NIEK TAX

METHODS FOR LARGE SCALE LEARNING-TO-RANK

[1st May 2014 at 14:46 – classicthesis version 0.1]

METHODS FOR LARGE SCALE LEARNING-TO-RANK

A study concerning parallelization of Learning-to-Rank algorithms using Hadoop

NIEK TAX BSC.
Research Chair Databases
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
University of Twente

UNIVERSITY OF TWENTE.

Februari 2014 – version 0.1

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

ABSTRACT

Short summary of the contents in English...

SAMENVATTING

Een korte samenvatting in het Nederlands...

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

IDF Inverse Document Frequency

IP Immune Programming

IR Information Retrieval

KL DIVERGENCE Kullback-Leibler divergence

KNN K-Nearest Neighbours

MAP Mean Average Precision

MHR Multiple Hyperplane Ranker

MLE Maximum Likelihood Estimator

MSE Mean Squared Error

NDCG Normalized Discounted Cumulative Gain

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

URL Uniform Resource Locator

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 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 amongst others: automatic document summarisation, machine translation, drug discovery and in determining the ideal order of maintenance operations [163]. In addition, the ranking task has been argued to be more fitting to recommender systems than regression-based rating prediction on a continuous scale [4, 129].

Research in the field of ranking models has for a long time been based on manually designed ranking functions, such as the well-known BM25 model [162]. Luhn [127] was the first to propose a model that assigned relevance scores to documents given a query back in 1957. 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 datasets have been proposed that enable comparison of the performance of different Learning-to-Rank methods. Well-known benchmark datasets include the *Yahoo! Learning to Rank Challenge* dataset [40], the Yandex Internet Mathematics competition[1], and the LETOR dataset [152] that was published by Microsoft Research. 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, while 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 training sets are becoming larger and larger [41]. Liu [123] 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 [49, 38]. 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 [13, 14, 33, 175, 170], since Liu [123] first stated its growing importance back in 2007. Only several of these studies

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

[175, 170] have explored the possibilities of efficient Learning-to-Rank through the use of parallel programming paradigms.

MapReduce [63] is a parallel programming framework that is inspired by the *Map* and *Reduce* functions commonly used in 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. Lin [118] 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 can still be optimized in such a way that its performance in a MapReduce setting is good enough.

The appearance of benchmark datasets gave insight in the performance of different Learning-to-Rank approaches, which resulted in increasing popularity of those methods that showed to perform well on one or more benchmark datasets. Up to now it remains unknown whether popular existing Learning-to-Rank methods scale well 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. A more extensive overview of my research goals and questions are described in section 1.2.

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 framework. This work focuses on those Learning-to-Rank algorithms that have shown leading performance on relevant benchmark datasets. This thesis addresses the following research questions:

RQ1 What are the best performing Learning-to-Rank algorithms in terms of accuracy on relevant benchmark datasets?

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 [178]:

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

RQ3 Can those Learning-to-Rank algorithms be adjusted in such a way that the parallel execution speed-up increases without decreasing accuracy?

1.3 APPROACH

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 is all definitions, 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.

To answer the first research question, I will implement Learning-to-Rank methods in the MapReduce framework and measure runtime as a factor of the number of cluster nodes used to complete the computation.

I will use the HDInsight cloud-based MapReduce implementation from Microsoft to implement the Learning-to-Rank algorithms. HDInsight is based on the popular open source MapReduce implementation Hadoop[2]. The algorithms that we include in the measurements will be determined based on experimental results on the *Yahoo! Learning to Rank Challenge* [40], the Yandex Internet Mathematics competition[3], the LETOR [152] dataset and the LETOR successors MSLR-WEB10k and MSLR-WEB30k.

1.4 THESIS OVERVIEW

CHAPTER II: BACKGROUND introduces the reader to the basic principles and recent work in the fields of Learning-to-Rank.

CHAPTER III: RELATED WORK concisely describes existing work in the field of parallel Learning-to-Rank.

CHAPTER IV: BENCHMARK RESULTS sketches the accuracy of existing Learning-to-Rank methods on several benchmark datasets and describes the selection of Learning-to-Rank methods for the parallelisation experiments.

2 http://hadoop.apache.org/
3 http://imat-relpred.yandex.ru/en/

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

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

CHAPTER VII: RESULTS & DISCUSSION presents and discusses speed-up results for the implemented Learning-to-Rank methods.

CHAPTER VIII: CONCLUSION 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.

