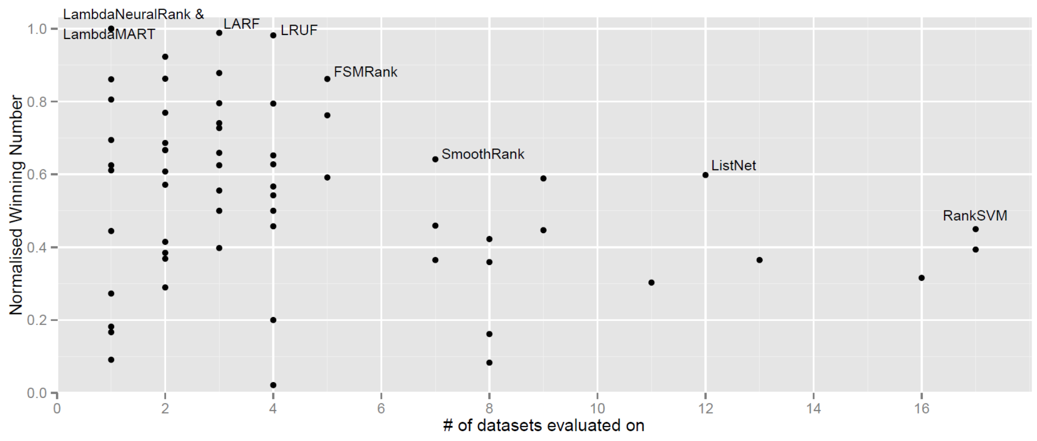

Figure 8: NDCG@10 comparison of Learning to Rank methods

but were close by, as can be seen in Tables 25 in Appendix B. DCMP is not in the set in contrast with the NDCG@3 and NDCG@5 comparison.

5.4.4 *MAP*

Figure 9 shows the performance of Learning to Rank methods for the MAP metric. Table 26 in Appendix C provides the raw NWN data for the Learning to Rank methods.

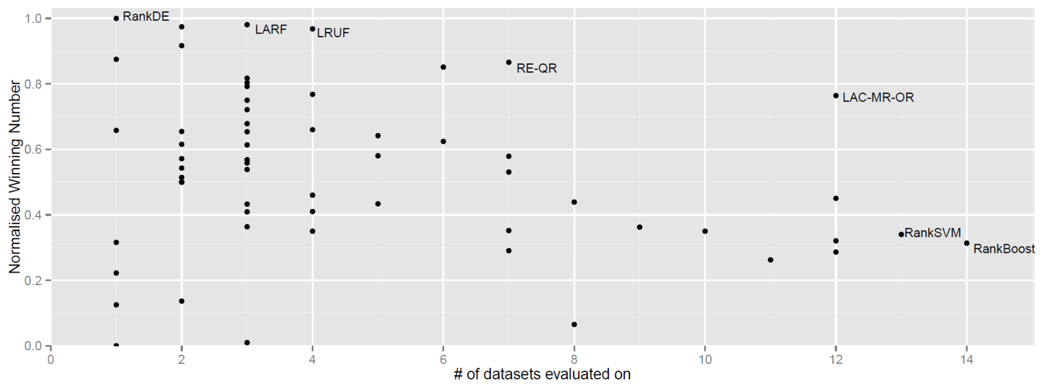

Figure 9: MAP comparison of Learning to Rank methods

Where comparisons on the NDCG-metric at different cut-off points where highly in agreement in terms of the best performing algorithms, the comparison in terms of MAP shows different results. RankDE scores a NWN of 1.0 on one dataset, like LambdaNeuralRank did on for all NDCG-comparisons. In contrast to LambdaNeuralRank, RankDE achieved this highest score on the LETOR 2.0 TD2003, a dataset on which many methods are evaluated.

LARF and LRUF score very high NWN scores, but based on only relatively few data sets, just as in the NDCG-comparisons. Notable is that SmoothRank and ListNet, which showed both high accuracy and high certainty on all NDCG-comparisons, are not within the best performing methods in the MAP-comparison.

A deeper look in the raw data Tables 25 and 26 in Appendices B and C respectively shows that LAC-MR-OR is evaluated on many more data sets for MAP compared to NDCG, which resulted in LAC-MR-OR obtaining equal certainty to ListNet with a higher NWN. SmoothRank performed a NWN of around 0.53 over 7 data sets, which is still good in both certainty and accuracy, but not among the top methods. RE-QR is one of the best performers in the MAP-comparison with a reasonable amount of benchmark evaluations. No reported NDCG performance was found in the literature study for RE-QR. There is a lot of certainty on the accuracy of RankBoost and RankSVM as both models are evaluated on the majority of data sets included in the comparison for the MAP-metric, but given their NWN it can said that both methods are not within the top performing Learning to Rank methods.

5.4.5 *Cross-metric*

Figure 10 shows the NWN as function of IWN for the methods described in Table 14. Table 27 in Appendix D provide the raw data plotted in Figure 10.

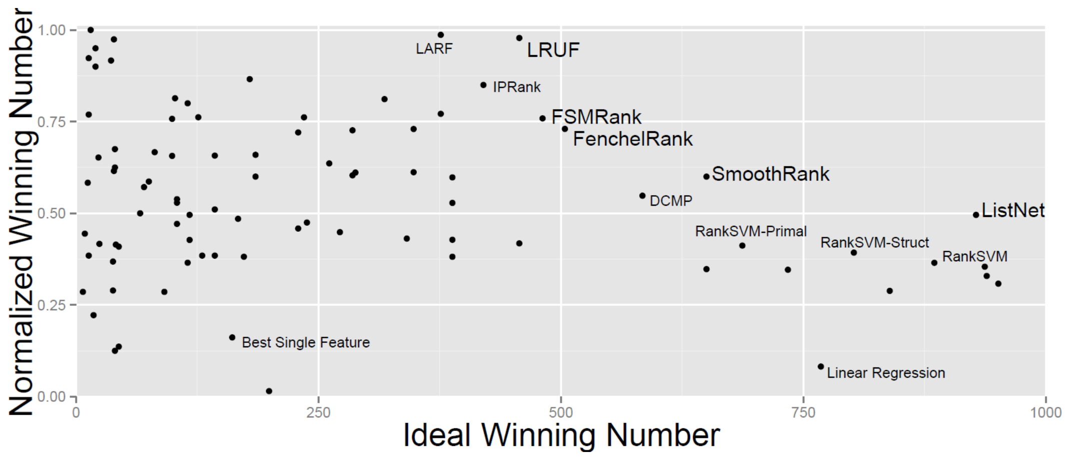

Figure 10: Cross-benchmark comparison of Learning to Rank methods

The cross-metric comparison is based on the NDCG@3, NDCG@5, NDCG@10 and MAP comparisons combined, which justifies analysing the comparison more thoroughly. Figure 10 labels the algorithms with no other algorithm having a higher value on both the horizontal axis and vertical axis, but also labels the algorithms with exactly one algorithm having a higher value on both axes with smaller font size. In addition, Linear Regression and the ranking method of simply sorting on the best single feature are labelled as baselines.

LRUF, FSMRank, FenchelRank, SmoothRank and ListNet showed to be the methods that have no other method superior to them in both IWN and NWN. LRUF is the only method that achieved this in all NDCG comparisons, the MAP comparison as well as the cross-metric comparison. With FenchelRank, FSMRank, SmoothRank and ListNet being among the top performing methods in all NDCG comparisons as well as in the cross-metric comparison, it can be concluded that the cross-metric results are highly defined by the NDCG performance as opposed to the MAP performance. This was to be expected, because the cross-metric comparison input data of three NDCG entries (@3, @5, and @10) enables it to have up to three times as many as many weight as the MAP comparison.

Support Vector Machine

LARF, IPRank and DCMP and several variants of RankSupport Vector Machine (SVM) performed very well on the cross-metric comparison, with all having only one method in its top right quadrant. LARF also performed among the top methods on the NDCG and MAP comparisons and DCMP was a top performer in a few of the NDCG comparisons.

C-CRF, DirectRank, FP-Rank, RankCSA, LambdaNeuralRank and VFLR all have near-perfect NWN measures, but have low IWN measures. Further evaluation runs of these methods on benchmark data sets that they are not yet evaluated on are desirable. The DirectRank paper [202] shows that the method is evaluated on more data sets than the number of data sets that we included evaluation results for in this meta-analysis. Some of the DirectRank measurements could not be used because measurements on some data sets were only available in graphical form and not in raw data.

LAC-MR-OR and RE-QR showed very good ranking accuracy in the MAP comparison on multiple data sets. Because LAC-MR-OR is only evaluated on two data sets for NDCG@10 and RE-QR is not evaluated for NDCG at all, LAC-MR-OR and RE-QR are not within the top performing methods in the cross-metric comparison.

5.5 LIMITATIONS

In the NWN calculation the weight of each benchmark on the total score is determined by the number of evaluation measurements on this benchmark. By calculating it in this way, we implicitly make the assumption that the Learning to Rank methods are (approximately) distributed uniformly over the benchmarks, such that the average Learning to Rank method tested are approximately equally hard for each data set. It could be the case however that this assumption is false and that significantly more accurate Learning to Rank methods are evaluated on some data sets than on other data sets.

A second limitation is that the data sets on which Learning to Rank methods have been evaluated can not always be regarded a random choice. It might be the case that some researchers chose to publish results for exactly those benchmark data sets that showed the most positive results for their Learning to Rank method.

Another limitation is that our comparison methodology relies on the correctness of the evaluation results found in the literature search step. This brings up a risk of overly optimistic evaluation results affecting our NWN results. Limiting the meta-analysis to those studies that report comparable results on one of the baseline methods of a benchmark set reduces this limitation but does not solve it completely. By taking IWN into account in Figure 10 we further mitigate this limitation, as IWN is loosely related with the number of studies that reported evaluation results for an algorithm.

Our comparison regarded evaluation results on NDCG@$\{3, 5, 10\}$ and MAP. By making the decision to include NDCG at three cut-off points and only a single MAP entry, we implicitly attain a higher weight for NDCG compared to MAP on an analysis that combines all measurements on the four metrics. This implicit weighting could be regarded as arbitrary, but the number of algorithm evaluation results gained by this makes it a pragmatic approach. Note that another implicit weighting lies in the paper dimension. Hence, the higher number of evaluation results specified in a paper, the higher the influence of this paper on the outcome of the analysis. This implicit weighting is not harmful to the validity of our comparison, as papers with a large number of evaluation results are more valuable than papers with a few evaluation results. In addition, papers with a high number of evaluation results are not expected to be less reliable than papers with fewer evaluation results.

5.6 CONCLUSIONS

We proposed a new way of comparing learning to rank methods based on sparse evaluation results data on a set of benchmark datasets. Our comparison methodology comprises of two components: 1) NWN, which provides insight in the ranking accuracy of the learning to rank method, and 2) IWN, which gives insight in the degree of certainty concerning the performance of the ranking accuracy.

Based on this this new comparison approach for a set of sparse evaluation results, we will now look back on the first research question of the thesis.

RQ1 What are the best performing Learning to Rank algorithms in terms of ranking accuracy on relevant benchmark data sets?

Although no closing arguments can be formulated on which Learning to Rank methods are most accurate, a lot of insight has been gained with the cross-benchmark comparison on which methods tend to perform better than others.

Based on our literature search for evaluation results on well-known benchmarks collections, a lot of insight has been gained with the cross-benchmark comparison on which methods tend to perform better than others. However, no closing arguments can be formulated on which learning to rank methods are most accurate. LRUF, FSMRank, FenchelRank, SmoothRank and ListNet were the learning to rank algorithms for which it holds that no other algorithm produced more accurate rankings with a higher degree of certainty of ranking accuracy. From left to right, the ranking accuracy of these methods decreases while the certainty of the ranking accuracy increases.

LRUF, FSMRank, FenchelRank, SmoothRank and ListNet were the Learning to Rank algorithms for which it holds that no other algorithm produced more accurate rankings with a higher degree of certainty of ranking accuracy. From left to right, the ranking accuracy of these methods decreases while the certainty of the ranking accuracy increases. For more definite conclusions on the relative performance of these five methods, more evaluation runs on are desirable for the methods on the left side on the list on benchmark data sets that these methods have not yet been evaluated on.

More evaluation runs are needed for the methods on the left side of Figure 10. Our work contributes to this by identifying promising learning to rank methods that researchers could focus on in performing additional evaluation runs.

In the following chapters of this thesis, concerning parallel execution of the Learning to Rank training phase, the scope will be limited to the five methods that turned out to be superior Learning to Rank methods in terms of ranking accuracy and certainty about this ranking accuracy: LRUF, FSMRank, FenchelRank, SmoothRank and ListNet. Although it can not be concluded that these methods are inarguably the most accurate Learning to Rank methods, a strong presumption has been raised that these five Learning to Rank are accurate ranking methods.

6 SELECTED LEARNING TO RANK METHODS

The learning algorithms of the well-performing Learning to Rank methods selected in Chapter 5 are presented and explained in this Chapter. The Learning to Rank methods will be discussed in order of an increasing degree of certainty and decreasing ranking accuracy, as concluded in Chapter 5.

6.1 LISTNET

ListNet [39] is a listwise ranking function whose loss function is not directly related to an information retrieval evaluation metric. ListNet's loss function is defined using a probability distribution on permutations. Probability distributions on permutations have been a research topic within the field of probability theory which has been extensively researched. ListNet is based on the Plackett-Luce model [164, 138], which is a permutation probability distribution over permutations that is well-known in the field of econometrics. The Plackett-Luce model defines a probability over permutations π from the set of all possible permutations Ω, given all document ranking scores s. The Plackett-Luce model defines the probability of a permutation π given the list as scores s as shown in Equation 1.

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})} \tag{1}$$

where π is a permutation on the n objects, ϕ is a monotonically increasing and strictly positive function, and $s_{\pi(j)}$ is the score of the object at position j of permutation π.

However, the total number of permutations of a list of n documents is $n!$, therefore, calculating the probabilities of all possible permutations quickly becomes intractable. To cope with this problem, the authors of ListNet propose the calculation of a *top one probability* as an alternative to the actual permutation probabilities. The top one probability of an object j equals the sum of the permutations probabilities of permutations in which object j is ranked as the top object. At first sight it seems to be the case that the $n!$ permutation probabilities still need to be calculated for this, but we can calculate the top one probability efficiently using Equation 2.

$$P_s(j) = \frac{\phi s_j}{\sum_{k=1}^{n} \phi(s_k)} \tag{2}$$

ListNet uses Gradient Descent to optimise a neural network such that its Cross Entropy loss compared to the top one over the training data relevance labels is minimal. For the monotonically increasing and strictly positive function ϕ is chosen to be the exponential function. With this choice for the exponential function, we can rewrite the abstract version top one probability definition of Equation 2 with the more specific Equation 3.

$$P_s(j) = \frac{\phi(s_j)}{\sum_{k=1}^{n} \phi(s_k)} = \frac{\exp(s_j)}{\sum_{k=1}^{n} \exp(s_k)} \tag{3}$$

Kullback-Leibler divergence

Note that some sources, including Liu [135], describe ListNet as using Kullback-Leibler divergence (KL divergence) as loss function. This difference in definition is not relevant however, as KL divergence and Cross Entropy are identical up to an additive constant when comparing distribution q against a fixed reference distribution p. The listwise Cross Entropy loss function $L(y^{(i)}, z^{(i)})$ is defined in Equation 4.

$$L(y^{(i)}, z^{(i)}) = \sum_{j=1}^{n} P_{y^{(i)}}(j) \log(P_{z^{(i)}}(j)) \tag{4}$$

where $H(p)$ is the entropy of p and $D_{KL}(p||q)$ is the KL divergence of q from p.

Equation 5 describes the gradient descent step to minimise listwise loss function $L(y^{(i)}, z^{(i)}(f_\omega))$ with respect to parameter ω.

$$\Delta\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial \omega} = -\sum_{j=1}^{n^{(i)}} P_{y^{(i)}}(x_j^{(i)}) \frac{\partial f_\omega(x_j^{(i)})}{\partial \omega} + \frac{1}{\sum_{j=1}^{n^{(i)}} \exp(f_\omega(x_j^{(i)}))} \sum_{j=1}^{n^{(i)}} \exp(f_\omega(x_j^{(i)})) \frac{\partial f_{\omega(x_j^{(i)})}}{\partial \omega} \tag{5}$$

The ListNet training algorithm is similar to RankNet, with the difference that ListNet uses a listwise loss function where RankNet uses a pairwise loss function. Algorithm 2 shows the pseudo-code of the ListNet training algorithm.

```
  Data: training data ${(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})}$
  Input: number of iterations $T$ and learning rate $\eta$
1 Initialize parameter $\omega$
2 for $t \leftarrow 1$ to $T$ do
3     for $i \leftarrow 1$ to $m$ do
4         Input $x^{(i)}$ of query $q^{(i)}$ to Neural Network and, for the current
          value of $\omega$, compute score list $z^{(i)}(f_\omega)$.
5         Compute gradient $\Delta\omega$ using Eq. (5).
6         Update $\omega = \omega - \eta \times \Delta\omega$.
7     end
8 end
9 Output Neural Network model $\omega$.
```

Algorithm 2: The ListNet learning algorithm, obtained from Cao et al. [39]

6.2 SMOOTHRANK

SmoothRank [47] is a listwise ranking method that, in contrast to ListNet, directly optimises an information retrieval evaluation measure. SmoothRank enables direct optimisation of the non-convex and discontinuous evaluation measure by smoothing, that is, approximating the rank position. In this section, we apply the SmoothRank approximation method to the Normalized Discounted Cumulative Gain (NDCG) evaluation measure for illustration, but the same procedure can be applied to Mean Average Precision (MAP) or any other information retrieval measure. The smoothing function used in SmoothRank is based on the softmax activation function [30], which is often used in neural networks. The softmax function is formally defined as shown in Equation 6.

Normalized Discounted Cumulative Gain
Mean Average Precision

$$p_i = \frac{e^{f_i/\sigma}}{\sum_{j=1}^{m} e^{f_j/\sigma}} \tag{6}$$

where σ is the smoothing parameter. Chapelle et al. [47] apply this softmax function to the NDCG formula and hereby introduce a soft version of indicator variable $h_{i,j}$ through the computation shown in Equation 7.

$$h_{i,j} = e^{-\frac{(f(x_i))-(f(x_{d(j)}))^2}{\sigma}} \Big/ \sum_{k=1}^{m} e^{-\frac{(f(x_k))-f(x_{d(j)})^2}{\sigma}} \tag{7}$$

It can be shown that the derivative of the smoothed NDCG version shown in Equation 7 and the smoothed versions of other Information Retrieval (IR) metrics can be calculated in $\mathcal{O}(m^2)$, which enable fast gradient descent optimisation. The optimisation step in SmoothRank uses the nonlinear conjugate gradient method with Polak-Ribiere update [186], which is a type of gradient descent method. This optimisation method is prone to local optima, which is alleviated by adding a pre-calculated, better than naive, starting point and by adding a regularisation term.