2 TECHNICAL BACKGROUND

This chapter is written with the goal to provide the reader with a subtle introduction in the Learning-to-Rank field. 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. It shows training elements from an input space that are mapped to an output space using a model such that the difference between the actual labels of the training elements and the labels predicted with with the model are minimal in terms of a loss function.

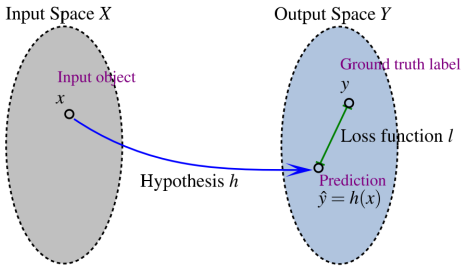

Figure 1: Machine learning framework for Learning-to-Rank, obtained from Liu [123]

Liu [123] 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*, which are itself defined as follows:

FEATURE BASED means that all documents under investigation are represented by feature vectors. Those feature vectors reflect 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.

Figure 2 shows how the machine learning process as described in Figure 1 typically takes place in a ranking scenario. A set of queries q_i with $n > i > 1$, the documents associated with these queries which are represented by feature vector x_i, and the relevant judgements of those documents y_i are used together to train a model h. Model h can after training be used to predict a ranking of the documents y_i, 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 loss function.

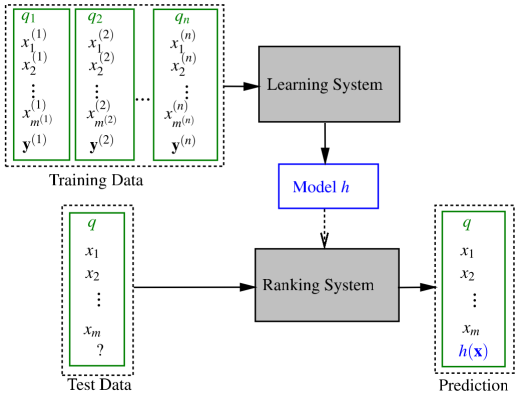

Figure 2: A typical Learning-to-Rank setting, obtained from Liu [123]

The predictions and the loss function might either be defined for:

1. the relevance of a single document
2. the classification of the most relevant document out of a document-pair
3. the ranking of documents directly

These three approaches are in literature respectively called the pointwise approach, the pairwise approach and the listwise approach. These three approaches to Learning-to-Rank will be described in more detail in section 2.3.

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 to express the evaluation results in chapter 4 of this thesis.

2.2.1 *Normalized Discounted Cumulative Gain*

Cumulative gain, or its predecessor discounted cumulative gain and normalized discounted cumulative gain, is one of 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 at position p was originally defined by Järvelin and Kekäläinen [101] as

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

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. [28] proposed an alternative definition of DCG that puts stronger emphasis on document relevance

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

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

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

Table 1 shows an example calculation for nDCG for both the Järvelin and Kekäläinen [101] and Burges et al. [28] version of DCG.

One limitation of nDCG that it does not penalise for missing documents in the result set. Normalized Discounted Cumulative Gain is often used with a fixed

	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

Järvelin and Kekäläinen [101] nDCG = $\frac{29.17}{29.96} = 0.97$
Burges [28] et al. nDCG = $\frac{732.7}{788.0} = 0.93$

Table 1: Example calculation of nDCG

set size for the result set to mitigate the missing document problem. nDCG with a fixed set size is often called nDCG@k, where k represents the set size.

2.2.2 *Expected Reciprocal Rank*

Expected Reciprocal Rank

Expected Reciprocal Rank (ERR) [42] 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 users are likely to stop exploring the result list once they have found a document that satisfied their information need. 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

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

where the product sequence part of the formula represents the chance that the user will stop at position r.
An 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.

```
1 p ← 1, ERR ← 0
2 for r ← 1 to n do
3 |   R ← R(g[r])
4 |   ERR ← ERR + p * R/r
5 |   p ← p * (1 − R)
6 end
7 Output ERR
```

Algorithm 1: Algorithm to compute the ERR metric, obtained from [42]

	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
										/#Relevant docs =	7
										AP@10 =	0.34

Table 2: Average Precision example calculation. The number of relevant documents R is assumed to be seven.

2.2.3 *Mean Average Precision*

Average Precision

Average Precision (AP) [235] 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

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

With k being the positions in the result set between 1 and n. Table 2 provides an example calculation of average precision where de documents at positions 1, 5, 6 and 8 in the ranking are relevant. Mean Average Precision (MAP) is the average AP for a set of queries.

Mean Average Precision

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

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 [116] and PRank [53].

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 [74]. An often used solution to this problem is to upper bound the number of classification mistakes by an easy to optimise function [16].

Well-known pairwise Learning-to-Rank algorithms include FRank [190], GBRank [231], LambdaRank [30], RankBoost [76], RankNet [28], Ranking SVM [95, 102], and SortNet [160].

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 [228, 184, 43]. 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 [215], BoltzRank [196], ListMLE [214], ListNet [35], RankCosine [155], SmoothRank [43], SoftRank [184], SVM^{map} [228].

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 I 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.

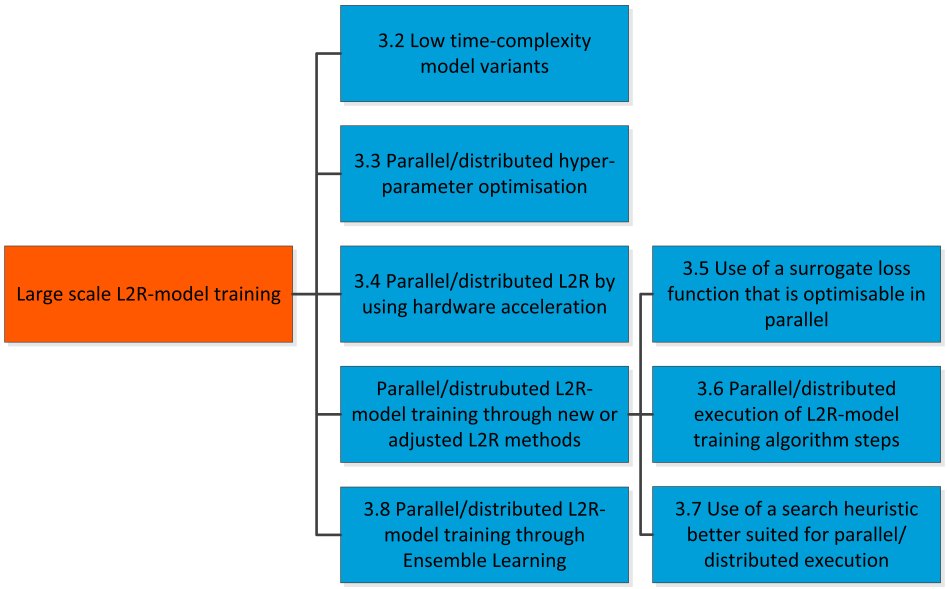

Figure 3: Categories in research on large scale training of Learning-to-Rank models

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. [139] described a pairwise Regularised Least-Squares (RLS) type of ranking function, RankRLS, with low time complexity. Airola et al. [7] 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 [96, 102] on the BioInfer corpus [149], a corpus for information extraction in the biomedical domain.

Support Vector Machine

Gradient Boosted Decision Tree

Airola et al. [8] and Lee and Lin [114] both described lower time complexity methods to train a linear kernel ranking Support Vector Machine (SVM) [96, 102]. Lee and Lin [114] 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. [82, 80] 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 [213] ranking model, which uses the Gradient Boosting [79] 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. [31] described their Yahoo! Learning to Rank Challenge submission that was built by performing an extensive hyperparameter search on a 122-node 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 ERR score of all Yahoo! Learning to Rank Challenge submissions.

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

[44]. 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. [221, 222, 223, 224] 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 [224] 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. [217, 218] designed an FPGA-based accelerator to reduce the training time of the RankBoost algorithm [76], a pairwise ranking function based on Freund and Schapire's AdaBoost ensemble learning method [77]. Xu et al. [218] 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 [217]. 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 [218].

3.4.2 *GPGPU for parallel Learning-to-Rank*

General-Purpose computing on Graphical Processing Units

Computing Unified Device Architecture

Wang et al. [200] 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. [62] 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. [194], 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 dataset. 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. [191] 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 [17], 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) [24] 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. [170] explored the parallelisation of the well-known ListNet Learning-to-Rank method using Spark. Spark 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 nDCG, ERR and 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) [226]. GAs are search heuristic functions that mimic the process of natural selection, consisting of mutation and cross-over steps [98]. The following subsection describe related work that uses search heuristics for parallel/distributed training.

Evolutionary Algorithm

Genetic Algorithm

3.6.1 *Immune Programming*

Immune Programming

Wang et al. [207] proposed a Immune Programming (IP) solution to direct ranking metric optimisation and observed that all EAs are generally easy to implement such that it can be carried out on different computers distributively. 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. [204, 206] proposed a parallel evolutionary algorithm based on Cooperative Coevolution (CC), a type of evolutionary algorithm. 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 dataset 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. [104] 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 dataset and their own dataset 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. [72] 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. [72] showed how to use ADMM to train a RankSVM [96, 102] 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. [23]. They implemented an ADMM based Learning-to-Rank method in Pregel [128], 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 [100] is defined as the task to lean a model in the presence of outliers. Yun et al. described a [229] 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 [91], 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 [114] of a linear kernel RankSVM [96, 102] described in section 3.2.