Information Retrieval

The starting point in SmoothRank is set to either the solution of a simple linear regression, or alternatively to the solution of RankSVM. Since this starting point is expected to already be a good solution, a regulariser term is added to the SmoothRank objective function to prevent the solution from deviating to much from the starting point. The regularised smooth objective function is formulated as $\lambda||w - w_0||_2$ where λ is a hyper-parameter tuned on the validation set, w_0 is the starting point solution, and $||x||_2$ is the $\ell 2$-norm regularisation function defined as $||x||_2 = \sqrt{\sum_i x_i^2}$.

The choice of the smoothing parameter σ in Equation 7 is important, because a too small value makes the function more non-smooth and therefore harder to optimise, while a too large value results in a optimisation function with optima that substantially differ from the true optimal rankings. To deal with the problem of choosing the smoothing parameter, SmoothRank uses an annealing method where the optimisation procedure starts with a large σ and iteratively reduces it by dividing it by two at each iteration. Algorithm 3 shows the algorithm that summarises all steps of the SmoothRank method.

```
1 Find an initial solution $w_0$ (by regression or RankSVM).
2 Set $w = w_0$ and $\sigma$ to a large value.
3 while Stopping condition not satisfied do
4     Starting from $w$, minimize by non-linear conjugate gradient descent:
      $\lambda||w - w_0||_2 - \sum_q A_q(w, \sigma)$
5     $\sigma = \sigma/2$
6 end
```

Algorithm 3: The SmoothRank learning algorithm, obtained from Chapelle and Wu [47]

6.3 FENCHELRANK

FenchelRank [119] is a ranking method that addresses the sparse Learning to Rank problem, which is the problem of learning a ranking function with only a few non-zero coefficients with respect to the input features. FenchelRank is based on the theory of Fenchel Duality [177] and uses a generic algorithm framework proposed by Shalev-Shwartz and Singer [185]. FenchelRank optimises the objective function shown in Equation 8 which is equivalent to the standard pairwise loss function with ℓ_1-norm regularisation. The ℓ_1-norm regularisation function in this equation is represented by $||x||_1$ and is defined as $||x||_1 = \sum_i |x_i|$.

$$\min_w G(w) = \min_w I_{||w||_1 \leqslant 1}(w) + \frac{r^2}{p} \sum_{i=1}^{p} \max(0, \frac{1}{r} - (Kw)_i)^2 \tag{8}$$

where $I_C(w)$ is a function that is 0 if condition C is satisfied, and ∞ otherwise. m is the dimension of the data, p is the number of comparable object pairs, K is a matrix in $\mathbb{R}^{p \times m}$ that contains pairwise information. The objective function in Equation 8 is not differentiable everywhere because of its ℓ_1-norm regularisation term. Fenchel's duality theorem [177], defined in Equation 9, provides a way to approximate the optimal value of this non-differentiable optimisation problem by instead solving the Fenchel dual of the optimisation problem.

$$\min_x (f(x) - g(x)) = \max_p (f_*(p) - f^*(p)) \tag{9}$$

where f^* is the convex, and f_* the concave conjugate of f.

To ease applying Fenchel's duality theorem (Equation 7) to the ℓ_1-regularised pairwise loss function (Equation 8), Lai et al. define $D(w)$ as $D(w) = -G(w)$. Equation 10 shown the resulting Fenchel dual of the ℓ_1-regularised pairwise loss function, which is the loss function that is used in FenchelRank.

$$\max_w D(w) = \max_w I_{||w||_1 \leqslant 1}(w) - \frac{r^2}{p} \sum_{i=1}^{p} \max(0, \frac{1}{r} - (Kw)_i)^2 \tag{10}$$

Algorithm 4 shows the FenchelRank training algorithm to optimise the Fenchel dual of the pairwise loss function. The $||x||_\infty$ term in this algorithm represents an ℓ_∞-norm regularisation term and is defined as $||x||_\infty = \max_i |x_i|$.

```
   Data: pairwise training data matrix $K$
   Input: desired accuracy $\epsilon$, maximum number of iterations $T$ and the radius
          $r$ of the $\ell 1$ ball.
 1 Initialize: $w_1 = 0_m$
 2 for $t \leftarrow 1$ to $T$ do
 3     //check if the early stopping criterion is satisfied
 4     if $||g_t||_\infty + \langle d_t, -Kw_t \rangle \leqslant \epsilon$ then
 5         //here $d_t = \nabla f^*(-Kw_t) = \frac{\partial f^*(-Kw)}{\partial(Kw)}|w = w_t$,
 6         // $\langle x, y \rangle$ represents the inner products of vectors $x$ and $y$, and
 7         // $g_t = d_t^T K$
 8         return $w_t$ as ranking predictor $w$
 9     end
10     //greedily choose a feature to update
11     Choose $j_t = \arg\max_j |(g_t)_j)|$
12     //compute an appropriate step size
13     Let $\mu_t = \arg\max_{0 \leqslant \mu_t \leqslant 1} D((1-\mu_t)w_t + \mu_t \operatorname{sign}((g_t)_{j_t})e^{j_t})$
14     //update the model with the chosen feature and step size
15     Update $w_{t+1} = (1-\mu_t)w_t + \mu_t \operatorname{sign}((g_t)j_t)e^{j_t}$
16 end
17 return $w_T$ as ranking predictor for $w$
```

Algorithm 4: The FenchelRank learning algorithm, obtained from Lai et al. [119]

6.4 FSMRANK

Lai et al. [122] observed that existing feature selection methods in Learning to Rank all follow a two-stage paradigm consisting of a first stage of selecting a subset of features from the original features, followed by a second stage where a ranking model learnt based on the selected features. Lai et al. [122] state it as a limitation of this paradigm that the selected features in the first step are not necessarily the optimal features for the second stage where the ranking model is build. FSMRank [122] addresses this limitation by formulating a joint convex optimisation function that minimises ranking errors while simultaneously selecting a subset of features.

FSMRank uses an extended version of gradient descent optimisation, proposed by Yuri Nesterov, that enables faster convergence for convex problems [147]. Nesterov's accelerated gradient descent can guarantee an ϵ-accurate solution in at most T iterations where $\epsilon = \mathcal{O}(1/T^2)$.

Let $S = (q_k, \hat{X}^{(q_k)}, Y^{(q_k)})_{k=1}^n$ be a training set with queries q_k, corresponding retrieval objects $\hat{X}^{(q_k)}$, and corresponding relevance labels $Y^{(q_k)}$. Let $||x||_1$ be the l_1-norm regularisation function that is defined as $||x||_1 = \sum_i |x_i|$. Let $\oslash$ be the element-wise division operator. $\hat{A}$ is a $d \times d$ similarity matrix that contains similarity scores between features. The convex joint optimisation function in FSMRank is defined as shown in Equation 11.

$$\min_{\hat{w}} \frac{\lambda_1}{2}\hat{w}^T\hat{A}\hat{w} + \lambda_2\|\hat{w} \oslash \hat{s}\|_1 + f(\hat{w}, (\hat{X}^{(q_k)}, Y^{(q_k)})_{k=1}^{n}) \tag{11}$$

The first term in Equation 11 applies a penalty on redundancy of large weighted features by using the $\hat{A}$ matrix. The well-known Pearson correlation coefficient is used to calculate similarity matrix $\hat{A}$, based on the values for the features in the training data. λ_1 is a hyper-parameter of the model and can be used to set the weight of the similarity penalty term.

The second term of Equation 11 contains a ℓ_1-norm regularisation term to select the effective features from the feature set. λ_2 is a hyper-parameter of the model and can be used to set the weight of the regularisation term.

The third and last term of the equation represents the loss function in terms of ranking errors. Loss function f can in theory be any convex loss function, but Lai et al. [122] used a squared hinge loss function for their evaluation measurements.

Algorithm 5 shows the steps of the FSMRank training algorithm. As stated, FSMRank uses an extended version of Nesterov's accelerated gradient method. This optimisation method can handle optimisation problems in the form of $\min_w ł(w) + r(w)$ where $l(w)$ is a convex function with Lipschitz gradient and $r(w)$ is convex, but non-smooth. The optimization function of Equation 11 is reformulated to match this form as presented in Equation 12.

$$\arg\min_{w_t : w_t \in \mathbb{R}} Q(w_t, z_t) = \lambda_2 \sum_{i=1}^{2d} \frac{w_i}{s_i} + \langle l'(z_t), w_t - z_t \rangle + \frac{L}{2}\|w_t - z_t\|^2 \tag{12}$$

where $Q(w_t, z_t)$ is a combination of the non-smooth part $r(w_t)$ and a quadratic approximation of the smooth part $l(w_t)$. λ_1, λ_2 and Lipschitz constant L_0 are input parameters of the algorithm and can be optimised through cross-validation or on a validation set.

```
Data: training set $S = (q_k, \hat{X}^{(q_k)}, Y^(q_k)_{k=1}^{n}$
Input: $\lambda_1, \lambda_2, T$ and $L_0$
1 Initialize: $w_0 = z_1 = 0, \alpha_1 = 1, \gamma = 2, L = L_0/\gamma^{10}$
2 for $t \leftarrow 1$ to $T$ do
3     Let $g = l^{'}(z_t)$
4     while true do
5         //projection step
6         $w_t = \arg\min_{w_t : w_t \in \mathbb{R}_+^{2d}} Q(w_t, z_t)$
7         if $l(w_t) \leqslant l(z_t) + \langle l^{'}(z_t), w_t - z_t \rangle + \frac{L}{2} \|w_t - z_t\|^2$ then
8             break
9         end
10        $L = \lambda L$
11    end
12    if $\frac{|F(w_t) - F(w_{t-1})|}{|F(w_{t-1})|} \leqslant \epsilon_s$ then
13        //early stopping criterion
14        break
15    end
16    $\alpha_{t+1} = \frac{1 + \sqrt{1 + 4\alpha_t^2}}{2}$
17    $z_{t+1} = w_t + \frac{\alpha_t - 1}{\alpha_t + 1}(w_t - w_{t-1})$
18 end
```
Algorithm 5: The FSMRank learning algorithm, obtained from Lai et al. [122]

6.5 LRUF

LRUF [208] is Learning to Rank method based on the theory of learning automata. Learning automata are adaptive decision-making units that learn the optimal set of actions through repeated interactions with its environment. Variable structure learning automata can be defined as a triple $< \beta, \alpha, L >$, with β being the set of inputs, α the set of actions. L, the learning algorithm ,is a recurrence relation that modifies the action probability vector of the actions in α.

Three well-known learning algorithms L used in variable structure learning automata are *linear reward-penalty* (L_{R-P}), *linear reward-ϵ-penalty* ($L_{R-\epsilon P}$) and *linear reward-inaction* (L_{R-I}). LRUF uses the L_{R-I} learning algorithm, which updates the action probability vector following Equation 13 when the selected action $a_i(k)$ is rewarded by the environment.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)] & \text{if } j = i \\ (1-a)p_j(k) & \text{otherwise} \end{cases} \tag{13}$$

where i and j are indices of action in α and $p(k)$ is the probability vector over the action set at rank k. a is the learning rate of the model.

A variable action-set learning automaton is an automaton in which the number of available actions change over time. It has been shown that a combination of a variable action-set learning automaton with the L_{R-I} learning algorithm is absolutely expedient and ϵ-optimal [204], which means that it is guaranteed to approximate the optimal solution to some value ϵ and each update step is guaranteed not to decrease performance of the model. A variable-action set learning automata has a finite set of r actions $\alpha = \alpha_1, ..., \alpha_r$. A is defined as the power set of α, $A = A_1, .., A_m$ with $m = 2^r - 1$. $\psi(k)$ is a probability distribution over A such that $\psi(k) = p(A(k) = A_i | A_i \in A, 1 \leqslant i \leqslant 2^r - 1)$. $\hat{p}_i(k)$ is the probability of choosing action α_i given that action subset $A(k)$ has already been selected and $\alpha_i \in A(k)$.

LRUF uses an optimisation problem that can be illustrated as a quintuple $< q_i, A_i, \underline{d_i}, \underline{R_i}, \underline{f_i} >$, where q_i is a submitted query, A_i is a variable action-set learning automaton, $\underline{d_i}$ is a set of documents associated with q_i, and $\underline{R_i} = r_i^j | \forall d_i^j \in \underline{d_i}$ is a ranking function that assigns rank r_i^j to each document. $\underline{f_i}$ is a feedback set used for the update step described in Equation 13. For each rank k the learning automaton A_i chooses one of its actions following its probability vector, jointly forming $\underline{R_i}$. LRUF translates the feedback in $\underline{f_i}$ to an understandable value to use it to converge the action probability vector in optimal configuration. Therefore LRUF defines a $g_i : \underline{f_i} \rightarrow \mathbb{R}^+$ to be a mapping from the feedback set into a positive real number. The LRUF mapping function g_i computes the average relevance score of ranking R_i based on f_i and is defined as shown in Equation 14.

$$g_i = \frac{1}{N_i} \sum\nolimits_{d_i^j \in \underline{f_i}} a(r_i^j)^{-1} \tag{14}$$

in which N_i refers to the size of feedback set $\underline{f_i}$, r_i^j is the rank of document d_i^j in R_i and, again, a denotes the learning rate. Initially, before any feedback, the action probability factors are initialised with equal probability.

In case the document set of the search engine changes, i.e. new documents are indexed or old documents are removed, LRUF has an Increase Action-Set Size (IAS) and a Reduce Action-Set Size (RAS) procedure defined to adapt to the new document set without needing a complete retraining of the model. Because this thesis focusses on model accuracy and scalability the IAS and RAS procedure of LRUF will not be explained in further detail.

Torkestani [208] does not provide pseudo code specifically for the training phase of the LRUF algorithm. Instead an algorithm including both ranking and automaton update is presented, and included in Algorithm 6. Initial training of the algorithm can be performed by using the training data relevance labels in the feedback set of the algorithm.

```
   Data: Query $q_i$, Number of results $n_i$
1  Assume: Let $A_i$ be the learning automaton corresponding to query $q_i$ with
   action-set $\alpha_i$
2  Action $\alpha_i^j \in \alpha_i$ is associated with document $d_i^j \in \underline{d_i}$
3  Let $k$ denote the stage number
4  Let $G$ be the total relevance score
5  Initialise: $k \leftarrow 1, T_i \leftarrow 0$
6  while $k \leqslant n_i$ do
7      $A_i$ chooses one of its actions (e.g. $a_i^k$) at random
8      Document $d_i^j$ corresponding to selected action $\alpha_i^j$ is ranked at $K^{th}$
       position of $\underline{R_i}$
9      Configuration of $A_i$ is updated by disabling action $\alpha_i^j$
10     $k \leftarrow k + 1$
11 end
12 Ranking $\underline{R_i}$ is shown to the user
13 $N_i \leftarrow 0, \underline{f_i} \leftarrow \emptyset, G \leftarrow 0$
14 repeat
15     for every document $d_i^j$ visited by user do
16         $\underline{f_i} \leftarrow \underline{f_i} + d_i^j$
17         $G \leftarrow G + a * (r_i^j)^{-1}$
18     end
19     $N_i \leftarrow N_i + 1$
20 until query session is expired;
21 $g_i \leftarrow \frac{G}{N_i}$
22 Configuration of $A_i$ is updated by re-enabling all disabled actions
23 if $g_i \geqslant T_i$ then
24     Reward the actions corresponding to all visited documents by Equation
       13
25     $T_i \leftarrow g_i$
26 end
27 for $\forall \alpha_i^j \in \underline{\alpha_i}$ do
28     if $p_i^j < T_\epsilon$ then
29         $d_i^j$ is replaced by another document of the searched results
30     end
31 end
32 Output $\underline{R_i}$
```

Algorithm 6: The LRUF algorithm, obtained from Torkestani [208]

7 IMPLEMENTATION

The first section of this chapter will briefly discuss the HDInsight platform, the Hadoop ecosystem components offered by HDInsight and, in this regard, the Hadoop components used for implementation of the algorithms described in Chapter 6. The second section of this chapter describes a Java framework that handles the joint components needed for MapReduce Learning to Rank computation, independent of the Learning to Rank model. The subsequent sections report the implementation details of specific Learning to Rank models.

7.1 ARCHITECTURE

Ranking algorithms consist of a sequence of operations on the input data which are often of iterative nature. Apache Pig [151] is used to implement the sequence of operations on the input data. Pig Latin is the data processing language that runs on Apache Pig. It was designed based on the observation that the traditional MapReduce paradigm is too low-level and rigid, and holds the middle between the declarative style of SQL and the procedural style of MapReduce. The Apache Pig system translates Pig Latin into MapReduce plans that are executed on Hadoop. The choice to implement the Learning to Rank algorithms in Pig Latin allows for more focus on the data operations and less focus on low-level implementation details, as compared to native Hadoop MapReduce. Furthermore, it allows us to rely on Apache Pig to create efficient MapReduce plans out of the Pig Latin code and therefore lowers the implementation-dependent factor of the experiments.

7.1.1 *The HDInsight Platform*

Hadoop Distributed File System

Windows Azure Storage Blob

Azure HDInsight supports the traditional Hadoop Distributed File System (HDFS) as described by Shvacko et al. [189], as well as Microsoft's own storage solution Windows Azure Storage Blob (WASB). Blob storage decouples the storage from the HDInsight Hadoop cluster; it enables safe deletion of a HDInsight cluster without losing data, as data is not solely stored on the cluster itself, but also on a separate storage that is not cluster-dependent. Azure WASB storage allows the user to select one of Microsoft's data centres for storage. WASB storage in the West Europe region (located in Amsterdam) is used for storage, as this data centre is located close to where the experiments are executed.