3.7.4 *Distributed Stochastic Gradient Descent*

Stochastic Gradient Descent

Long et al. [125] 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. [236] 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 [165]. 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 [78], bagging [25], AdaBoost [77] and stacked generalisation (often called stacking) [210].

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 [231] or listwise [46] setting.

3.8.1 *Gradient Boosting*

Gradient Boosting [78] 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. [106] 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. [225] 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 [79]. 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 [181, 180] designed two approaches to train the LambdaMART [213] 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 [25], 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. [144] 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. [81, 80] 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 [213] ranking model, that uses the Gradient Boosting [79] 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 dataset 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
Mean Squared Error

A Deep Stacking Network (DSN) is a processing architecture developed from the field of *Deep Learning*, that uses the Mean Squared Error (MSE) loss function that is easily optimisable. DSN is based on the stacked generalization (stacking) ensemble learning method [210], an approach where several learning models are stacked on top of each other in such a way that the output of a model is used as one of the input features of models higher up in the stack. Each layer in the stack learns has the task of learning the weights of the inputs of that layer.

The close-form constraints between input and output weights allow the input weight matrices to be estimated in a parallel manner. Deng et al. [64]

3.8.4 *Randomisation*

Extremely Randomised Trees

Most ensemble learning based Learning-to-Rank methods are based on boosting. Although boosting have 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 [89] 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 [88]. 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 dataset 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.

4

BENCHMARK RESULTS

Benchmark datasets enable fair comparisons of ranking models over a fixed set of documents and queries. The approach of using fixed benchmark datasets 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 [51].

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

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 nDCG definition is unambiguous, Busa-Fekete et al. [33] found that nDCG evaluation tools of benchmark data sets produced different scores

4.1 YAHOO! LEARNING TO RANK CHALLENGE

Yahoo's observation that all existing benchmark datasets were too small to draw reliable conclusions prompted Yahoo to release two internal datasets from Yahoo! search. Datasets used at commercial search engines are many times larger than available benchmark datasets. 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 [40] is a public Learning-to-Rank competition which took place from March to May 2010, with the goal to promote the datasets 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 datasets 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, I will only look at the standard Learning-to-Rank dataset, because transfer learning is a separate research area that is not included in this thesis.

Both nDCG and ERR are measured as performance metrics, but the final standings of the challenge were based on the ERR values. Model validation on the

	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 3: Yahoo! Learning to Rank Challenge dataset characteristics, as described in the challenge overview paper [40]

Learning-to-Rank methods participating in the challenge is performed using a train/validation/test-set split following the characteristics shown in Table 3. 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 datasets makes the Yahoo! Learning to Rank Challenge dataset 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 [40]:

WEB GRAPH Quality and popularity metrics of web documents, e.g. PageRank [137].

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 [161] 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

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

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 4: Final standings of the Yahoo! Learning to Rank Challenge, as presented in the challenge overview paper [40]

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 4 shows the top five participants in the Yahoo! Learning to Rank Challenge in terms of ERR score. Burges [31] created a linear combination ensemble of eight LambdaMART [29], 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 [144] ensemble technique, which combines bagging and boosting. Sorokina used a similar combination of bagging and boosting that is called Additive Groves [174].

Gradient Boosted Decision Tree

The challenge overview paper [40] 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 [144] method used by Pavlov & Brunk. The bagging component of the BagBoo method enables it to achieve high scalability through parallelism. Pavlov et al. [144] 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 [124] 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 in December 2008 and LETOR 4.0 in July 2009.

4.2.1 *LETOR 2.0*

LETOR 2.0 consists of three datasets: the OHSUMED dataset and two datasets 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 [54]:

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 [96, 102]
- RankBoost [76]
- Multiple Hyperplane Ranker (MHR) [156]
- FRank [190]

Multiple Hyperplane Ranker

LISTWISE

- ListNet [35]
- AdaRank-MAP [215]
- AdaRank-nDCG [215]

4.2.1.1 *Results*

Table 5 shows the performance of the baseline methods on the datasets 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 datasets and ranked third on the third dataset. MHR was only evaluated on the OHSUMED dataset, 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 5: nDCG@10 results of baseline methods on LETOR 2.0

4.2.2 *LETOR 3.0*

The LETOR 3.0 benchmark collection [152] as released in 2007 contained two data sets: the OHSUMED collection and the .gov collection. Tables 15 and 14 in Appendix A provide the descriptions of the features of the .gov and the OHSUMED 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 datasets 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 MAP. The *winning number* metric is defined as the number of other algorithms that it can beat over all of the seven datasets (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:

POINTWISE

- Linear regression

PAIRWISE

- RankSVM [96, 102]
- RankBoost [76]
- FRank [190]

LISTWISE

- ListNet [35]
- AdaRank-MAP [215]
- AdaRank-nDCG [215]
- SVM^{MAP} [228]

4.2.2.1 *Results*

The LETOR paper [152] 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.

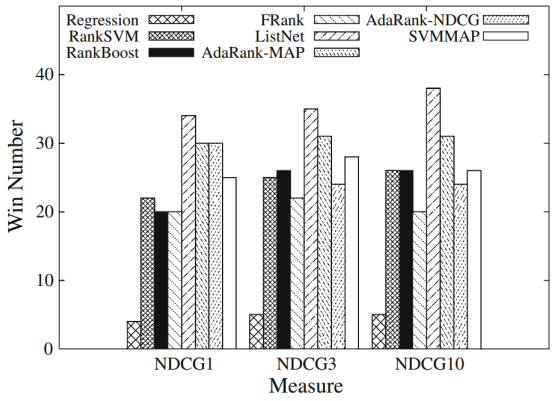

Figure 4: Comparison across the seven datasets in LETOR by nDCG, obtained from Qin et al. [152]

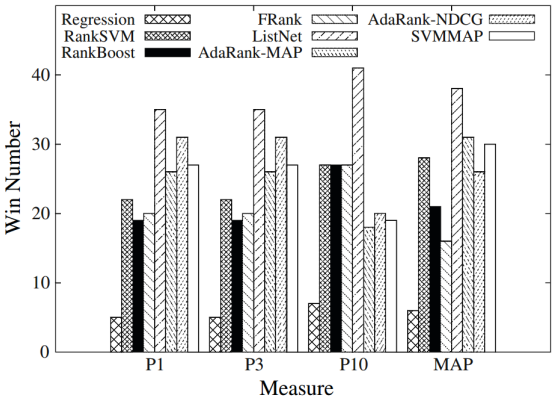

Figure 5: Comparison across the seven datasets in LETOR by MAP, obtained from Qin et al. [152]

I 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. [152], 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 datasets in nDCG value. Note that this does not necessarily imply that these methods would have scored a higher nDCG winning number than ListNet. Table 6 shows the ListNet performance on the seven LETOR datasets in terms of nDCG@10 and MAP.

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

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 6: Performance of ListNet on LETOR 3.0

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 datasets	2	2	1	3	-

Table 7: 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 datasets 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 8 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 data into folds was performed beforehand by the creators of the MQ2007 and MQ2008 datasets. Documents in the dataset were

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

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

Table 8: Characteristics of the LETOR 4.0 collection

Model	Dataset	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 9: Comparison of LETOR 4.0 baseline models

4.2.3.1 *Results*

RankSVM-Struct, ListNet, AdaRank-nDCG, AdaRank-MAP and RankBoost were used as baseline models on the LETOR 4.0 dataset and were implemented and evaluated by the publishers of LETOR 4.0. Table 9 shows the performance of those baseline models on the LETOR 4.0 benchmark collection.

BoostedTree model [106] showed an nDCG of 0.5071 on the MQ2007 dataset and thereby beat all baseline models.

4.3 OTHER DATASETS

LETOR and the Yahoo! Learning to Rank Challenge datasets 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 datasets 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 dataset only includes features that are commonly used in the research community, as proprietary features were excluded from the

4 http://www.bing.com/

dataset. Microsoft published these datasets 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 dataset 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 [9], contains of two datasets from the Chilean search engine TodoCL[6]. Both datasets 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 [96, 102] and RankBoost [76] algorithms. In addition two ranking methods developed at the same university of the WCL2R paper were evaluated: LAC [194] and a ranking algorithm based on Genetic Programming (GP) [61]. Table 10 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 10: nDCG results of baseline methods on the WCL2R collection, obtained from [9]

4.3.3 *AOL*

Pass et al. [143] describe an America Online Learning-to-Rank dataset which they published in 2006. This dataset is unique in that it is the only large commercial web search dataset that contains user session and user behaviour information. The dataset contains 20 million search keywords for over 650,000 users over a three month time period. AOL later realised the publication of the dataset to be a mistake, as personally identifiable information turned out 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

5 http://research.microsoft.com/en-us/projects/mslr/download.aspx
6 www.todocl.cl
7 http://select.nytimes.com/gst/abstract.html?res=F10612FC345B0C7A8CDDA10894DE404482

sites[8]. This controversial AOL dataset 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. [144] in their paper claim to have won the Yandex Internet Mathematics contest 2009 with their BagBoo method.