Microsoft offers a scalable and on-demand Hadoop service with HDInsight, which enables Hadoop services for those not able to make the required investments for their own cluster. The latest HDInsight version available at the time of writing, HDInsight 3.1, runs the Hortonworks distribution of Hadoop version 2.4. While early versions of Hadoop were merely an open source implementation of MapReduce, newer versions since Hadoop 2.0 offer support for variety of programming models with the introduction of Hadoop YARN [213]. HDInsight Hadoop data nodes are regular Large A3 Microsoft Azure virtual machines with four core processors and 7GB RAM. Newly supported programming models since Hadoop 2.0 include Dryad [107], Giraph [17], Message Passing Interface (MPI), REEF [54], Spark [251], and Storm [12]. Even though these programming models are now supported by Hadoop and some of these programming models have recently increased in popularity, they still lack the critical mass as the data processing framework of choice that MapReduce has as Lin argued back in 2012 [129] (see section 1.1). Therefore, even though some of these programming models might be a better fit for iterative algorithms, we use Hadoop MapReduce of the programming model to implement the Learning to Rank algorithms.

Message Passing Interface

HDInsight 3.1 offers multiple ways of submitting Hadoop jobs to a HDInsight cluster, which are described in Table 15. Similar to WASB storage, one of the Microsoft data centres can be chosen to host the Hadoop cluster when a HDInsight cluster is created. To guarantee proximity between storage and cluster, the West Europe data centre in Amsterdam is used as cluster location.

7.1.2 *Framework description*

Fold handling in the context of a cross-validation experiment is the process of loading the correct data folds for training, validation and testing in the multiple rounds that form the cross-validation experiment. Fold handling is a task that needs to be taken care of independent of the ranking model that is being evaluated. Given that most ranking models are of iterative nature, the task of iteration handling is also a task that need to be performed independent of the ranking model. Iteration handling and fold handling are procedures in which the same steps are repeated for each iteration or cross-validation round respectively. Pig Latin, in contrast to more procedural MapReduce-job-generating languages like Sawzall [163], has no support for loops, which are needed to perform iteration handling and fold handling. Since iteration handling and fold handling are tasks that need to be addressed for each ranking model and that cannot be solved with Pig, we create a framework that takes care of both iteration and fold handling and generates the Pig Latin code for the current iteration and fold.

Learning to Rank algorithms are likely to consist of multiple loops over the input data per iteration of the algorithm, and consequently multiple Pig jobs will be needed per iteration of the algorithm. An example of this is a Pig implementation of gradient descent, where a first Pig job might calculate gradients and writes them to storage, after which the framework assists in reading the gradients from storage and allows them to be used as input parameters of a second Pig job that calculates new feature weights based on its gradient para-

Job submission method	Type	Description
Powershell	Powershell scripts	The Azure module for Windows PowerShell enables direct submission of Hadoop jobs through PowerShell cmdlets.
C# API	C# API	A wrapper API is offered to submit Hadoop MapReduce jobs directly from C# code.
HiveServer/HiveServer2	REST endpoint	Apache Hive [205] is an open-source data warehousing solution on top of Hadoop, that supports processing of a SQL-like declarative language called HiveQL. HiveServer and its successor HiveServer 2 are REST endpoints that allow remote submission of HiveQL queries.
Oozie	REST endpoint	Apache Oozie [108] is a workflow scheduler to manage Apache Hadoop jobs. Oozie enables users to specify Directed Acyclical Graphs of action, where each action is specified in either MapReduce or Pig.
WebHCat/Templeton	REST endpoint	WebHCat, formerly known as Templeton, is a REST API for HCatalog, a table and storage management layer for Hadoop. WebHCat allows users to use either Apache Pig, Apache MapReduce or Apache Hive for data processing.

Table 15: HDInsight REST endpoints for job submission

Job submission procedure for Apache Oozie	Job submission procedure for WebHCat/Templeton
1. Let the framework build the Pig job dynamically.	1. Let the framework build the Pig job dynamically.
2. Encapsulate the Pig job in an Oozie workflow.	2. Submit the Pig job through the WebHCat/Templeton REST API.
3. Upload the Oozie workflow to HDFS storage.	
4. Execute the Oozie workflow through the Oozie REST API.	

Table 16: Comparison of Oozie and WebHCat job submission procedures

meter and a predetermined step size. Handling communication between Pig jobs within a single iteration of a Learning to Rank algorithm is not trivial. A Pig Job, after completion, writes its result to WASB storage, while this result is needed by a subsequent Pig job as parameter. The framework enables reading the result of a Pig job from WASB storage which can then be used as parameter within a subsequent Pig job.

The aim of this framework is to let implementations of ranking models focus solely on implementing the sequential steps of one iteration of the algorithm, while the framework handles that 1) these sequential steps are performed iteratively, 2) these sequential steps are performed on the multiple training folds of data and 3) data can be passed from one Pig job within the algorithm to another Pig job. Our framework that handles folding and iterations will be implemented in Java and will work such that fold- and iteration dependent parts of the Pig code will be generated dynamically by the Java framework after which the Pig job will be sent to the cluster.

Oozie and WebHCat/Templeton are the two methods for job submission in Table 15 that both 1) support Apache Pig jobs, and 2) Can be used from within Java code. Table 16 shows the necessary procedures for job submission from Oozie as well as from WebHCat will be sketched. Table 16 shows Oozie job submission to be more complex for the case of dynamically generated jobs than WebHCat/Templeton. Oozie is more fitting for static Hadoop jobs that require a workflow consisting of a sequence of Pig and MapReduce jobs mixed together, but is a lesser fit for our situation. Templeton will be used for submission of the Pig jobs, as the HDInsight version that was available at the start of the implementation did not yet support WebHCat.

We create a utility class HDInsightWrapper, which abstracts away connection setup, authorisation and response parsing from the tasks of submitting jobs to the cluster and retrieving data from WASB storage. The HDInsightWrapper class is parametrised with connection details like the cluster name, cluster user, cluster password, storage account, storage container and storage key, and has two methods:

VOID RUNPIG(STRING PIGCODE) Receives a Pig script in String format as input and handles the submission of this script as MapReduce job on the cluster. Polls the Templeton REST API with progress requests every 50 milliseconds until the MapReduce job has completed. Blocks until the MapReduce job has completed.

STRING RUNPIG(STRING PIGCODE, STRING OUTPUTDIR) Identical to runPig(String pigCode), but reads the result of the computation from location 'outputDir' after completion of the MapReduce job and outputs the result as String value.

Setting the degree of parallelisation is crucial to obtain optimal cluster utilisation. HDInsight cluster machines offer four mapper slots and two reducer slots per data node. Cluster utilisation is optimal when the computational work is evenly distributed over exactly the number of available mappers in the map phase, and evenly distributed over exactly the number of reducers in the reduce phase. The number of mappers used can be controlled by setting MapReduce configuration parameters *mapreduce.input.fileinputformat.split.maxsize*, *mapreduce.input.fileinputformat.split.minsize*, and the Pig configuration parameter *pig.maxCombinedSplitSize* to the data size in bytes of the input training data divided by the number of available mappers. To control the number of used reducers, Pig offers two options: 1) the default_parallel parameters, which can set a default number of reducers used throughout all MapReduce jobs that the Pig script consists of, and 2) the "PARALLEL" clause, which sets the number of reducers at operator level (overrides default_parallel if set). The framework uses listed configuration parameters to control the numbers of mappers used and uses the "PARALLEL" clause to set the number of used reducers. Pig code to set the number of mappers and reducers are dynamically set by the framework such that the optimal number of mappers and reducers are used.

7.2 LISTNET

The following sections describe the Pig jobs that jointly form the three independent parts of the ListNet ranking model: 1) Preprocessing, 2) Training (this includes evaluation over the validation set) and 3) Testing (evaluation over the test set).

7.2.1 *Preprocessing*

The preprocessing phase consists of two separate Pig jobs. The first Pig job determines the minimum and the maximum values per feature in the training set. The second Pig job rescales each feature of the train, validation and test data sets using the following formula for rescaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{15}$$

UDF	Description
udf.util.ToStandardForm()	Transforms the data set into the standard form of relevance label in first column followed by feature values. Strips data of any other columns, if present.
udf.util.GetMinMax()	Extracts the minimum and maximum value per feature, for the documents of a single query.
udf.util.CombineMinMax()	Combines outputs of the udf.util.GetMinMax() UDF for each query into globally minimum and maximum feature values.

Table 17: Description of preprocessing phase User Defined Functions (Pig job 1)

This rescaling procedure sets the values of all features to be within range $[0, 1]$.

The first Pig job:

```
REGISTER [path prefix]/lib/*.jar;
TRAIN = LOAD '[path prefix]/input/[data set name]/Fold[fold number]/train.
    txt' USING PigStorage(' ');
TRAIN_STD = FOREACH TRAIN GENERATE flatten(udf.util.ToStandardForm(*));
TRAIN_STD_BY_QUERY = GROUP TRAIN_STD BY $1 PARALLEL [available reducers];
MIN_MAX = FOREACH TRAIN_STD_BY_QUERY GENERATE flatten(udf.util.GetMinMax(*)
    );
MIN_MAX_GRPD = GROUP MIN_MAX ALL;
MIN_MAX_FIN = FOREACH MIN_MAX_GRPD GENERATE flatten(udf.util.CombineMinMax
    (*));
STORE MIN_MAX_FIN INTO 'minmax[fold number]';
```

All expressions between square brackets in the code snippet above are dynamically set by the framework. The minimum and maximum values per feature stored in MIN_MAX_FIN are read from storage by the framework after completion of the MapReduce job. The framework passes these minimum and maximum values to the second Pig job by passing an array of minimum and maximum values to the constructor of the udf.util.ScaleFeatures() User Defined Function (UDF), that performs the rescaling operation of Equation 15.

User Defined Function

UDF	Description
udf.util.ToStandardForm()	See Table 17 for description.
udf.util.ScaleFeatures()	Uses the minimum and maximum feature values with which it is parametrised perform the following rescaling transformation to the features: $x' = \frac{x - min(x)}{max(x) - min(x)}$.

Table 18: Description of preprocessing phase User Defined Functions (Pig job 2)

The second Pig job:

```
REGISTER [path prefix]/lib/*.jar;
TRAIN = LOAD '[path prefix]/input/[data set name]/Fold[fold number]/train.
    txt' USING PigStorage(' ');
VALIDATE = LOAD '[path prefix]/input/[data set name]/Fold[fold number]/vali
    .txt' USING PigStorage(' ');
TEST = LOAD '[path prefix]/input/[data set name]/Fold[fold number]/test.txt
    ' USING PigStorage(' ');
TRAIN_STD = FOREACH TRAIN GENERATE flatten(udf.util.ToStandardForm(*));
VALIDATE_STD = FOREACH VALIDATE GENERATE flatten(udf.util.ToStandardForm(*)
    );
TEST_STD = FOREACH TEST GENERATE flatten(udf.util.ToStandardForm(*));
DEFINE ScaleFeatures udf.util.ScaleFeatures('[array with minimum and
    maximum feature values]');
TRAIN_SCA = FOREACH TRAIN_STD GENERATE flatten(ScaleFeatures(*));
VALIDATE_SCA = FOREACH VALIDATE_STD GENERATE flatten(ScaleFeatures(*));
TEST_SCA = FOREACH TEST_STD GENERATE flatten(ScaleFeatures(*));
STORE TRAIN_SCA INTO 'train_sca[fold number]' USING BinStorage();
STORE VALIDATE_SCA INTO 'validate_sca[fold number]' USING BinStorage();
STORE TEST_SCA INTO 'test_sca[fold number]' USING BinStorage();
```

7.2.2 *Training*

The Pig jobs that form the ListNet training phase form an implementation of the theoretical description of ListNet as described in section 6.1. The training stage, like the preprocessing stage, consists of two separate Pig jobs. The first Pig job calculates the Cross Entropy loss on training data of the current model and calculates the gradients for the next model update. The second Pig job is an internal validation step that validates the model on the validation set by calculating Normalized Discounted Cumulative Gain (NDCG)@k.

Normalized Discounted Cumulative Gain

The first Pig job:

```
REGISTER [path prefix]/lib/*.jar;
DEFINE QueryLossGradient udf.listnet.QueryLossGradient('[feature
    dimensionality of data set]');
DEFINE ExpRelOurScores udf.listnet.ExpRelOurScores('[neural network weights
     & iteration number]');
[FIRST TRAINING ITERATION:]
        TRAIN_SCA = LOAD 'train_sca[fold number]/*' USING BinStorage();
        TR_BY_QUERY = GROUP TRAIN_SCA BY $1 PARALLEL [number of avaiable
            reducers];
        TR_EXP_REL_SCORES = FOREACH TR_BY_QUERY GENERATE flatten(
            ExpRelOurScores(TRAIN_SCA));
        STORE TR_EXP_REL_SCORES INTO 'tr_exp_rel_scores-f[fold number]'
            USING BinStorage();
[SUBSEQUENT TRAINING ITERATIONS:]
        TR_EXP_REL_SCORES = LOAD 'tr_exp_rel_scores-f[fold number]/*' USING
             BinStorage();
        TR_EXP_REL_SCORES = FOREACH TR_EXP_REL_SCORES GENERATE flatten(
            ExpRelOurScores(*)) PARALLEL [number of available reducers];
TR_QUERY_LOSS_GRADIENT = FOREACH TR_EXP_REL_SCORES GENERATE flatten(
    QueryLossGradient(*)) PARALLEL [number of available reducers];
TR_QUERY_LOSS_GRADIENT_GRPD = GROUP TR_QUERY_LOSS_GRADIENT ALL;
TR_LOSS_GRADIENT = FOREACH TR_QUERY_LOSS_GRADIENT_GRPD GENERATE flatten(udf
    .listnet.MultiSum(*));
STORE TR_LOSS_GRADIENT INTO 'tr_loss_gradient-f[fold number]i[iteration
    number]';
```

The actual neural network weights, passed by the framework as constructor parameter in the script above, are administered in Java code. The per-feature gradients calculated by the Pig job above are read from storage after completion and used to update the neural network by, for each feature, adding the feature gradient multiplied with a step size parameter to the previous feature weight.

The second Pig job of the training stage validates the performance of the model weights that were trained in the first Pig job on the validation data set.

UDF	Description
udf.listnet.QueryLossGradient()	Calculates the Cross Entropy loss for a query and calculates the gradients per feature based on this query.
udf.listnet.ExpRelOurScores()	Calculates the predicted relevance label of a query based on the current model weights and transforms this following the transformation $x \rightarrow e^x$. In case the current iteration is the first iteration, the same transformation is applied to the ground truth relevance label.
udf.listnet.MultiSum()	Calculates aggregated loss and feature gradients by summing the per-query losses and per-query feature gradients.

Table 19: Description of training phase User Defined Functions (Pig job 1)

UDF	Description
udf.util.Ndcg()	Calculates NDCG@k for a query.

Table 20: Description of training phase User Defined Functions (Pig job 2)

The second Pig job:

```
REGISTER [path prefix]/lib/*.jar;
DEFINE Ndcg udf.util.Ndcg('[neural network weights & NDCG cut-off parameter
    ]');
[FIRST TRAINING ITERATION:]
        VALIDATE_SCA = LOAD 'validate_sca[fold number]/*' USING BinStorage
            ();
        VA_BY_QUERY = GROUP VALIDATE_SCA BY $1 PARALLEL [number of
            available reducers];
        STORE VA_BY_QUERY INTO 'va_by_query-f[fold number]' USING
            BinStorage();
[SUBSEQUENT TRAINING ITERATIONS:]
        VA_BY_QUERY = LOAD 'va_by_query-f[fold number]/*' USING BinStorage
            ();
NDCG = FOREACH VA_BY_QUERY GENERATE Ndcg(*);
NDCG_GRPD = GROUP NDCG ALL;
AVG_NDCG = FOREACH NDCG_GRPD GENERATE AVG(NDCG);
STORE AVG_NDCG INTO 'avg_ndcg-f[fold number]i[iteration number]';
```

7.2.3 *Testing*

The testing stage tests the best model found in the training iterations, selected on validation set NDCG@k (as calculated in the second Pig job of the training stage), by calculating the NDCG@k of this model on the test set.

Test stage Pig job:

```
REGISTER [path prefix]/lib/*.jar;
TEST_SCA = LOAD 'test_sca[fold number]/*' USING BinStorage();
TE_BY_QUERY = GROUP TEST_SCA BY $1 PARALLEL [number of available reducers];
DEFINE Ndcg udf.util.Ndcg('[neural network weights & NDCG cut-off parameter
    ]');
NDCG = FOREACH TE_BY_QUERY GENERATE Ndcg(*);
NDCG_GRPD = GROUP NDCG ALL;
AVG_NDCG = FOREACH NDCG_GRPD GENERATE AVG(NDCG);
STORE AVG_NDCG INTO 'avg_ndcg';
```

7.3 SMOOTHRANK

7.3.1 *Preprocessing*

7.3.2 *Training*

7.3.3 *Testing*

8 MAPREDUCE EXPERIMENTS

This chapter describes and discusses the results of the execution time measurements of the cluster and baseline implementations as described in 7. In addition, ranking accuracy results are included to validate the correctness of the Learning to Rank algorithm implementations.

8.1 LISTNET

8.1.1 *Ranking accuracy*

Normalized Discounted Cumulative Gain

To validate our ListNet cluster implementation we run experiments on the LETOR 3.0 OHSUMED, the LETOR 4.0 datasets MQ2007 and MQ2008 and compare their performance with the RankLib implementation of ListNet and with the official evaluation results in terms of Normalized Discounted Cumulative Gain (NDCG)@10. Hadoop job scheduling overhead is large when processing relatively small datasets (as we will explain in more detail in section 8.1.2), therefore NDCG@10 experiments are limited to five training iterations and are run only on one of the five folds. Because of the stochastic nature of the ListNet ranking procedure, which is a result of random weight initialisation prior to the first training iteration, the NDCG@10 performance is not guaranteed to be identical between multiple runs on the same input data. To account for this non-determinism of ListNet, we repeat each experiment five times and report the average as well as the standard deviation of the resulting rankings on the testing fold in terms of NDCG@10.