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 datasets. 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 datasets. Gomes et al. [90] analysed ranking accuracy of a set of models on both LETOR 3.0 and LETOR 4.0. Busa-Fekete et al. [32] compared the accuracy of a small set of models over the LETOR 4.0 datasets, both MSLR datasets, both Yahoo! Learning to Rank Challenge datasets and the OHSUMED dataset from LETOR 3.0. To my 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 datasets described in foregoing sections.

5.1 COLLECTING EVALUATION RESULTS

With a literature review we will collect evaluation results on the datasets/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 11 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. [124]	307
LETOR 3.0	Qin et al. [152]	105
Yahoo! Learning to Rank Challenge	Chapelle et al. [40]	102
AOL dataset	Pass et al. [143]	339
WCL2R	Alcântara et al. [9]	2

Table 11: 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 12: Google scholar search results for Learning-to-Rank benchmarks

5.2 COMPARISON METHODOLOGY

The LETOR 3.0 paper [152] states that it may differ between datasets what the most accurate ranking methods are. To evaluate the overall performance of Learning-to-Rank methods over the multiple datasets in the LETOR 3.0 collections, Qin et al. [152] proposed a measure called *winning number* as the number of other algorithms that an algorithm can beat over the set of datasets. 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 datasets 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 $I_{\{M_i(j)>M_k(j)\}}$ is an indicator function such that

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

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 takes only those datasets 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 datasets on which it has been evaluated. We will redefine the indicator function I in order to only take into account those datasets that an algorithm is evaluated on, as

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

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

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

where Ideal Winning Number is defined as

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

where j is the index of a dataset, n the number of datasets 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 datasets. An exception is the Yahoo! Learning to Rank Challenge datasets on which mainly 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 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 13 gives an overview of the Learning-to-Rank methods for which evaluation results were found for one or more of the benchmark datasets listed in section 5.1 through the literature review process also described in section 5.1. Occurrences of L2, L3 and L4 in Table 13 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 enumerates 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 [87, 119]
2. The study focussed on a different Learning-to-Rank task (e.g. rank aggregation or transfer ranking) [59, 58, 65, 60, 45, 6, 199, 56, 131, 97, 57, 71, 11, 150, 198, 66, 140, 117, 197, 55]
3. The study used an altered version of a benchmark that contained additional features [21, 68]
4. The study provides no exact data of the evaluation results (e.g. results are only in graphical form) [209, 202, 220, 107, 115, 214, 232, 211, 234, 104, 182, 141, 135, 50, 176, 147, 5, 39, 154, 3, 166, 99, 10, 177, 93, 18, 84, 47, 219, 169]
5. The study reported evaluation results in a different metric than the metrics chosen for this meta-analysis [227, 186, 139, 105, 132]
6. The study reported a higher performance on baseline methods than official benchmark runs [69, 15, 145, 173, 20, 19, 36, 2, 145, 189, 12]
7. The study did not report any baseline performance that allowed us to check validity of the results [37, 203, 27].

5.4 RESULTS & DISCUSSION

The following subsections provide the performance of Learning-to-Rank methods in terms of Normalised Winning Number for nDCG@3, nDCG@5, nDCG@10 and MAP. Performance of the Learning-to-Rank methods is plotted with Normalised Winning Number on the vertical axis and the number of datasets on which the method has been evaluated on the horizontal 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