Table 21 shows the mean and the standard deviation of the ranking accuracy in NDCG@10 on the first data fold for our own cluster implementation (with and without the preprocessing step described in section 7.2.1) of ListNet, the RankLib implementation of ListNet, as well as the official LETOR evaluation results of ListNet as reported on the LETOR 3.0 [1] and LETOR 4.0 [2] websites. Note that the official LETOR evaluation results are structurally higher compared to the results obtained with RankLib and our cluster implementation, which can be attributed to the fact that the official evaluation runs use as many ListNet iterations as needed for convergence on the validation fold, while we limited our experiment to five iterations. The official evaluation results of the LETOR benchmarks do not report the standard deviation of their experimental results and it is not mentioned in the LETOR 3.0 and 4.0 papers [168, 169] how many times their experiments were repeated. For the ListNet step size hyperparameter we choose a value of 0.0001, which is the default value step size value in RankLib, for both the cluster and the RankLib runs. It is not clear what step size

1 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor3baseline.aspx
2 http://research.microsoft.com/en-us/um/beijing/projects/letor/letor4baseline.aspx

	OHSUMED (LETOR 3.0)	MQ2008	MQ2007
RankLib	0.1916 ± 0.0634	0.3889 ± 0.0600	0.2918 ± 0.0546
Cluster (no preprocessing)	0.2212 ± 0.0564	0.3496 ± 0.0788	0.2846 ± 0.0891
Cluster (preprocessing)	0.3395 ± 0.0952	0.4280 ± 0.0998	0.4554 ± 0.0085
Official evaluation	0.3793	0.4689	0.4767

Table 21: NDCG@10 performance on the test set of the first fold

was used in the official evaluation runs. We expect, given that LETOR aims to compare ranking methods in terms of their ranking potential, that the official LETOR benchmark results optimises the step size hyperparameter on the validation set and reported the performance on the optimal step size found with this procedure. Using this procedure one is expected to find a higher ranking accuracy than one would expect to find using a default step size value. Hyperparameter optimisation on the validation set would however not be feasible for our cluster implementation of ListNet given the Hadoop job scheduling overhead of the cluster.

The ranking accuracy of the ListNet cluster implementation without the normalisation preprocessing step seems very comparable to the ranking accuracy obtained with the RankLib ListNet implementation, which suggests that the ListNet ranking method is indeed implemented correctly. Note however that the standard deviations of the measured ranking accuracies are rather high, which leaves us unable to conclude that the ranking accuracy of RankLib ListNet and our cluster implementation are indeed equivalent in achieved ranking accuracy up to high precision.

A remarkable finding is that the cluster implementation with the normalisation preprocessing procedure shows better ranking accuracy than the RankLib version of ListNet after five iterations. It seems to be the case that the normalisation procedure enables ListNet to converge faster. It has been shown in literature, that the gradient descent optimisation procedure, which is also being used in ListNet, converges faster on normalised data [148].

8.1.2 *Speedup*

Figure 11 (linear data size axis) and Figure 12 (logarithmic data size axis) show the processing times of a single iteration of the ListNet training algorithm, using the ListNet implementation included in the RankLib library as well as the cluster implementation described in chapter 7 with different numbers of data nodes. The horizontal positions of the measurements are identical between execution method, as they are originate from the data sizes of the data sets used.

Data set	Collection	Single fold training size
MINI	GENERATED	143.38 KB
OHSUMED	LETOR 3.0	4.55 MB
MQ2008	LETOR 4.0	5.93 MB
MQ2007	LETOR 4.0	25.52 MB
MSLR-WEB10K	MSLR-WEB10K	938.01 MB
MSLR-WEB30K	MSLR-WEB30K	2.62 GB
CUSTOM-2	GENERATED	5.25 GB
CUSTOM-5	GENERATED	13.12 GB
CUSTOM-10	GENERATED	26.24 GB
CUSTOM-20	GENERATED	52.42 GB
CUSTOM-50	GENERATED	131.21 GB
CUSTOM-100	GENERATED	262.41 GB

Table 22: Description of data sets used for running time experiments

Measurements were performed on a set of data sets consisting of the LETOR 3.0 OHSUMED data set, LETOR 4.0 data sets and the MSLR-WEB10/30K data sets are used, supplemented with generated data sets that are multiplications of MSLR-WEB30K (as described in section 1.3.3). Table 22 describes the data sets used in the experiments and their single training fold data sizes.

Figures 11 and 12 show that the single-machine RankLib implementation of ListNet is very fast compared to the Hadoop MapReduce cluster implementation of ListNet for data sets that fit in memory. As soon as the amount of data processed with RankLib ListNet approaches the physical memory limit of the machine that is used for computation (16 GB for our single-machine measurements), RankLib ListNet processing time start to increase exponentially. This increase is likely to be a result of the machine needing to use the on-disk sections of virtual memory and the swapping that takes place as a result thereof. As an exception to the data sets described in Table 22, the largest data set processed with RankLib ListNet is CUSTOM-8, which is 20.99 GB in size and thereby larger than the 16 GB physical memory limit. Experiments with RankLib ListNet on CUSTOM-10 were attempted, but were stopped after not completing within 12 hours. In section 1.2 we defined speed-up as Sun and Gustafson *relative speed-up* metric [197], which defines speed-up as the number of times that the execution of the fastest single machine solution is lower than the execution time with N machines. Note that speed-up, following this definition, is not computable for larger data sets, as it is not computationally feasible to perform computation on data sets larger than CUSTOM-8 with a single machine. As an alternative, Figure 13 shows visualises speed-up by plotting cluster size against training time.

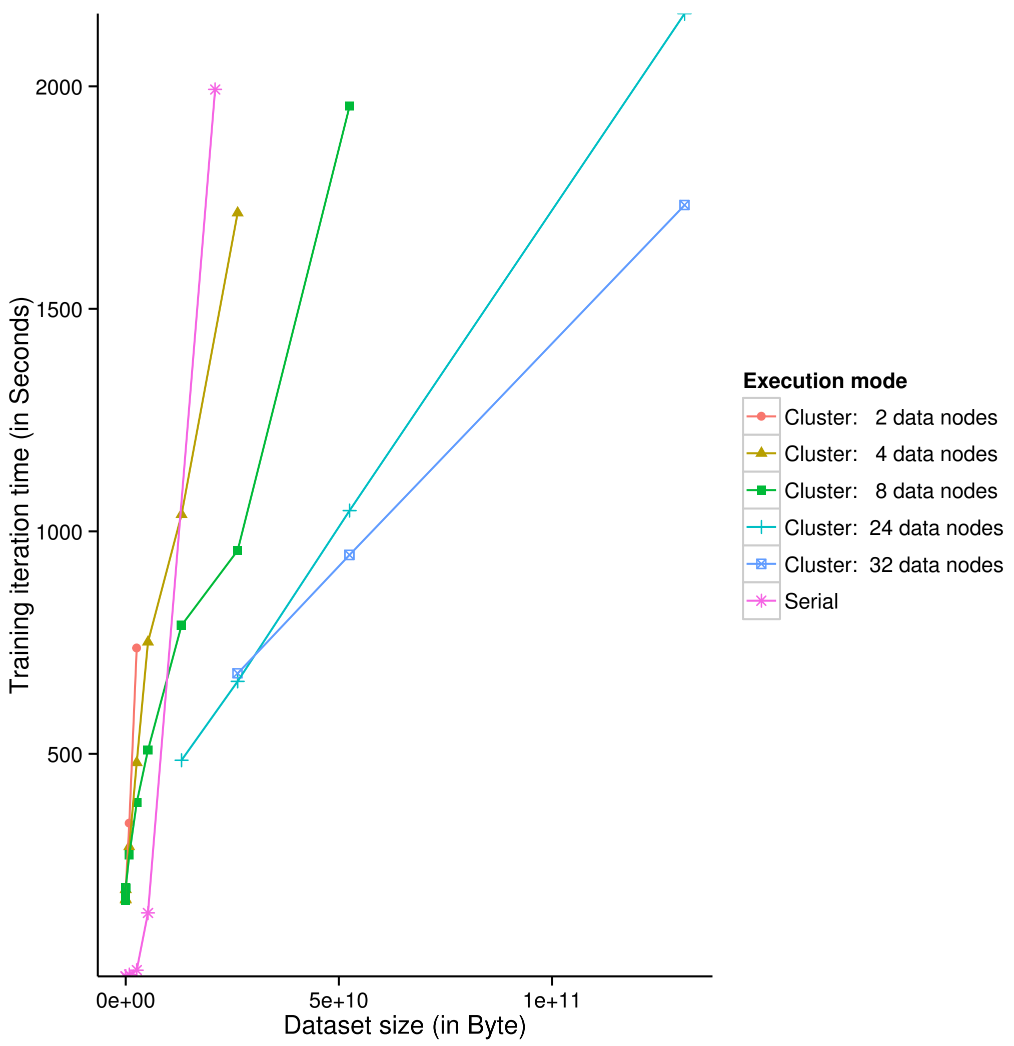

Figure 11: Processing time of a single ListNet training iteration

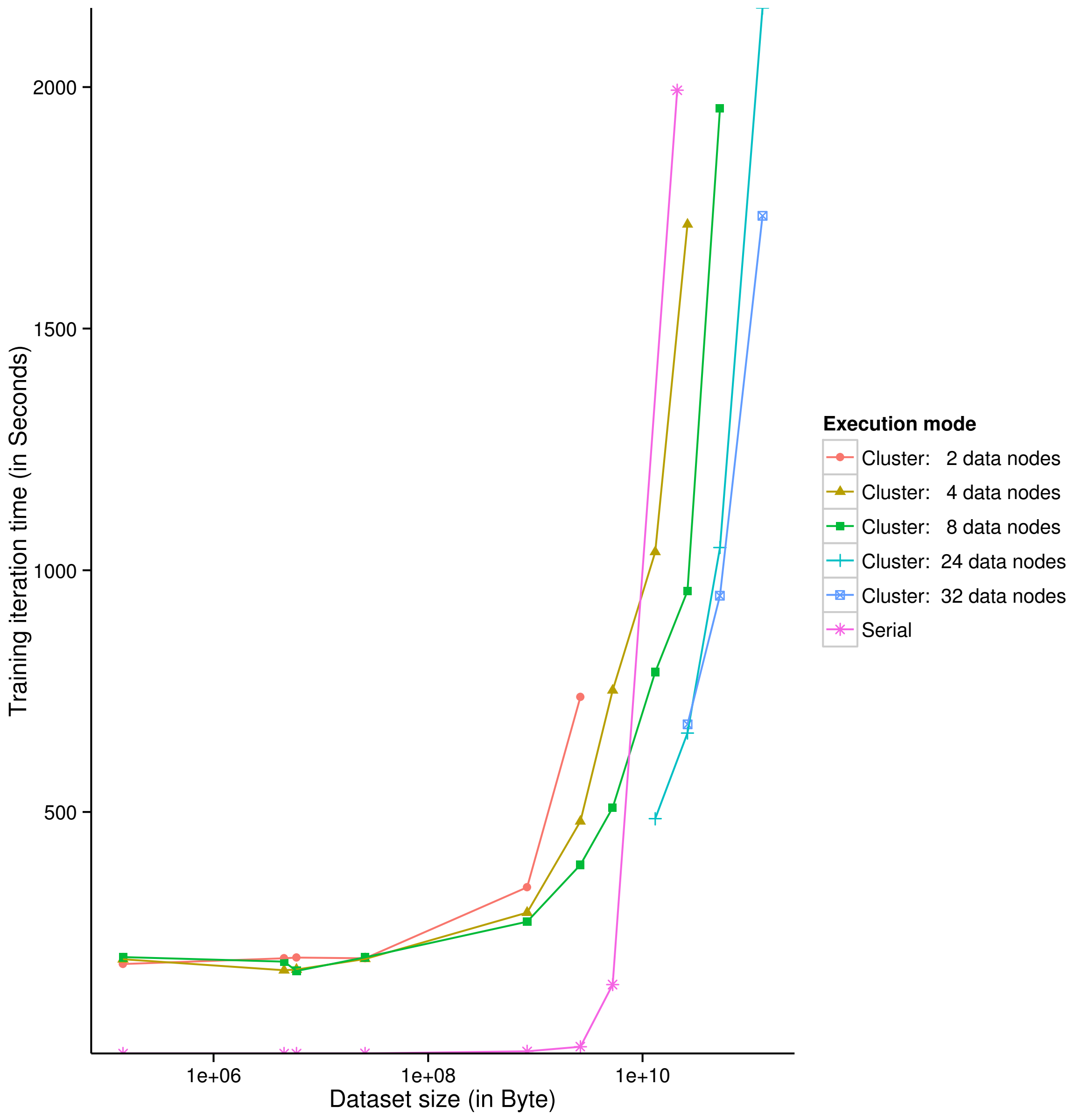

Figure 12: Processing time of a single ListNet training iteration on a logarithmic data size axis

Figure 13 shows that the speed-up achieved as function of the number of processing data nodes is sub-linear, from which we can deduct that the training time converges to a constant unit of time. Based on our measurements on the small data sets MINI, OHSUMED, MQ2008 and MQ2007, this constant time seems to be within the range of 150 to 200 seconds. This time is likely to be caused by Hadoop job scheduling overhead, this presumption is strengthened by long waiting periods between the separate MapReduce jobs that form a training iteration.

Amdahl's law states that the speed-up of a program using parallel computing is limited by the time needed for the sequential fraction of the program. A consequence of Amdahl's Law is that all parallel programs that have a non-parallelisable part have sub-linear speed-up. Behaviour in accordance with Amdahl's law can be seen in Figure 13, where the speed-up is sub-linear as a result of the existence a non-parallelisable fraction of the program.

Note however that Hadoop job scheduling overhead is independent of data set size. Therefore, the non-parallelisable fraction of the program will be smaller when the to be processed data set is larger, allowing larger speed-up values for larger data sets. From this observation we can derive that for *"large enough"* data sets, the speed-up obtained by parallelising ListNet using Hadoop MapReduce is large enough for the parallelisation to be beneficial in terms of processing time compared to the RankLib single-machine version of ListNet, even when the size of the to be processed data set would not have been memory-bounded in RankLib ListNet.

Figure 14 shows the processing speed in bytes per second. Our observation that RankLib ListNet performs very slow for data sets that do not fit in physical memory and virtual memory is needed is very notable in this graph. Furthermore, this graph shows how both increase in number in data nodes in a cluster and increase in input data size result in an increase in processing speed. The sub-linear growth of processing speed as a factor of input data size can again be explained as a result of the non-parallelisable Hadoop job scheduling part of the operation.

A result of that the Hadoop job scheduling overhead does not grow when the input data grows, is that the speed-up is linear or potentially better than linear as a function of data size. This can be seen in Figure 11. In this Figure the cluster run lines are linear or, in case of the 24 and 32 data nodes cluster lines, slightly better than linear. The lines of the clusters of 8 data nodes or smaller have high variance, this can be attributed to the fact that these clusters were mostly tested on very small datasets, leaving the measurement very dependent on the variance of the job scheduling overhead.

8.2 SMOOTHRANK

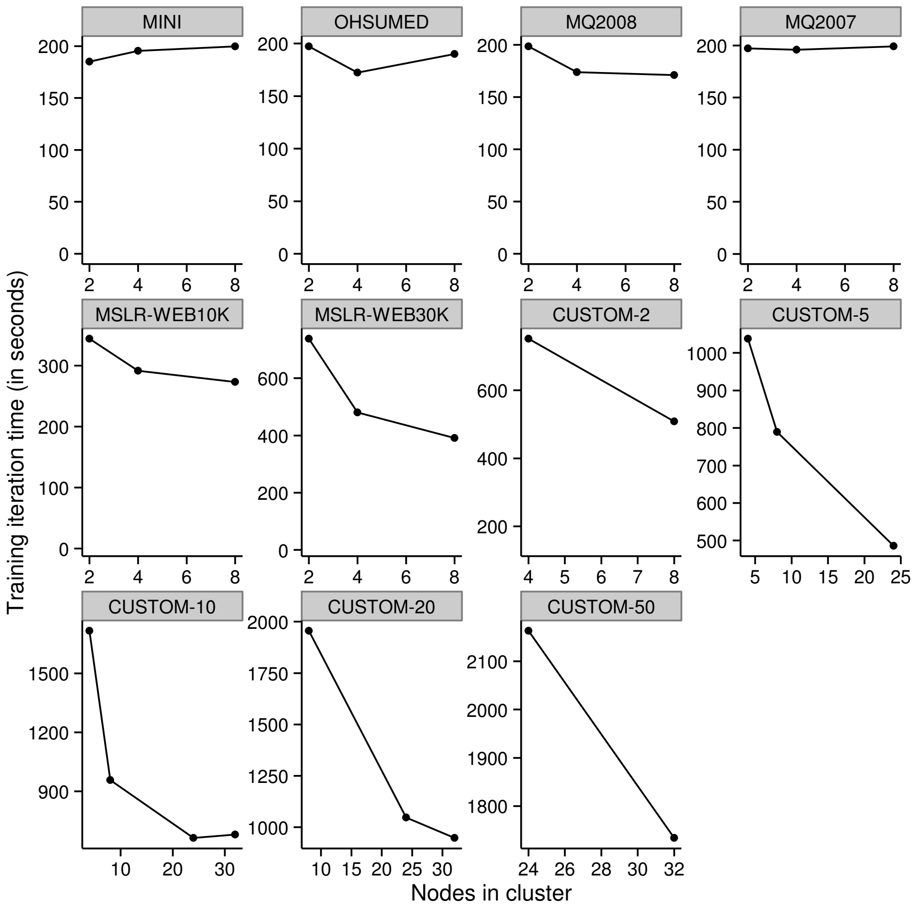

Figure 13: Processing time of a single ListNet training iteration as a function of the number of data nodes in a cluster

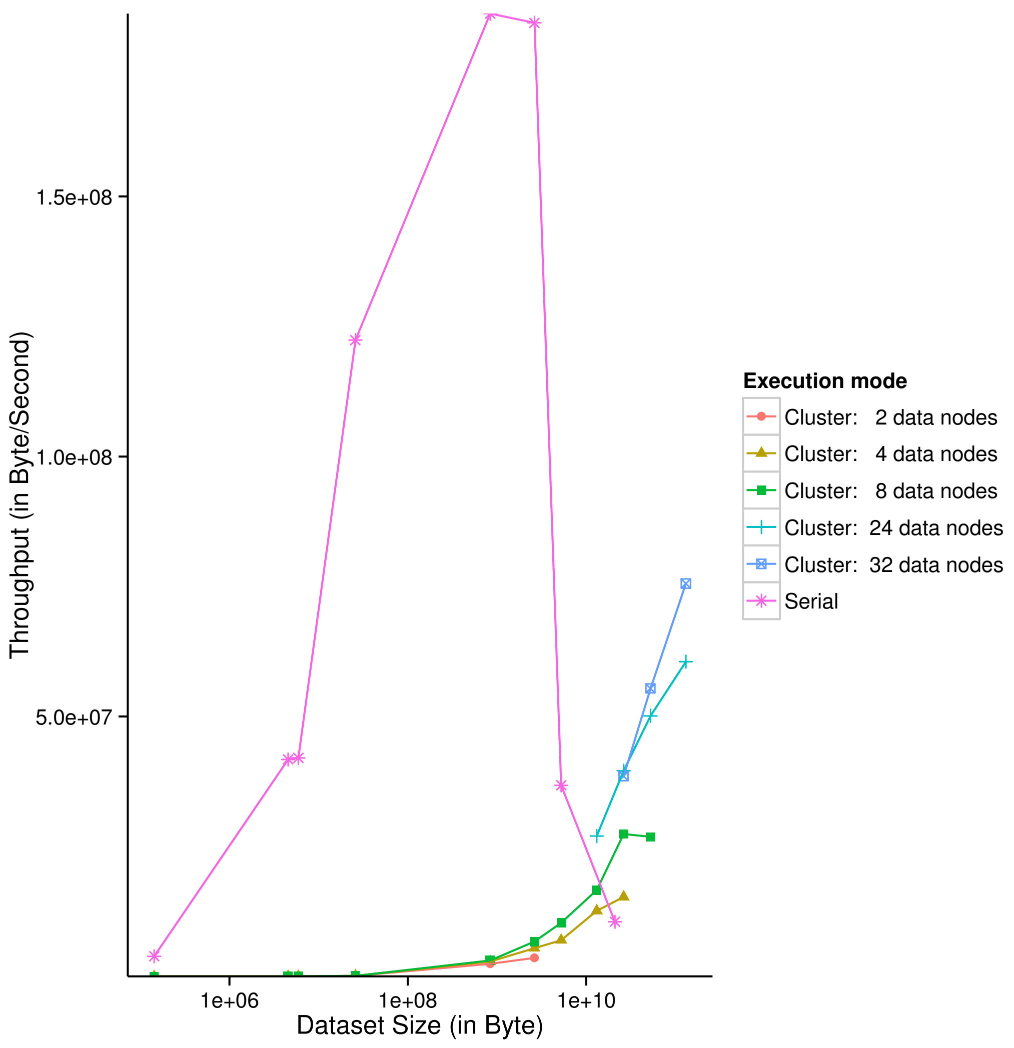

Figure 14: Processing speed of a a single ListNet training iteration on various data sets

9 CONCLUSIONS

Using our experimental results we will now reflect on our research questions stated in the 1.2 section of this thesis. We formulated the following research questions:

RQ1 What are the best performing Learning to Rank algorithms in terms of ranking accuracy on relevant benchmark data sets?

Normalised Winning Number
Ideal Winning Number

To answer this research question we proposed a new way of comparing learning to rank methods based on sparse evaluation results data on a set of benchmark datasets. Our comparison methodology comprises of two components: 1) Normalised Winning Number (NWN), which provides insight in the ranking accuracy of the learning to rank method, and 2) Ideal Winning Number (IWN), which gives insight in the degree of certainty concerning the performance of the ranking accuracy. Based on our literature search for evaluation results on well-known benchmarks collections, a lot of insight has been gained with the cross-benchmark comparison on which methods tend to perform better than others. However, no closing arguments can be formulated on which learning to rank methods are most accurate. LRUF, FSMRank, FenchelRank, SmoothRank and ListNet were the learning to rank algorithms for which it holds that no other algorithm produced more accurate rankings with a higher degree of certainty of ranking accuracy. From left to right, the ranking accuracy of these methods decreases while the certainty of the ranking accuracy increases. More evaluation runs are needed for the methods on the left side of Figure 10. Our work contributes to this by identifying promising learning to rank methods that researchers could focus on in performing additional evaluation runs.

RQ2 What is the speed-up of those Learning to Rank algorithms when executed using the MapReduce framework?

Where the definition of *relative speed-up* is used for speed-up [197]:

$$S_N = \frac{\text{execution time using one core}}{\text{execution time using } N \text{ cores}}$$

To answer this research question, we took the approach of implementing the list of algorithms found in answering the first research question, starting with the Learning to Rank method with the highest certainty on ranking accuracy, ListNet.

We found that running ListNet on a Hadoop cluster using the MapReduce computing model comes with its own cost in the form of a job scheduling overhead in the range of 150-200 seconds per training iteration. This makes Hadoop very inefficient for the processing of small data sets, where the Hadoop overhead tend to make up a large share of the total processing time. For small data sets where the constant 150-200 seconds job scheduling overhead is a large fraction of the total processing time, single-machine computation, which does not have this job scheduling overhead, is found to be faster than Hadoop MapReduce computation. For large data sets, where 150-200 seconds overhead per iteration is small compared to the total time that it would take to process the data, Hadoop MapReduce can provide a speed-up to the training process. ListNet on a single machine does not scale well to data sizes larger than the physical memory size. To process large data sets with the ListNet training algorithm, Hadoop MapReduce is a large improvement compared to a single machine.

Moreover, we found that the addition of a normalisation preprocessing step to the ListNet procedure can greatly improve the ranking accuracy of the ListNet training procedure after the first five iterations. This suggests that the addition of a normalisation preprocessing step can reduce the number of training iterations needed for convergence. Lin stated in his essay [129] that MapReduce is often good enough for tasks that are not-amenable to the MapReduce model. Lin motivates this statement in the context of iterative algorithms with the observation that these iterative algorithms can often be optimised in such a way that less iterations are needed for convergence. Our preprocessing step can improve the convergence rate of the ListNet training iteration, and therefore fits into Lin's point of view.

Most importantly, we found the training time of our cluster version of ListNet to grow better than linearly in terms of data size increase. This shows that the cluster implementation of ListNet can be used to scale the ListNet training procedure to arbitrarily large data sets, given that enough data nodes are available for computation.

No generalisations can be drawn from these results to the scalability on MapReduce of other Learning to Rank algorithms. However, we can extend our findings on job scheduling overhead and scaling benefit on large data sets to the gradient descent procedure that is used in ListNet as well as in many other Learning to Rank algorithms and in many learning algorithms in general. Other Learning to Rank algorithms using the gradient descent procedure might not scale well when the MapReduce computing model is used, but any bad scaling behaviour on MapReduce of Learning to Rank algorithms will not be caused by bad scaling of the gradient descent procedure.

10 FUTURE WORK

Potential future areas of research that follow from this thesis can be categorised into three categories. These categories of potential future work are described in the sections below.

10.1 DISTRIBUTED COMPUTING MODELS

We explored the possibilities of Hadoop MapReduce for distributed computation of Learning to Rank training algorithms. Hadoop, since the introduction of Hadoop YARN in Hadoop 2.0, offers integration with other distributed computing models, including Dryad [107], Spark [251], Storm [12] and Message Passing Interface (MPI). Of these distributed programming models, Spark is particularly promising, since Shukla et al. [188] already showed that it is able to speed-up the ListNet training procedure with much lower job scheduling overhead than that we found for MapReduce. For the newly supported programming models in Hadoop, it holds that they lack the critical mass as the distributed programming model of choice. This lack of a critical mass results in higher integration costs and less available support. The integration of new programming model into the Hadoop YARN framework is a step in the good direction to alleviate the integration cost of these programming models, but does not yet completely eliminate the problem. Cloud-based Hadoop services like Microsoft HDInsight and Amazon Elastic MapReduce.

Message Passing Interface

10.2 LEARNING TO RANK ALGORITHMS

As an answer to the first research question of this thesis, we found five Learning to Rank methods for it holds that no other algorithm produced more accurate rankings with a higher degree of certainty of ranking accuracy. These algorithms were, from highest ranking accuracy / lowest certainty to lowest ranking accuracy / highest certainty: ListNet, SmoothRank, FenchelRank, FSMRank and LRUF. In this thesis we explored the speed-up characteristics of ListNet on Hadoop MapReduce. The speed-up characteristics of SmoothRank, FenchelRank, FSMRank and LRUF are using the MapReduce computing model are still to be explored.

10.3 OPTIMISATION ALGORITHMS

Lin [129] stated that the gradient descent optimisation procedure can often be replaced with other optimisation procedures with faster convergence properties (often at the cost of being more computationally expensive per iteration), in order to make iterative machine learning algorithms perform better on MapReduce. An example of an optimisation procedures that fits this higher convergence rate at the cost of higher per-iteration computational costs is L-BFGS [134], or any other quasi-Newton optimisation method. Testing the effect that replacing the optimisation method of Learning to Rank methods has on the speed-up that can be achieved by parallelising the methods with the MapReduce model is still to be determined. Note however that replacing the optimisation method in a Learning to Rank algorithm basically turns it into a different, new, Learning to Rank method, as different optimisation methods might not be equivalent in how well they are able to find a good set of model parameters. One should therefore also explore the ranking accuracy characteristics when an existing Learning to Rank method is evaluated in combination with an optimisation algorithm that is not prescribed to be a part of the Learning to Rank method.

A

LETOR FEATURE SET

ID	Feature Description	ID	Feature Description
1	$\sum_{q_i \in q \cap d} c(q_i, d)$ of title	24	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log(\frac{\lvert C \rvert}{df(q_i)})$ of abstract
2	$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$ of title	25	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert}) \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1$ of abstract
3	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ of title	26	BM25 of abstract
4	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} + 1)$ of title	27	$\log$(BM25) of abstract
5	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{df(q_i)})$ of title	28	LMIR.DIR of abstract
6	$\sum_{q_i \in q \cap d} \log(\log(\frac{\lvert C \rvert}{df(q_i)}))$ of title	29	LMIR.JM of abstract
7	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{c(q_i,C)} + 1)$ of title	30	LMIR.ABS of abstract
8	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{df(q_i)} + 1)$ of title	31	$\sum_{q_i \in q \cap d} c(q_i, d)$ of title + abstract
9	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log(\frac{\lvert C \rvert}{df(q_i)})$ of title	32	$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$ of title + abstract
10	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert}) \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1$ of title	33	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ of title + abstract
11	BM25 of title	34	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} + 1)$ of title + abstract
12	$\log$(BM25) of title	35	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{df(q_i)})$ of title + abstract
13	LMIR.DIR of title	36	$\sum_{q_i \in q \cap d} \log(\log(\frac{\lvert C \rvert}{df(q_i)}))$ of title + abstract
14	LMIR.JM of title	37	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{c(q_i,C)} + 1)$ of title + abstract
15	LMIR.ABS of title	38	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{df(q_i)} + 1)$ of title + abstract
16	$\sum_{q_i \in q \cap d} c(q_i, d)$ of abstract	39	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log(\frac{\lvert C \rvert}{df(q_i)})$ of title + abstract
17	$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$ of abstract	40	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert}) \cdot \frac{\lvert C \rvert}{c(q_i,C)} + 1$ of title + abstract
18	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ of abstract	41	BM25 of title + abstract
19	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} + 1)$ of abstract	42	$\log$(BM25) of title + abstract
20	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{df(q_i)})$ of abstract	43	LMIR.DIR of title + abstract
21	$\sum_{q_i \in q \cap d} \log(\log(\frac{\lvert C \rvert}{df(q_i)}))$ of abstract	44	LMIR.JM of title + abstract
22	$\sum_{q_i \in q \cap d} \log(\frac{\lvert C \rvert}{c(q_i,C)} + 1)$ of abstract	45	LMIR.ABS of title + abstract
23	$\sum_{q_i \in q \cap d} \log(\frac{c(q_i,d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{df(q_i)} + 1)$ of abstract		

Table 23: Features of the LETOR 3.0 OHSUMED data set, obtained from Qin et al [168]

ID	Feature Description	ID	Feature Description
1	TF of body	36	LMIR.JM of body
2	TF of anchor	37	LMIR.JM of anchor
3	TF of title	38	LMIR.JM of title
4	TF of URL	39	LMIR.JM of URL
5	TF of whole document	40	LMIR.JM of whole document
6	IDF of body	41	Sitemap based term propagation
7	IDF of anchor	42	Sitemap based score propagation
8	IDF of title	43	Hyperlink based score propagation: weighted in-link
9	IDF of URL	44	Hyperlink based score propagation: weighted out-link
10	IDF of whole document	45	Hyperlink based score propagation: uniform out-link
11	TF-IDF of body	46	Hyperlink based feature propagation: weighted in-link
12	TF-IDF of anchor	47	Hyperlink based feature propagation: weighted out-link
13	TF-IDF of title	48	Hyperlink based feature propagation: uniform out-link
14	TF-IDF of URL	49	HITS authority
15	TF-IDF of whole document	50	HITS hub
16	Document length of body	51	PageRank
17	Document length of anchor	52	HostRank
18	Document length of title	53	Topical PageRank
19	Document length of URL	54	Topical HITS authority
20	Document length of whole document	55	Topical HITS hub
21	BM25 of body	56	In-link number
22	BM25 of anchor	57	Out-link number
23	BM25 of title	58	Number of slashes in URL
24	BM25 of URL	59	Length of URL
25	BM25 of whole document	60	Number of child page
26	LMIR.ABS [252] of body	61	BM25 of extracted title
27	LMIR.ABS of anchor	62	LMIR.ABS of extracted title
28	LMIR.ABS of title	63	LMIR.DIR of extracted title
29	LMIR.ABS of URL	64	LMIR.JM of extracted title
30	LMIR.ABS of whole document		
31	LMIR.DIR of body		
32	LMIR.DIR of anchor		
33	LMIR.DIR of title		
34	LMIR.DIR of URL		
35	LMIR.DIR of whole document		

Table 24: Features of the LETOR 3.0 .gov data set, obtained from Qin et al [168]

B RAW DATA FOR COMPARISON ON NDCG@3 AND NDCG@5

Method	NWN on NDCG@3	NWN on NDCG@5	# of data sets for NDCG@3	# of data sets for NDCG@5
AdaRank-MAP	0.34862385	0.3883929	12	12
AdaRank-NDCG	0.30733945	0.3258929	12	12
ApproxNDCG	0.79487179	0.7500000	1	1
BagBoo	0.84782609	0.8400000	2	1
BL-MART	0.95238095	0.7200000	2	1
BoltzRank-Pair	0.83333333	0.8349515	4	4
BoltzRank-Single	0.75490196	0.7184466	4	4
BT	0.72727273	0.7878788	3	3
C-CRF	-	0.9500000	0	2
CoList	1.00000000	-	1	0
DCMP	0.54591837	0.5078534	9	9
EnergyNDCG	0.36363636	0.3777778	2	2
FBPCRank	0.41463415	0.5529412	3	3
FenchelRank	0.77419355	0.7500000	5	5
FocusedBoost	0.37500000	0.4545455	2	2
FocusedNet	0.45833333	0.6363636	2	2
FocusedSVM	0.25000000	0.2727273	2	2
FPRank	-	0.9000000	0	1
FRank	0.30845771	0.2849462	11	10
FSMRank	0.83333333	0.8775510	4	4
FSMSVM	0.22916667	0.4081633	4	4
GAS-E	0.37500000	0.4693878	4	4
GPRank	0.87096774	0.7252747	3	3
GroupCE	0.73118280	-	3	0
GroupMLE	0.52688172	-	3	0
IntervalRank	0.58974359	0.3750000	1	1
IPRank	0.93548387	0.8131868	3	3
Kernel-PCA RankBoost	-	0.2857143	0	3
KL-CRF	0.59459459	0.5789474	2	2
LambdaMART	0.57142857	-	2	0
LambdaNeuralRank	1.00000000	1.0000000	1	1
LambdaRank	0.20000000	0.2000000	1	1

LARF	0.98924731	0.9890110	3	3
Linear Regression	0.07142857	0.1099476	9	9
ListMLE	0.00000000	-	1	0
ListNet	0.44954128	0.4910714	12	12
ListReg	0.73118280	0.6923077	3	3
LRUF	0.98230088	0.9816514	4	4
MHR	0.75000000	0.6000000	1	1
NewLoss	0.51612903	0.4285714	3	3
OWPC	0.65000000	-	6	0
PERF-MAP	0.38938053	0.2660550	4	4
Q.D.KNN	-	0.3205128	0	3
RankAggNDCG	-	0.5000000	0	3
RankBoost	0.32110092	0.2794118	12	10
RankDE	-	0.5384615	0	1
RankELM (pairwise)	0.61538462	0.6500000	1	1
RankELM (pointwise)	0.69230769	0.7000000	1	1
RankNet	0.18867925	0.2857143	1	3
Rank-PMBGP	-	0.7692308	0	1
RankSVM	0.30097087	0.3612565	12	11
RankSVM-Primal	0.39204545	0.4508671	8	8
RankSVM-Struct	0.35204082	0.4136126	9	9
RCP	-	0.5757576	0	3
REG-SHF-SDCG	0.38461538	0.4500000	1	1
Ridge Regression	0.40880503	0.3333333	7	7
RSRank	0.57291667	0.5306122	4	4
SmoothRank	0.60377358	0.6339869	7	7
SoftRank	0.23076923	0.2750000	1	1
SortNet	0.25000000	0.5147059	2	4
SparseRank	0.82242991	0.8173077	4	4
SVD-RankBoost	-	0.2727273	0	3
SVM^{MAP}	0.28930818	0.3801170	7	8
SwarmRank	-	0.1538462	0	1
TGRank	0.54166667	0.6122449	4	4
TM	0.59090909	0.7575758	3	3

Table 25: Raw Normalised Winning Number data calculated on NDCG@3 and NDCG@5 evaluation results

C RAW DATA FOR COMPARISON ON NDCG@10 AND MAP

Method	NWN on NDCG@10	NWN on MAP	# of data sets for NDCG@10	# of data sets for MAP
AdaRank-MAP	0.36480687	0.320610687	13	12
AdaRank-NDCG	0.31578947	0.286259542	16	12
ADMM	0.44444444	-	1	0
ApproxNDCG	0.86111111	-	1	0
ApproxAP	-	0.500000000	0	2
BagBoo	-	0.654545455	0	2
Best Single Feature	0.16149068	-	8	0
BL-MART	-	0.803571429	0	3
BoltzRank-Pair	-	0.580419580	0	5
BoltzRank-Single	-	0.433566434	0	5
BT	-	0.750000000	0	3
CA	0.65217391	-	4	0
CCRank	-	0.615384615	0	2
CoList	0.16666667	-	1	0
Consistent-RankCosine	0.76923077	-	2	0
DCMP	0.58883249	-	9	0
DirectRank	0.92307692	-	2	0
EnergyNDCG	0.41463415	-	2	0
FenchelRank	0.76229508	0.641791045	5	5
FocusedBoost	0.68627451	-	2	0
FocusedNet	0.86274510	-	2	0
FocusedSVM	0.60784314	-	2	0
FRank	0.30288462	0.262295082	11	11
FSMRank	0.86206897	0.578947368	5	7
FSMSVM	0.54255319	0.350000000	4	4
GAS-E	0.45744681	0.410000000	4	4
GP	0.66666667	0.500000000	2	2
GPRank	0.65909091	0.817307692	3	3
GRankRLS	0.28947368	-	2	0
GroupCE	0.72727273	0.721153846	3	3
GroupMLE	0.62500000	0.653846154	3	3
IntervalRank	-	0.315789474	0	1