Method	Described	Evaluated	Method	Described	Evaluated
AdaRank-MAP	[215]	L2, L3, L4	Linear Regression	[52]	L3, [201, 195]
AdaRank-nDCG	[215]	L2, L3, L4, [32, 183]	ListMLE	[214]	[122, 120, 83]
ADMM	[72]	[72]	ListNet	[35]	L2, L3, L4
ApproxAP	[151]	[151]	ListReg	[211]	[211]
ApproxnDCG	[151]	[151]	LRUF	[188]	[188]
BagBoo	[144]	[81]	MCP	[112]	[112]
Best Single Feature		[90]	MHR	[156]	L2
BL-MART	[81]	[81]	MultiStageBoost	[103]	[103]
BoltzRank-Single	[196]	[196, 198]	NewLoss	[146]	[146]
BoltzRank-Pair	[196]	[196, 81, 198]	OWPC	[192]	[192]
BT	[233]	[233]	PERF-MAP	[141]	[188]
C-CRF	[153]	[153]	PermuRank	[216]	[216]
CA	[130]	[32, 183]	Q.D.KNN	[85]	[208]
CCRank	[205]	[205]	RandomForest		[90]
CoList	[83]	[83]	Rank-PMBGP	[164]	[164]
Consistent-RankCosine	[157]	[183]	RankAggnDCG	[208]	[208]
DCMP	[158]	[158]	RankBoost	[76]	L2, L3, L4, [32, 9]
DirectRank	[183]	[183]	RankCSA	[94]	[94]
EnergynDCG	[75]	[75]	RankDE	[22]	[164]
FBPCRank	[110]	[110]	RankELM (pairwise)	[237]	[237]
FenchelRank	[108]	[108, 109, 112]	RankELM (pointwise)	[237]	[237]
FocusedBoost	[136]	[136]	RankMGP	[119]	[119]
FocusedNet	[136]	[136]	RankNet	[28]	[32, 142, 136]
FocusedSVM	[136]	[136]	RankRLS	[139]	[138]
FP-Rank	[172]	[172]	RankSVM	[96, 102]	L2, L3, [32, 75, 94, 9]
FRank	[190]	L2, L3, [201]	RankSVM-Primal		L3, [110]
FSMRank	[111]	[111, 112]	RankSVM-Struct		L3, L4
FSM^{SVM}	[111]	[111]	RCP	[73]	[73]
GAS-E	[86]	[111]	RE-QR	[193]	[193]
GP	[61]	[9]	REG-SHF-SDCG	[212]	[212]
GPRank	[171]	[187]	Ridge Regression	[52]	L3
GRankRLS	[138]	[138]	RSRank	[179]	[108]
GroupCE	[120]	[120]	SmoothGrad	[113]	[183]
GroupMLE	[122]	[120]	SmoothRank	[43]	L3, [43]
IntervalRank	[133]	[133, 75]	SoftRank	[184, 92]	[151]
IPRank	[207]	[207, 187]	SortNet	[160]	[160, 75]
KeepRank	[48]	[48]	SparseRank	[109]	[109]
Kernel-PCA RankBoost	[70]	[70, 164]	SVD-RankBoost	[121]	[121]
KL-CRF	[195]	[195]	SVM^{MAP}	[228]	L3, [201, 216, 136]
LAC-MR-OR	[194]	[194]	SwarmRank	[67]	[164]
LambdaMART	[29]	[13, 81]	TGRank	[108]	[108]
LambdaNeuralRank	[142]	[142]	TM	[233]	[233, 142, 183]
LambdaRank	[30]		VFLR	[34]	[34]
LARF	[187]	[187]			

Table 13: Learning-to-Rank algorithms with measurements on benchmark datasets

that has 1) a higher Normalised Winning Number and 2) a higher number datasets 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 16 in Appendix B provides the raw Normalised Winning Number data for the Learning-to-Rank methods for which nDCG@3 evaluation results were available.