IPRank	0.79545455	0.851351351	3	6
KeepRank	-	0.538461538	0	3
LAC-MR-OR	0.66666667	0.764192140	2	12
LambdaMART	1.00000000	0.678571429	1	3
LambdaNeuralRank	1.00000000	-	1	0
LambdaRank	0.57142857	-	2	0
LARF	0.98863636	0.980769231	3	3
Linear Regression	0.08287293	0.065000000	8	8
ListMLE	0.02127660	0.009615385	4	3
ListNet	0.59821429	0.450381679	12	12
ListReg	-	0.432692308	0	3
LRUF	0.98181818	0.968000000	4	4
MCP	-	0.571428571	0	2
MHR	0.62500000	0.000000000	1	1
MultiStageBoost	-	0.136363636	0	2
NewLoss	0.39772727	-	3	0
OWPC	-	0.624113475	0	6
PERF-MAP	0.20000000	0.768000000	4	4
PermuRank	-	0.409090909	0	3
Q.D.KNN	0.50000000	0.558441558	3	3
RandomForest	0.42236025	0.438888889	8	8
RankAggNDCG	0.87837838	0.792207792	3	3
RankBoost	0.39357430	0.313432836	17	14
RankCSA	-	0.916666667	0	2
RankDE	1.000000000	0.18181818	1	1
RankELM (pairwise)	0.69444444	0.514285714	1	2
RankELM (pointwise)	0.80555556	0.542857143	1	2
RankMGP	-	0.222222222	0	1
RankNet	0.59154930	-	5	0
Rank-PMBGP	0.27272727	0.875000000	1	1
RankRLS	0.36842105	-	2	0
RankSVM	0.44957983	0.340000000	17	13
RankSVM-Primal	0.45911950	0.351955307	7	7
RankSVM-Struct	0.44670051	0.362385321	9	9
RCP	0.74074074	0.363636364	3	3
REF-SHF-SDCG	-	0.657894737	0	1
RE-QR	-	0.865921788	0	7
Ridge Regression	0.36477987	0.290502793	7	7
RSRank	0.62765957	0.660000000	4	4

SmoothGrad	0.38461538	-	2	0
SmoothRank	0.64150943	0.530726257	7	7
SoftRank	0.61111111	-	1	0
SortNet	0.56666667	0.500000000	4	2
SparseRank	0.79439252	-	4	0
SVD-RankBoost	0.55555556	0.568181818	3	3
SVM^{MAP}	0.35911602	0.349775785	8	10
SwarmRank	0.09090909	0.125000000	1	1
TGRank	0.50000000	0.460000000	4	4
TM	-	0.613636364	0	3
VFLR	-	0.974358974	0	2

Table 26: Raw Normalised Winning Number data calculated on NDCG@10 and MAP evaluation results

D RAW DATA ON NORMALISED WINNING NUMBER FOR CROSS-COMPARISON

Method	Winning Number	IWN	NWN
AdaRank-MAP	332	937	0.35432231
AdaRank-acsNDCG	293	951	0.30809674
ADMM	4	9	0.44444444
ApproxAP	33	66	0.50000000
ApproxNDCG	92	115	0.80000000
BagBoo	96	126	0.76190476
Best Single Feature	26	161	0.16149068
BL-MART	83	102	0.81372549
BoltzRank-Pair	254	348	0.72988506
BoltzRank-Single	213	348	0.61206897
BT	75	99	0.75757576
C-CRF	19	20	0.95000000
CA	15	23	0.65217391
CCRank	24	39	0.61538462
CoList	2	7	0.28571429
Consistent-RankCosine	10	13	0.76923077
DCMP	320	584	0.54794521
DirectRank	12	13	0.92307692
EnergyNDCG	50	130	0.38461538
FBPCRank	81	167	0.48502994
FenchelRank	368	504	0.73015873
FocusedBoost	73	143	0.51048951
FocusedNet	94	143	0.65734266
FocusedSVM	55	143	0.38461538
FP-Rank	18	20	0.90000000
FRank	242	839	0.28843862
FSMRank	365	481	0.75883576
FSM^{SVM}	148	388	0.38144330
GAS-E	166	388	0.42783505
GP	7	12	0.58333333
GPRank	290	376	0.77127660
GRankRLS	11	38	0.28947368

GroupCE	207	285	0.72631579
GroupMLE	172	285	0.60350877
IntervalRank	50	117	0.42735043
IPRank	357	420	0.85000000
KeepRank	56	104	0.53846154
Kernel-PCA RankBoost	26	91	0.28571429
KL-CRF	44	75	0.58666667
LAC-MR-OR	179	235	0.76170213
LambdaMART	54	81	0.66666667
LambdaNeuralRank	15	15	1.00000000
LambdaRank	10	24	0.41666667
LARF	371	376	0.98670213
Linear Regression	63	761	0.08278581
ListMLE	3	199	0.01507538
ListNet	460	928	0.49568966
ListReg	176	288	0.61111111
LRUF	447	457	0.97811816
MCP	40	70	0.57142857
MHR	17	41	0.41463415
MultiStageBoost	6	44	0.13636364
NewLoss	122	272	0.44852941
OWPC	166	261	0.63601533
PERF-MAP	191	457	0.41794311
PermuRank	18	44	0.40909091
Q.D.KNN	105	229	0.45851528
RandomForest	147	341	0.43108504
Rank-PMBGP	27	40	0.67500000
RankAggNDCG	165	229	0.72052402
RankBoost	309	939	0.32907348
RankCSA	33	36	0.91666667
RankDE	25	40	0.62500000
RankELM (pairwise)	111	185	0.60000000
RankELM (pointwise)	122	185	0.65945946
RankMGP	4	18	0.22222222
RankNet	66	173	0.38150289
RankRLS	14	38	0.36842105
RankSVM	323	885	0.36497175
RankSVM-Primal	283	687	0.41193595
RankSVM-Struct	315	793	0.39276808

RCP	55	104	0.52884615
RE-QR	155	179	0.86592179
REG-SHF-SDCG	58	117	0.49572650
Ridge Regression	226	650	0.34769231
RSRank	232	388	0.59793814
SmoothGrad	5	13	0.38461538
SmoothRank	390	650	0.60000000
SoftRank	42	115	0.36521739
SortNet	113	238	0.47478992
SparseRank	258	318	0.81132075
SVD-RankBoost	49	104	0.47115385
SVM^{MAP}	254	734	0.34604905
SwarmRank	5	40	0.12500000
TGRank	205	388	0.52835052
TM	65	99	0.65656566
VFLR	38	39	0.97435897

Table 27: Raw Normalised Winning Number data calculated cross-metric

BIBLIOGRAPHY

[1] Acharyya, S. *Learning to Rank in Supervised and Unsupervised Settings using Convexity and Monotonicity*. PhD thesis, The University of Texas, Austin, 2013.

[2] Acharyya, S., Koyejo, O., and Ghosh, J. Learning to Rank with Bregman Divergences and Monotone Retargeting. In *Proceedings of the 28th Conference on Uncertainty in artificial intelligence (UAI)* (2012).

[3] Adams, R. P., and Zemel, R. S. Ranking via Sinkhorn Propagation. *arXiv preprint arXiv:1106.1925* (2011).

[4] Adomavicius, G., and Tuzhilin, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 17*, 6 (2005), 734–749.

[5] Agarwal, S., and Collins, M. Maximum Margin Ranking Algorithms for Information Retrieval. In *Proceedings of the 32nd European Conference on Information Retrieval Research (ECIR)* (2010), pp. 332–343.

[6] Agarwal, S., Dugar, D., and Sengupta, S. Ranking Chemical Structures for Drug Discovery: a New Machine Learning Approach. *Journal of Chemical Information and Modeling (JCIM) 50*, 5 (2010), 716–731.

[7] Ah-Pine, J. Data Fusion in Information Retrieval Using Consensus Aggregation Operators. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)* (2008), vol. 1, pp. 662–668.

[8] Airola, A., Pahikkala, T., and Salakoski, T. Large scale training methods for linear rankrls. In *Proceedings of the ECML/PKDD-10 Workshop on Preference Learning* (2010).

[9] Airola, A., Pahikkala, T., and Salakoski, T. Training Linear Ranking SVMs in Linearithmic Time using Red-Black Trees. *Pattern Recognition Letters 32*, 9 (2011), 1328–1336.

[10] Alcântara, O. D., Pereira Jr, Á. R., Almeida, H. M., Gonçalves, M. A., Middleton, C., and Baeza-Yates, R. Wcl2r: A benchmark collection for learning to rank research with clickthrough data. *Journal of Information and Data Management (JIDM) 1*, 3 (2010), 551.

[11] Alejo, O., Fernández-Luna, J. M., Huete, J. F., and Pérez-Vázquez, R. Direct Optimization of Evaluation Measures in Learning to Rank Using Particle Swarm. In *Proceedings of the Workshop on Database and Expert Systems Applications (DEXA)* (2010), pp. 42–46.

[12] Aniello, L., Baldoni, R., and Querzoni, L. Adaptive Online Scheduling in Storm. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems* (2013), pp. 207–218.

[13] Argentini, A. *Ranking Aggregation Based on Belief Function Theory*. PhD thesis, University of Trento, 2012.

[14] Asadi, N. *Multi-Stage Search Architectures for Streaming Documents*. PhD thesis, University of Maryland, 2013.

[15] Asadi, N., and Lin, J. Training Efficient Tree-Based Models for Document Ranking. In *Proceedings of the 25th European Conference on Information Retrieval Research (ECIR)* (2013), vol. 7814, pp. 146–157.

[16] Asadi, N., Lin, J., and de Vries, A. Runtime Optimizations for Prediction with Tree-Based Models. *IEEE Transactions on Knowledge and Data Engineering (TKDE) PP*, 99 (2013), 1.

[17] Avery, C. Giraph: Large-Scale Graph Processing Infrastruction on Hadoop.

[18] Banerjee, S., Dubey, A., Machchhar, J., and Chakrabarti, S. Efficient and Accurate Local Learning for Ranking. In *Proceedings of the SIGIR 2009 Workshop on Learning to Rank for Information Retrieval* (2009), pp. 1–8.

[19] Bartlett, P. L., Jordan, M. I., and McAuliffe, J. D. Convexity, Classification, and Risk Bounds. *Journal of the American Statistical Association (JASA) 101*, 473 (2006), 138–156.

[20] Ben-Haim, Y., and Tom-Tov, E. A Streaming Parallel Decision Tree Algorithm. *Journal of Machine Learning Research (JMLR) 11* (2010), 849–872.

[21] Benbouzid, D., Busa-Fekete, R., and Kégl, B. Fast classification using sparse decision DAGs. In *Proceedings of the 29th International Conference on Machine Learning (ICML)* (2012), pp. 951–958.

[22] Bian, J. *Contextualized Web Search: Query-dependent Ranking and Social Media Search*. PhD thesis, Georgia Institute of Technology, 2010.

[23] Bian, J., Li, X., Li, F., Zheng, Z., and Zha, H. Ranking Specialization for Web Search: A Divide-and-conquer Approach by Using Topical RankSVM. In *Proceedings of the 19th International Conference on World Wide Web (WWW)* (2010).

[24] Bidoki, A. M. Z., and Thom, J. A. Combination of Documents Features Based on Simulated Click-through Data. In *Proceedings of the 31st European Conference on Information Retrieval Research (ECIR)* (2009), vol. 5478, pp. 538–545.

[25] Bollegala, D., Noman, N., and Iba, H. RankDE: Learning a Ranking Function for Information Retrieval using Differential Evolution. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (2011), pp. 1771–1778.

[26] Boyd, S., Cortes, C., Jiang, C., Mohri, M., Radovanovic, A., and Skaf, J. Large-scale Distributed Optimization for Improving Accuracy at the Top. In *Proceedings of the NIPS Workshop on Optimization for Machine Learning* (2012).

[27] Bravo-Marquez, F., and Manriquez, M. A Zipf-like Distant Supervision Approach for Multi-Document Summarization Using Wikinews Articles. In *String Processing and Information Retrieval (SPIRE)* (2012), pp. 143–154.

[28] Breiman, J. H., Friedman, R. A., Stone, J. C., and Olshen, L. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[29] Breiman, L. Bagging Predictors. *Machine Learning 24*, 2 (1996), 123–140.

[30] Bridle, J. S. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In *NATO ASI Series F: Computer and Systems Sciences* (1990), pp. 227–236.

[31] Buffoni, D., Gallinari, P., Usunier, N., and Calauzènes, C. Learning Scoring Functions with Order-Preserving Losses and Standardized Supervision. In *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011), pp. 825–832.

[32] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to Rank using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (2005), pp. 89–96.

[33] Burges, C. J. C. From RankNet to LambdaRank to LambdaMART: An Overview. Tech. rep., Microsoft Research, 2010.

[34] Burges, C. J. C., Ragno, R., and Le, Q. V. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems (NIPS)* (2006), vol. 6, pp. 193–200.

[35] Burges, C. J. C., Svore, K. M., Bennett, P. N., Pastusiak, A., and Wu, Q. Learning to Rank Using an Ensemble of Lambda-Gradient Models. *Journal of Machine Learning Research (JMLR) 14* (2011), 25–35.

[36] Busa-Fekete, R., Kégl, B., Éltető, T., and Szarvas, G. Tune and Mix: Learning to Rank using Ensembles of Calibrated Multi-Class Classifiers. *Machine learning 93*, 2-3 (2013), 261–292.

[37] Busa-Fekete, R., Szarvas, G., Elteto, T., and Kégl, B. An Apple-to-Apple Comparison of Learning-to-Rank Algorithms in Terms of Normalized Discounted Cumulative Gain. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)* (2012), vol. 242.

[38] Cai, F., Guo, D., Chen, H., and Shu, Z. Your Relevance Feedback Is Essential: Enhancing the Learning to Rank Using the Virtual Feature Based Logistic Regression. *PloS One 7*, 12 (2012), e50112.

[39] Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., and Li, H. Learning to Rank: from Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML)* (2007), pp. 129–136.

[40] Carvalho, V. R., Elsas, J. L., Cohen, W. W., and Carbonell, J. G. Suppressing Outliers in Pairwise Preference Ranking. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)* (2008), pp. 1487–1488.

[41] Chakrabarti, S., Khanna, R., Sawant, U., and Bhattacharyya, C. Structured Learning for Non-smooth Ranking Losses. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2008), pp. 88–96.

[42] Chang, E. Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., and Cui, H. PSVM: Parallelizing Support Vector Machines on Distributed Computers. In *Advances in Neural Information Processing Systems (NIPS)* (2007), pp. 257–264.

[43] Chang, X., and Zheng, Q. Preference Learning to Rank with Sparse Bayesian. In *Proceedings of the IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) - Volume III* (2009), pp. 143–146.

[44] Chapelle, O., and Chang, Y. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research (JMLR) 14* (2011), 1–24.

[45] Chapelle, O., Chang, Y., and Liu, T. Y. Future Directions in Learning to Rank. *JMLR Workshop and Conference Proceedings 14* (2011), 91–100.

[46] Chapelle, O., Metlzer, D., Zhang, Y., and Grinspan, P. Expected Reciprocal Rank for Graded Relevance. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (Nov. 2009), p. 621.

[47] Chapelle, O., and Wu, M. Gradient Descent Optimization of Smoothed Information Retrieval Metrics. *Information Retrieval 13*, 3 (2010), 216–235.

[48] Che, S., Li, J., Sheaffer, J. W., Skadron, K., and Lach, J. Accelerating Compute-Intensive Applications with GPUs and FPGAs. In *Proceedings of the Symposium on Application Specific Processors (SASP)* (2008), pp. 101–107.

[49] Chen, D., Xiong, Y., Yan, J., Xue, G. R., Wang, G., and Chen, Z. Knowledge Transfer for Cross Domain Learning to Rank. *Information Retrieval 13*, 3 (2010), 236–253.

[50] Chen, K., Lu, R., Wong, C. K., Sun, G., Heck, L., and Tseng, B. Trada: Tree Based Ranking Function Adaptation. In *Proceedings of the 17th ACM conference on Information and knowledge management (CIKM)* (2008), pp. 1143–1152.

[51] Chen, M., Weinberger, K. Q., Chapelle, O., Kedem, D., and Xu, Z. Classifier Cascade for Minimizing Feature Evaluation Cost. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2012), pp. 218–226.

[52] Chen, X. W., Wang, H., and Lin, X. Learning to Rank with a Novel Kernel Perceptron Method. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 505–512.

[53] Chu, C., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. Map-Reduce for Machine Learning on Multicore. vol. 19, p. 281.

[54] Chun, B.-G., Condie, T., Curino, C., Douglas, C., Matusevych, S., Myers, B., Narayanamurthy, S., Ramakrishnan, R., Rao, S., and Rosen, J. REEF: Retainable Evaluator Execution Framework. *Proceedings of the VLDB Endowment 6*, 12 (2013), 1370–1373.

[55] Ciaramita, M., Murdock, V., and Plachouras, V. Online Learning from Click Data for Sponsored Search. In *Proceedings of the 17th International Conference on World Wide Web* (2008), pp. 227–236.

[56] Cleverdon, C. W., and Keen, M. Aslib Cranfield Research Project - Factors Determining the Performance of Indexing Systems; Volume 2, Test Results. Tech. rep., Cranfield University, 1966.

[57] Cossock, D., and Zhang, T. Subset Ranking using Regression. In *Proceedings of the 19th Annual Conference on Learning Theory (COLT)*. 2006, pp. 605–619.

[58] Crammer, K., and Singer, Y. Pranking with Ranking. In *Advances in Neural Information Processing Systems (NIPS)* (2001), pp. 641–647.

[59] Craswell, N., Hawking, D., Wilkinson, R., and Wu, M. Overview of the TREC 2003 Web Track. In *Proceedings of the 12th Text Retrieval Conference (TREC)* (2003), vol. 3, p. 12th.

[60] Dammak, F., Kammoun, H., and Ben Hamadou, A. An Extension of RankBoost for Semi-Supervised Learning of Ranking Functions. In *Proceedings of the Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM* (2011), pp. 49–54.

[61] De, A. On the Role of Compensatory Operators in Fuzzy Result Merging for Metasearch. In *Proceedings of the Fifth Internation Conference on Pattern Recognition and Machine Intelligence (RReMI)* (2013), vol. 8251, pp. 551–556.

[62] De, A., and Diaz, E. Fuzzy Analytical Network Models for Metasearch. In *Revised and Selected Papers of the International Joint Conference on Computational Intelligence (IJCCI)*, vol. 399. 2012, pp. 197–210.

[63] De, A., Diaz, E., and Raghavan, V. V. Search Engine Result Aggregation using Analytical Hierarchy Process. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)* (2010), vol. 3, pp. 300–303.

[64] De, A., and Diaz, E. D. A Fuzzy Ordered Weighted Average (OWA) Approach to Result Merging for Metasearch Using the Analytical Network Process. In *Proceedings of the Second International Conference on Emerging Applications of Information Technology (EAIT)* (2011), pp. 17–20.

[65] De, A., Diaz, E. D., and Raghavan, V. V. Weighted Fuzzy Aggregation for Metasearch: An Application of Choquet Integral. In *Proceeings of the 14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)* (2012), pp. 501–510.

[66] de Almeida, H. M., Gonçalves, M. A., Cristo, M., and Calado, P. A Combined Component Approach for Finding Collection-adapted Ranking Functions based on Genetic Programming. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 399–406.

[67] De Sousa, D. X., Rosa, T. C., Martins, W. S., Silva, R., and Gonçalves, M. A. Improving On-Demand Learning to Rank through Parallelism. In *Proceedings of the 13th International Conference on Web Information Systems Engineering (WISE)* (2012), pp. 526–537.

[68] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM 51*, 1 (2004), 107–113.

[69] Deng, L., He, X., and Gao, J. Deep Stacking Networks for Information Retrieval. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2013), pp. 3153–3157.

[70] Derhami, V., Khodadadian, E., Ghasemzadeh, M., and Zareh Bidoki, A. M. Applying Reinforcement Learning for Web Pages Ranking Algorithms. *Applied Soft Computing 13*, 4 (Apr. 2013), 1686–1692.

[71] Desarkar, M., Joshi, R., and Sarkar, S. Displacement Based Unsupervised Metric for Evaluating Rank Aggregation. In *Proceedings of the Fourth International Conference on Pattern Recognition and Machine Intelligence (PReMI)* (2011), vol. 6744, pp. 268–273.

[72] Diaz-Aviles, E., Nejdl, W., and Schmidt-Thieme, L. Swarming to rank for information retrieval. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (2009), pp. 9–16.

[73] Ding, W., Qin, T., and Zhang, X. D. Learning to Rank with Supplementary Data. In *Proceedings of the Sixth Asia Information Retrieval Societies Conference (AIRS)* (2010), vol. 6458, pp. 478–489.

[74] Dubey, A., Machchhar, J., Bhattacharyya, C., and Chakrabarti, S. Conditional Models for Non-smooth Ranking Loss Functions. In *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM)* (2009), pp. 129–138.

[75] Duh, K., and Kirchhoff, K. Learning to Rank with Partially-labeled Data. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), pp. 251–258.

[76] Duh, K., and Kirchhoff, K. Semi-supervised Ranking for Document Retrieval. *Computational Speech and Language 25*, 2 (Apr. 2011), 261–281.

[77] Duh, K., Suzuki, J., and Nagata, M. Distributed Learning-to-Rank on Streaming Data using Alternating Direction Method of Multipliers. In *Proceedings of the NIPS Big Learning Workshop* (2011).

[78] Elsas, J. L., Carvalho, V. R., and Carbonell, J. G. Fast Learning of Document Ranking Functions with the Committee Perceptron. In *Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM)* (2008), pp. 55–64.

[79] Feldman, V., Guruswami, V., Raghavendra, P., and Wu, Y. Agnostic Learning of Monomials by Halfspaces is Hard. *SIAM Journal on Computing 41*, 6 (2012), 1558–1590.

[80] Freno, N., Papini, T., and Diligenti, M. Learning to Rank using Markov Random Fields. In *Proceeings of the 10th International Conference on Machine Learning and Applications and Workshops (ICMLA)* (2011), vol. 2, pp. 257–262.

[81] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An Efficient Boosting Algorithm for Combining Preferences. *The Journal of Machine Learning Research (JMLR) 4* (Dec. 2003), 933–969.

[82] Freund, Y., and Schapire, R. E. A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *Journal of computer and system sciences (JCSS) 55*, 1 (1997), 119–139.

[83] Friedman, J. H. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* (2001), 1189–1232.

[84] Friedman, J. H. Stochastic Gradient Boosting. *Computational Statistics & Data Analysis 38*, 4 (2002), 367–378.

[85] Ganjisaffar, Y. *Tree Ensembles for Learning to Rank*. PhD thesis, University of California, Irvine, 2011.

[86] Ganjisaffar, Y., Caruana, R., and Lopes, C. V. Bagging Gradient-Boosted Trees for High Precision, Low Variance Ranking Models. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 85–94.

[87] Ganjisaffar, Y., Debeauvais, T., Javanmardi, S., Caruana, R., and Lopes, C. V. Distributed Tuning of Machine Learning Algorithms using MapReduce Clusters. In *Proceedings of the Third Workshop on Large Scale Data Mining: Theory and Applications* (2011), p. 2.

[88] Gao, W., and Yang, P. Democracy is Good for Ranking: Towards Multi-View Rank Learning and Adaptation in Web Search. In *Proceedings of the 7th ACM international Conference on Web Search and Data Mining (WSDM)* (2014), pp. 63–72.

[89] Geng, B., Yang, Y., Xu, C., and Hua, X. S. Ranking Model Adaptation for Domain-Specific Search. 745–758.

[90] Geng, X., Liu, T. Y., Qin, T., Arnold, A., Li, H., and Shum, H.-Y. Query Qependent Ranking using k-Nearest Neighbor. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), pp. 115–122.

[91] Geng, X., Liu, T. Y., Qin, T., and Li, H. Feature Selection for Ranking. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 407–414.

[92] Geng, X., Qin, T., Liu, T. Y., Cheng, X. Q., and Li, H. Selecting Optimal Training Data for Learning to Rank. *Information Processing & Management (IPM) 47*, 5 (2011), 730–741.

[93] Geurts, P., Ernst, D., and Wehenkel, L. Extremely Randomized Trees. *Machine Learning 63*, 1 (2006), 3–42.

[94] Geurts, P., and Louppe, G. Learning to Rank with Extremely Randomized Trees. In *JMLR: Workshop and Conference Proceedings* (2011), vol. 14.

[95] Gomes, G., Oliveira, V. C., Almeida, J. M., and Gonçalves, M. A. Is Learning to Rank Worth it? A Statistical Analysis of Learning to Rank Methods in the LETOR Benchmarks. *Journal of Information and Data Management (JIDM) 4*, 1 (2013), 57.

[96] Gopal, S., and Yang, Y. Distributed Training of Large-scale Logistic Models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)* (2013), pp. 289–297.

[97] Guiver, J., and Snelson, E. Learning to Rank with SoftRank and Gaussian Processes. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), pp. 259–266.

[98] He, Q., Ma, J., and Niub, X. Learning to Rank for Information Retrieval Using the Clonal Selection Algorithm. *Journal of Information and Computational Science 7*, 1 (2010), 153–159.

[99] He, Q., Ma, J., and Wang, S. Directly Optimizing Evaluation Measures in Learning to Rank based on the Clonal Selection Algorithm. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM)* (2010), pp. 1449–1452.

[100] Herbrich, R., Graepel, T., and Obermayer, K. Large Margin Rank Boundaries for Ordinal Regression. pp. 115–132.

[101] Herbrich, R., Graepel, T., and Obermayer, K. Support vector learning for ordinal regression. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)* (1999), vol. 1, pp. 97–102 vol.1.

[102] Hoi, S. C. H., and Jin, R. Semi-supervised Ensemble Ranking. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2* (2008), pp. 634–639.

[103] HOLLAND, J. H. *Hidden Order: How Adaptation Builds Complexity*. Ulam Lecture Series. Perseus Books, 1995.

[104] HOPKINS, M., AND MAY, J. Tuning as Ranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2011), pp. 1352–1362.

[105] HUANG, J. C., AND FREY, B. J. Structured Ranking Learning using Cumulative Distribution Networks. In *Advances in Neural Information Processing Systems (NIPS)* (2008), pp. 697–704.

[106] HUBER, P. J. *Robust Statistics*. John Wiley and Sons, New York, 1981.

[107] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *Operating Systems Review (OSR) 41*, 3 (2007), 59–72.

[108] ISLAM, M., HUANG, A. K., BATTISHA, M., CHIANG, M., SRINIVASAN, S., PETERS, C., NEUMANN, A., AND ABDELNUR, A. Oozie: Towards a Scalable Workflow Management System for Hadoop. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies* (2012), p. 4.

[109] JÄRVELIN, K., AND KEKÄLÄINEN, J. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems (TOIS) 20*, 4 (2002), 422–446.

[110] JOACHIMS, T. Optimizing Search Engines using Clickthrough Data. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2002), pp. 133–142.

[111] KAO, C. Y., AND FAHN, C. S. A Multi-Stage Learning Framework for Intelligent System. *Expert Systems with Applications (ESWA) 40*, 9 (2013), 3378–3388.

[112] KARIMZADEHGAN, M., LI, W., ZHANG, R., AND MAO, J. A Stochastic Learning-to-Rank Algorithm and its Application to Contextual Advertising. In *Proceedings of the 20th International conference on World Wide Web (WWW)* (2011), pp. 377–386.

[113] KAYALA, M. A., AZENCOTT, C.-A., CHEN, J. H., AND BALDI, P. Learning to Predict Chemical Reactions. *Journal of Chemical Information and Modeling (JCIM) 51*, 9 (2011), 2209–2222.

[114] KERSTING, K., AND XU, Z. Learning Preferences with Hidden Common Cause Relations. In *Proceedings of the European Conference on Machine Learning (ECML)and Knowledge Discovery in Databases (KDD): Part I* (2009), pp. 676–691.

[115] KOCSIS, L., GYÖRGY, A., AND BÁN, A. N. BoostingTree: Parallel Selection of Weak Learners in Boosting, with Application to Ranking. *Machine learning 93*, 2-3 (2013), 293–320.

[116] Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)* (1995), pp. 1137–1143.

[117] Koza, J. R. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT press, 1992.

[118] Kuo, J. W., Cheng, P. J., and Wang, H. M. Learning to Rank from Bayesian Decision Inference. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 827–836.

[119] Lai, H., Pan, Y., Liu, C., Lin, L., and Wu, J. Sparse Learning-to-Rank via an Efficient Primal-Dual Algorithm. *IEEE Transactions on Computers (TC) 62*, 6 (2013), 1221–1233.

[120] Lai, H., Pan, Y., Tang, Y., and Liu, N. Efficient Gradient Descent Algorithm for Sparse Models with Application in Learning-to-Rank. *Knowledge-Based Systems (KBS) 49* (2013), 190–198.

[121] Lai, H., Tang, Y., Luo, H. X., and Pan, Y. Greedy Feature Selection for Ranking. In *Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2011), pp. 42–46.

[122] Lai, H. J., Pan, Y., Tang, Y., and Yu, R. FSMRank: Feature Selection Algorithm for Learning to Rank. *IEEE transactions on Neural Networks and Learning Systems (T-NNLS) 24*, 6 (2013), 940–952.

[123] Laporte, L., Flamary, R., Canu, S., Dejean, S., and Mothe, J. Nonconvex Regularizations for Feature Selection in Ranking With Sparse SVM. 1.

[124] Le, Q., and Smola, A. Direct Optimization of Ranking Measures. Tech. rep., NICTA, 2007.

[125] Lee, C. P., and Lin, C. J. Large-scale Linear RankSVM. *Neural Computation* (2014), 1–37.

[126] Li, D., Wang, Y., Ni, W., Huang, Y., and Xie, M. An Ensemble Approach to Learning to Rank. In *Proceedings of the Fifth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* (2008), vol. 2, pp. 101–105.

[127] Li, P., Burges, C. J. C., Wu, Q., Platt, J. C., Koller, D., Singer, Y., and Roweis, S. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems (NIPS)* (2007), vol. 7, pp. 845–852.

[128] Lin, H. Y., Yu, C. H., and Chen, H. H. Query-Dependent Rank Aggregation with Local Models. In *Proceedings of the Seventh Asia Information Retrieval Societies Conference (AIRS)* (2011), vol. 7097, pp. 1–12.

[129] Lin, J. Mapreduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *Big Data 1*, 1 (2013), 28–37.

[130] Lin, J. Y., Yeh, J. Y., and Liu, C. C. Learning to Rank for Information Retrieval using Layered Multi-Population Genetic Programming. In *Proceedings of the 2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom)* (2012), pp. 45–49.

[131] Lin, Y., Lin, H., Wu, J., and Xu, K. Learning to Rank with Cross Entropy. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)* (2011), pp. 2057–2060.

[132] Lin, Y., Lin, H., Yang, Z., and Su, S. A Boosting Approach for Learning to Rank Using SVD with Partially Labeled Data. In *Proceedings of the Fifth Asia Information Retrieval Symposium (AIRS)* (2009), pp. 330–338.

[133] Lin, Y., Lin, H., Ye, Z., Jin, S., and Sun, X. Learning to Rank with Groups. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM)* (2010), pp. 1589–1592.

[134] Liu, D. C., and Nocedal, J. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming 45*, 1-3 (1989), 503–528.

[135] Liu, T. Y. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval 3*, 3 (2007), 225–331.

[136] Liu, T. Y., Xu, J., Qin, T., Xiong, W., and Li, H. LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *Proceedings of the SIGIR 2007 Workshop on Learning to Rank for Information Retrieval* (2007), pp. 3–10.

[137] Long, B., Chang, Y., Dong, A., and He, J. Pairwise cross-domain factor model for heterogeneous transfer ranking. In *Proceedings of the fifth ACM International Conference on Web Search and Data Mining (WSDM)* (2012), pp. 113–122.

[138] Luce, R. D. Individual Choice Behavior, a Theoretical Analysis. *Bulletin of the American Mathematics Society 66* (1959), 2–9904.

[139] Luhn, H. P. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development 1*, 4 (1957), 309–317.

[140] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. Pregel: a System for Large-scale Graph Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)* (2010), pp. 135–146.

[141] McNee, S. M., Riedl, J., and Konstan, J. A. Being Accurate is not Enough: How Accuracy Metrics have Hurt Recommender Systems. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (2006), pp. 1097–1101.

[142] Metzler, D., and Croft, W. B. Linear Feature-based Models for Information Retrieval. *Information Retrieval 10*, 3 (2007), 257–274.

[143] Miao, Z., and Tang, K. Semi-supervised Ranking via List-Wise Approach. In *Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)* (2013), pp. 376–383.

[144] Mohan, A., Chen, Z., and Weinberger, K. Q. Web-Search Ranking with Initialized Gradient Boosted Regression Trees. *Journal of Machine Learning Research (JMLR) 14* (2011), 77–89.

[145] Moon, T., Smola, A., Chang, Y., and Zheng, Z. IntervalRank: Isotonic Regression with Listwise and Pairwise Constraints. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM)* (2010), pp. 151–160.

[146] Musilek, P., Lau, A., Reformat, M., and Wyard-Scott, L. Immune programming. *Information Sciences 176*, 8 (2006), 972–1002.

[147] Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*, vol. 87. Springer, 2004.

[148] Ng, S. C., Leung, S. H., and Luk, A. Fast Convergent Generalized Back-Propagation Algorithm with Constant Learning Rate. *Neural Processing Letters 9*, 1 (1999), 13–23.

[149] Ni, W., Huang, Y., and Xie, M. A Query Dependent Approach to Learning to Rank for Information Retrieval. In *Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management (WAIM)* (2008), pp. 262–269.

[150] Niu, S., Guo, J., Lan, Y., and Cheng, X. Top-k Learning to Rank: Labeling, Ranking and Evaluation. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), pp. 751–760.

[151] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig Latin: a Not-So-Foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (2008), pp. 1099–1110.

[152] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web.

[153] Pahikkala, T., Airola, A., Naula, P., and Salakoski, T. Greedy RankRLS: a Linear Time Algorithm for Learning Sparse Ranking Models. In *Proceedings of the SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval* (2010), pp. 11–18.

[154] Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., and Boberg, J. An Efficient Algorithm for Learning to Rank from Preference Graphs. *Machine Learning 75*, 1 (2009), 129–165.

[155] Pan, Y., Lai, H., Liu, C., Tang, Y., and Yan, S. Rank Aggregation via Low-Rank and Structured-Sparse Decomposition. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)* (2013).

[156] PAN, Y., LUO, H.-X., TANG, Y., AND HUANG, C.-Q. Learning to Rank with Document Ranks and Scores. *Knowledge-Based Systems (KBS) 24*, 4 (2011), 478–483.

[157] PAPINI, T., AND DILIGENTI, M. Learning-to-rank with Prior Knowledge as Global Constraints. In *Proceedings of the First International Workshop on Combining Constraint Solving with Mining and Learning (CoCoMiLe)* (2012).

[158] PASS, G., CHOWDHURY, A., AND TORGESON, C. A picture of search. In *Proceedings of the First International Conference on Scalable Information Systems (InfoScale)* (2006), vol. 152.

[159] PAVLOV, D. Y., GORODILOV, A., AND BRUNK, C. A. BagBoo: A Scalable Hybrid Bagging-the-Boosting Model. In *Proceedings of the 19th ACM International conference on Information and Knowledge Management (CIKM)* (2010), pp. 1897–1900.

[160] PENG, J., MACDONALD, C., AND OUNIS, I. Learning to Select a Ranking Function. In *Proceedings of the 32nd European Conference on Information Retrieval Research (ECIR)* (2010), pp. 114–126.

[161] PENG, Z., TANG, Y., LIN, L., AND PAN, Y. Learning to rank with a Weight Matrix. In *Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2010), pp. 18–21.

[162] PETTERSON, J., YU, J., MCAULEY, J. J., AND CAETANO, T. S. Exponential Family Graph Matching and Ranking. In *Advances in Neural Information Processing Systems (NIPS)* (2009), pp. 1455–1463.

[163] PIKE, R., DORWARD, S., GRIESEMER, R., AND QUINLAN, S. Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming 13*, 4 (2005), 277–298.

[164] PLACKETT, R. L. The Analysis of Permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 24*, 2 (1975), 193–202.

[165] POTTER, M. A., AND DE JONG, K. A. Cooperative Coevolution: an Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation 8*, 1 (2000), 1–29.

[166] PYYSALO, S., GINTER, F., HEIMONEN, J., BJÖRNE, J., BOBERG, J., JÄRVINEN, J., AND SALAKOSKI, T. BioInfer: A Corpus for Information Extraction in the Biomedical Domain. *BMC Bioinformatics 8*, 1 (2007), 50.

[167] QIN, T., GENG, X., AND LIU, T. Y. A New Probabilistic Model for Rank Aggregation. In *Advances in Neural Information Processing Systems (NIPS)* (2010), vol. 10, pp. 1948–1956.

[168] QIN, T., LIU, T., XU, J., AND LI, H. LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Information Retrieval 13*, 4 (2010), 346–374.

[169] Qin, T., and Liu, T.-Y. Introducing LETOR 4.0 Datasets. *arXiv preprint arXiv:1306.2597* (2013).

[170] Qin, T., Liu, T. Y., and Li, H. A General Approximation Framework for Direct Optimization of Information Retrieval Measures. *Information Retrieval 13*, 4 (2010), 375–397.

[171] Qin, T., Liu, T. Y., Zhang, X. D., Wang, D. S., and Li, H. Global Ranking using Continuous Conditional Random Fields. In *Advances in neural information processing systems (NIPS)* (2008), pp. 1281–1288.

[172] Qin, T., Liu, T. Y., Zhang, X. D., Wang, D. S., Xiong, W. Y., and Li, H. Learning to Rank Relational Objects and Its Application to Web Search. In *Proceedings of the 17th International Conference on World Wide Web (WWW)* (2008), pp. 407–416.

[173] Qin, T., Zhang, X. D., Tsai, M. F., Wang, D. S., Liu, T. Y., and Li, H. Query-level Loss Functions for Information Retrieval. *Information Processing & Management (IPM) 44*, 2 (2008), 838–855.

[174] Qin, T., Zhang, X. D., Wang, D. S., Liu, T. Y., Lai, W., and Li, H. Ranking with Multiple Hyperplanes. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 279–286.

[175] Ravikumar, P. D., Tewari, A., and Yang, E. On NDCG Consistency of Listwise Ranking Methods. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTAT)* (2011), pp. 618–626.

[176] Renjifo, C., and Carmen, C. The Discounted Cumulative Margin Penalty: Rank-learning with a List-wise Loss and Pair-wise Margins. In *Proceedings of the 2012 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)* (2012), pp. 1–6.

[177] Rifkin, R. M., and Lippert, R. A. Value Regularization and Fenchel Duality. *Journal of Machine Learning Research (JMLR) 8* (2007), 441–479.

[178] Rigutini, L., Papini, T., Maggini, M., and Scarselli, F. Sortnet: Learning to Rank by a Neural-based Sorting Algorithm. In *Proceedings of the SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR)* (2008), pp. 76–79.

[179] Robertson, S., and Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval 3*, 4 (Apr. 2009), 333–389.

[180] Robertson, S. E., and Walker, S. Some Simple Effective Approximations to the 2-poisson Model for Probabilistic Weighted Retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1994), pp. 232–241.

[181] Rudin, C. The P-Norm Push: A Simple Convex Ranking Algorithm that Concentrates at the Top of the List. *Journal of Machine Learning Research (JMLR) 10* (2009), 2233–2271.

[182] Sato, H., Bollegala, D., Hasegawa, Y., and Iba, H. Learning Non-linear Ranking Functions for Web Search using Probabilistic Model Building GP. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)* (2013), pp. 3371–3378.

[183] Schapire, R. E. The Strength of Weak Learnability. *Machine learning 5*, 2 (1990), 197–227.

[184] Sculley, D. Large Scale Learning to Rank. In *Proceedings of the NIPS Workshop on Advances in Ranking* (2009), pp. 1–6.

[185] Shalev-Shwartz, S., and Singer, Y. On The Equivalence of Weak Learnability and Linear Separability: New Relaxations and Efficient Boosting Algorithms. *Machine Learning 80*, 2-3 (2010), 141–163.

[186] Shewchuk, J. R. An Introduction to the Conjugate Gradient Method without the Agonizing Pain. Tech. rep., Carnegie Mellon University, 1994.

[187] Shivaswamy, P. K., and Joachims, T. Online Learning with Preference Feedback. *arXiv preprint arXiv:1111.0712* (2011).

[188] Shukla, S., Lease, M., and Tewari, A. Parallelizing ListNet Training Using Spark. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), pp. 1127–1128.

[189] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop Distributed File System. In *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)* (2010), pp. 1–10.

[190] Silva, T. P. C., de Moura, E. S., Cavalcanti, J. a. M. B., da Silva, A. S., de Carvalho, M. G., and Gonçalves, M. A. An Evolutionary Approach for Combining Different Sources of Evidence in Search Engines. *Information Systems 34*, 2 (2009), 276–289.

[191] Song, Y., Leung, K., Fang, Q., and Ng, W. FP-Rank: An Effective Ranking Approach Based on Frequent Pattern Analysis. In *Database Systems for Advanced Applications (DASFAA)* (2013), pp. 354–369.

[192] Song, Y., Ng, W., Leung, K. W. T., and Fang, Q. SFP-Rank: Significant Frequent Pattern Analysis for Effective Ranking. *Knowledge and Information Systems (KAIS)* (2014), 1–25.

[193] Sorokina, D., Caruana, R., and Riedewald, M. Additive Groves of Regression Trees. In *Proceedings of the 18th European Conference on Machine Learning (ECML)* (2007), pp. 323–334.

[194] SOUSA, D. X., ROSA, T. C., MARTINS, W. S., SILVA, R., AND GONÇALVES, M. A. Improving On-Demand Learning to Rank through Parallelism. In *Proceedings of the 13th International Conference on Web Information Systems Engineering (WISE)* (2012), vol. 7651, pp. 526–537.

[195] STEWART, A., AND DIAZ, E. Epidemic Intelligence: For the Crowd, by the Crowd. In *Proceedings of the 12th International Conference on Web Engineering (ICWE)* (Berlin, Heidelberg, 2012), pp. 504–505.

[196] SUN, H., HUANG, J., AND FENG, B. QoRank: A Query-Dependent Ranking Model using LSE-based Weighted Multiple Hyperplanes Aggregation for Information Retrieval. *International Journal of Intelligent Systems 26*, 1 (2011), 73–97.

[197] SUN, X. H., AND GUSTAFSON, J. L. Toward a Better Parallel Performance Metric. *Parallel Computing 17*, 10 (1991), 1093–1109.

[198] SUN, Z., QIN, T., TAO, Q., AND WANG, J. Robust Sparse Rank Learning for Non-Smooth Ranking Measures. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2009), pp. 259–266.

[199] SVORE, K. M., AND BURGES, C. J. Large-scale Learning to Rank using Boosted Decision Trees. *Scaling Up Machine Learning: Parallel and Distributed Approaches* (2012).

[200] SVORE, K. M., AND BURGES, C. J. C. Learning to Rank on a Cluster using Boosted Decision Trees. In *Proceedings of NIPS Learning to Rank on Cores, Clusters and Clouds Workshop* (2010), p. 16.

[201] SWERSKY, K., TARLOW, D., ADAMS, R., ZEMEL, R., AND FREY, B. Probabilistic n-Choose-k Models for Classification and Ranking. In *Advances in Neural Information Processing Systems (NIPS)* (2012), pp. 3059–3067.

[202] TAN, M., XIA, T., GUO, L., AND WANG, S. Direct Optimization of Ranking Measures for Learning to Rank Models. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2013), pp. 856–864.

[203] TAYLOR, M., GUIVER, J., ROBERTSON, S., AND MINKA, T. SoftRank: Optimizing Non-Smooth Rank Metrics. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM)* (2008), pp. 77–86.

[204] THATHACHAR, M. A. L., AND HARITA, B. R. Learning Automata with Changing Number of Actions. *IEEE Transactions on Systems, Man and Cybernetics (SMC) 17*, 6 (1987), 1095–1100.

[205] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. Hive: a Warehousing Solution over a Map-Reduce Framework. *Proceedings of the VLDB Endowment 2*, 2 (2009), 1626–1629.

[206] Thuy, N. T. T., Vien, N. A., Viet, N. H., and Chung, T. C. Probabilistic Ranking Support Vector Machine. In *Proceedings of the Sixth International Symposium on Neural Networks (ISNN)* (2009), vol. 5552, pp. 345–353.

[207] Torkestani, J. A. An Adaptive Learning Automata-based Ranking Function Discovery Algorithm. *Journal of Intelligent Information Systems (JIIS) 39*, 2 (2012), 441–459.

[208] Torkestani, J. A. An Adaptive Learning to Rank Algorithm: Learning Automata Approach. *Decision Support Systems (DSS) 54*, 1 (2012), 574–583.

[209] Tran, T. T., and Pham, D. S. ConeRANK: Ranking as Learning Generalized Inequalities. *arXiv preprint arXiv:1206.4110* (2012).

[210] Tsai, M.-F., Liu, T. Y., Qin, T., Chen, H. H., and Ma, W. Y. FRank: A Ranking Method with Fidelity Loss. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 383–390.

[211] Tyree, S., Weinberger, K. Q., Agrawal, K., and Paykin, J. Parallel Boosted Regression Trees for Web Search Ranking. In *Proceedings of the 20th International Conference on World Wide Web (WWW)* (2011), pp. 387–396.

[212] Usunier, N., Buffoni, D., and Gallinari, P. Ranking with Ordered Weighted Pairwise Classification. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)* (2009), pp. 1057–1064.

[213] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., and Seth, S. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC)* (2013), p. 5.

[214] Veloso, A., Gonçalves, M. A., Meira Jr, W., and Mossri, H. Learning to Rank using Query-level Rules. *Journal of Information and Data Management (JIDM) 1*, 3 (2010), 567.

[215] Veloso, A. A., Almeida, H. M., Gonçalves, M. A., and Meira Jr, W. Learning to Rank at Query-time using Association Rules. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), pp. 267–274.

[216] Volkovs, M. N., Larochelle, H., and Zemel, R. S. Loss-sensitive Training of Probabilistic Conditional Random Fields. *arXiv preprint arXiv:1107.1805* (2011).

[217] Volkovs, M. N., and Zemel, R. S. BoltzRank: Learning to Maximize Expected Ranking Gain. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)* (2009), pp. 1089–1096.

[218] Volkovs, M. N., and Zemel, R. S. A Flexible Generative Model for Preference Aggregation. In *Proceedings of the 21st International Conference on World Wide Web (WWW)* (2012), pp. 479–488.

[219] Volkovs, M. N., and Zemel, R. S. CRF Framework for Supervised Preference Aggregation. In *Proceedings of the 22Nd ACM International Conference on Conference on Information and Knowledge Management (CIKM)* (2013), pp. 89–98.

[220] Wang, B., Tang, J., Fan, W., Chen, S., Yang, Z., and Liu, Y. Heterogeneous Cross Domain Ranking in Latent Space. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 987–996.

[221] Wang, B., Wu, T., Yan, F., Li, R., Xu, N., and Wang, Y. RankBoost Acceleration on both NVIDIA CUDA and ATI Stream platforms. In *Proceedings of the 15th International Conference on Parallel and Distributed Systems (ICPADS)* (2009), pp. 284–291.

[222] Wang, C. J., Huang, H. S., and Chen, H. H. Automatic Construction of an Evaluation Dataset from Wisdom of the Crowds for Information Retrieval Applications. In *Proceedings of the Sixth IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2012), pp. 490–495.

[223] Wang, F., and Xu, X. AdaGP-Rank: Applying Boosting Technique to Genetic Programming for Learning to Rank. In *Proceedings of the IEEE Youth Conference on Information Computing and Telecommunications (YC-ICT)* (2010), pp. 259–262.

[224] Wang, L., Bennett, P. N., and Collins-Thompson, K. Robust Ranking Models via Risk-sensitive Optimization. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), pp. 761–770.

[225] Wang, S., Gao, B. J., Wang, K., and Lauw, H. W. CCRank: Parallel Learning to Rank with Cooperative Coevolution. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)* (2011).

[226] Wang, S., Gao, B. J., Wang, K., and Lauw, H. W. CCRank: Parallel Learning to Rank with Cooperative Coevolution. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI)* (2011).

[227] Wang, S., Gao, B. J., Wang, K., and Lauw, H. W. Parallel Learning to Rank for Information Retrieval. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 1083–1084.

[228] Wang, S., Ma, J., and Liu, J. Learning to Rank using Evolutionary Computation: Immune Programming or Genetic Programming? In *Proceedings of the 18th ACM conference on Information and knowledge management (CIKM)* (2009), pp. 1879–1882.

[229] Wang, Y., Huang, Y., Pang, X., Lu, M., Xie, M., and Liu, J. Supervised Rank Aggregation Based on Query Similarity for Document Retrieval. *Soft Computing 17*, 3 (2013), 421–429.

[230] Wang, Y., Kuai, Y. H., Huang, Y. L., Li, D., and Ni, W. J. Uncertainty-based Active Ranking for Document Retrieval. In *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC)* (July 2008), vol. 5, pp. 2629–2634.

[231] Wolpert, D. H. Stacked Generalization. *Neural Networks 5*, 2 (1992), 241–259.

[232] Wu, J., Yang, Z., Lin, Y., Lin, H., Ye, Z., and Xu, K. Learning to Rank using Query-level Regression. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 1091–1092.

[233] Wu, M., Chang, Y., Zheng, Z., and Zha, H. Smoothing DCG for Learning to Rank: A Novel Approach using Smoothed Hinge Functions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 1923–1926.

[234] Wu, Q., Burges, C. J. C., Svore, K. M., and Gao, J. Ranking, Boosting, and Model Adaptation. Tech. rep., Microsoft Research, 2008.

[235] Xia, F., Liu, T. Y., Wang, J., Zhang, W., and Li, H. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th Annual International Conference on Machine Learning (ICML)* (2008), pp. 1192–1199.

[236] Xu, J., and Li, H. Adarank: A Boosting Algorithm for Information Retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 391–398.

[237] Xu, J., Liu, T. Y., Lu, M., Li, H., and Ma, W. Y. Directly Optimizing Evaluation Measures in Learning to Rank. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Rnformation Retrieval* (2008), pp. 107–114.

[238] Xu, N. Y., Cai, X. F., Gao, R., Zhang, L., and Hsu, F. H. FPGA-based Accelerator Design for RankBoost in Web Search Engines. In *Proceedings of the 177th International Conference on Field-Programmable Technology (ICFPT)* (2007), pp. 33–40.

[239] Xu, N. Y., Cai, X. F., Gao, R., Zhang, L., and Hsu, F. H. FPGA Acceleration of RankBoost in Web Search Engines. *ACM Transactions on Reconfigurable Technology and Systems (TRETS) 1*, 4 (2009), 19.

[240] Xu, Z., Chapelle, O., and Weinberger, K. Q. The Greedy Miser: Learning under Test-time Budgets. In *Proceedings of the 29th International Conference on Machine Learning (ICML)* (2012), pp. 1175–1182.

[241] Xu, Z., Kersting, K., and Joachims, T. Fast Active Exploration for Link-Based Preference Learning Using Gaussian Processes. In *Proceedings of the European Conference on Machine Learning (ECML) and Principles and Practice of Knowledge Discovery in Databases (PKDD)* (2010), vol. 6323, pp. 499–514.

[242] Yan, J., Xu, N. Y., Cai, X. F., Gao, R., Wang, Y., Luo, R., and Hsu, F.-H. FPGA-based acceleration of neural network for ranking in web search engine with a streaming architecture. In *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL)* (2009), pp. 662–665.

[243] Yan, J., Xu, N.-Y., Cai, X.-F., Gao, R., Wang, Y., Luo, R., and Hsu, F.-H. LambdaRank Acceleration for Relevance Ranking in Web Search Engines. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)* (2010), p. 285.

[244] Yan, J., Xu, N. Y., Cai, X.-F., Gao, R., Wang, Y., Luo, R., and Hsu, F. H. An FPGA-based Accelerator for LambdaRank in Web Search Engines. *ACM Transactions on Reconfigurable Technology and Systems (TRETS) 4*, 3 (2011), 25.

[245] Yan, J., Zhao, Z. X., Xu, N. Y., Jin, X., Zhang, L. T., and Hsu, F. H. Efficient Query Processing for Web Search Engine with FPGAs. In *Proceedings of the 20th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2012), pp. 97–100.

[246] Ye, J., Chow, J. H., Chen, J., and Zheng, Z. Stochastic Gradient Boosted Distributed Decision Trees. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 2061–2064.

[247] Yeh, J. Y., Lin, J. Y., Ke, H. R., and Yang, W. P. Learning to Rank for Information Retrieval using Genetic Programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR)* (2007).

[248] Yu, C.-N. J., and Joachims, T. Learning Structural SVMs with Latent Variables. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)* (New York, NY, USA, 2009), pp. 1169–1176.

[249] Yue, Y., Finley, T., Radlinski, F., and Joachims, T. A Support Vector Method for Optimizing Average Precision. In *Proceedings of the 30th Annual Unternational ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 271–278.

[250] Yun, H., Raman, P., and Vishwanathan, S. V. N. Ranking via Robust Binary Classification and Parallel Parameter Estimation in Large-Scale Data. *arXiv preprint arXiv:1402.2676* (2014).

[251] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (2010), p. 10.

[252] Zhai, C., and Lafferty, J. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), pp. 334–342.

[253] Zheng, Z., Chen, K., Sun, G., and Zha, H. A Regression Framework for Learning Ranking Functions using Relative Relevance Judgments. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 287–294.

[254] Zhou, D., Ding, Y., You, Q., and Xiao, M. Learning to Rank Documents using Similarity Information Between Objects. In *Proceedings of the 18th International Conference on Neural Information Processing (ICONIP) - Volume Part II* (2011), pp. 374–381.

[255] Zhou, K., Xue, G.-R., Zha, H., and Yu, Y. Learning to Rank with Ties. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), pp. 275–282.

[256] Zhu, C., Chen, W., Zhu, Z. A., Wang, G., Wang, D., and Chen, Z. A General Magnitude-preserving Boosting Algorithm for Search Ranking. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)* (2009), pp. 817–826.

[257] Zhu, M. Recall, Precision and Average Precision. Tech. rep., Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, 2004.

[258] Zinkevich, M., Weimer, M., Smola, A. J., and Li, L. Parallelized Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems (NIPS)* (2010), pp. 2595–2603.

[259] Zong, W., and Huang, G. B. Learning to Rank with Extreme Learning Machine. *Neural Processing Letters* (2013), 1–12.