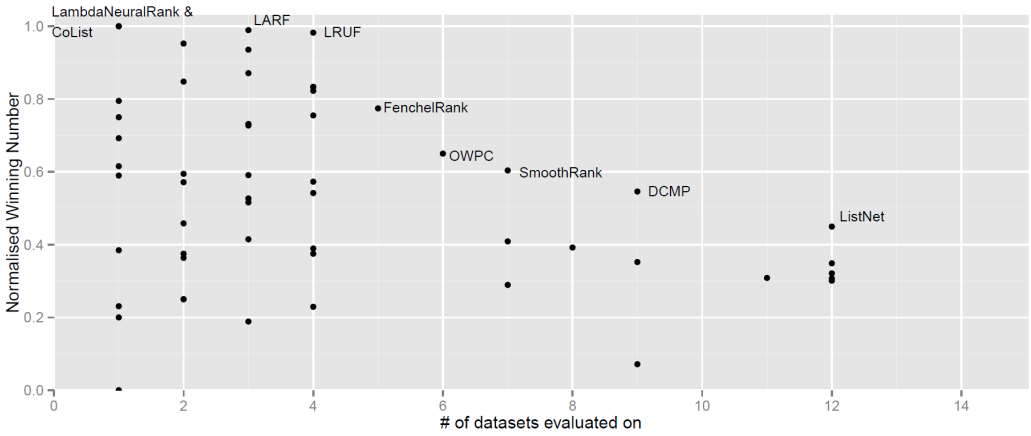

Figure 6: nDCG@3 comparison of Learning-to-Rank methods

LambdaNeuralRank and CoList both acquired a perfect Normalised Winning Number 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 datasets, which can be said to have a higher degree of certainty on the methods' performance because they are validated on three datasets which in addition are more relevant datasets than AOL and Yahoo Set 2 because there are more evaluation results available for the LETOR 3.0 datasets (see Table 13). FenchelRank, OWPC, SmoothRank, DCMP and ListNet are in that order increasingly lower in Normalised Winning Number, but increasingly higher in number of datasets 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 results 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 17 in Appendix C provides the raw Normalised Winning Number data for the Learning-to-Rank methods.

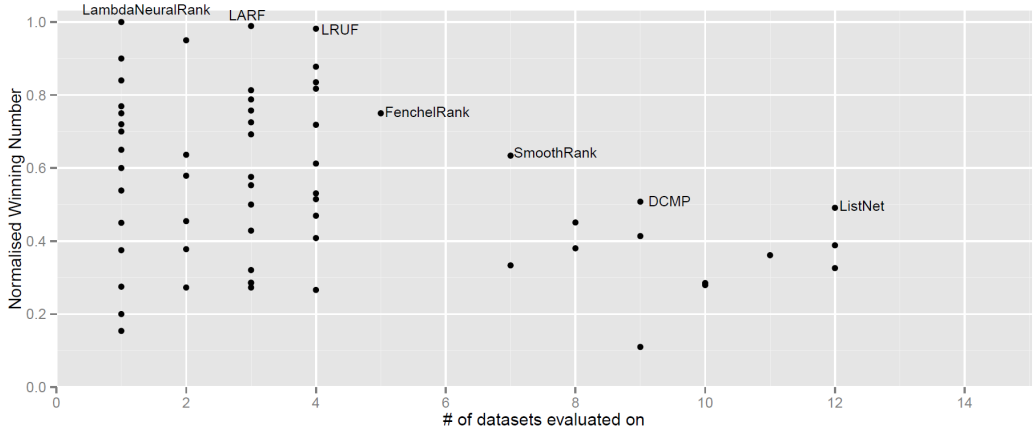

Figure 7: nDCG@5 comparison of Learning-to-Rank methods

LambdaNeuralRank again beat all other methods solely with results on the AOL dataset scoring a Normalised Winning Number of 1.0. LARF, LRUF, FenchelRank, SmoothRank, DCMP and ListNet are from left to right evaluated on an increasing number of datasets, but score decreasingly well in terms of Normalised Winning Number. 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 18 in Appendix D provides the raw Normalised Winning Number data for the Learning-to-Rank methods.

LambdaMART and LambdaNeuralRank score a Normalised Winning Number 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 Normalised Winning Number and number of datasets 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, but were close by, as can be

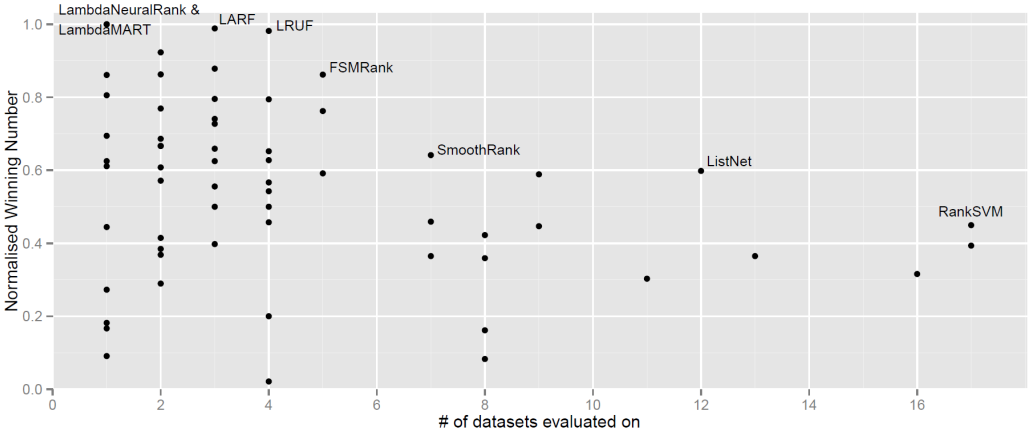

Figure 8: nDCG@10 comparison of Learning-to-Rank methods

seen in Tables 16 and 17 in the Appendices. 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 19 in Appendix E provides the raw Normalised Winning Number data for the Learning-to-Rank methods.

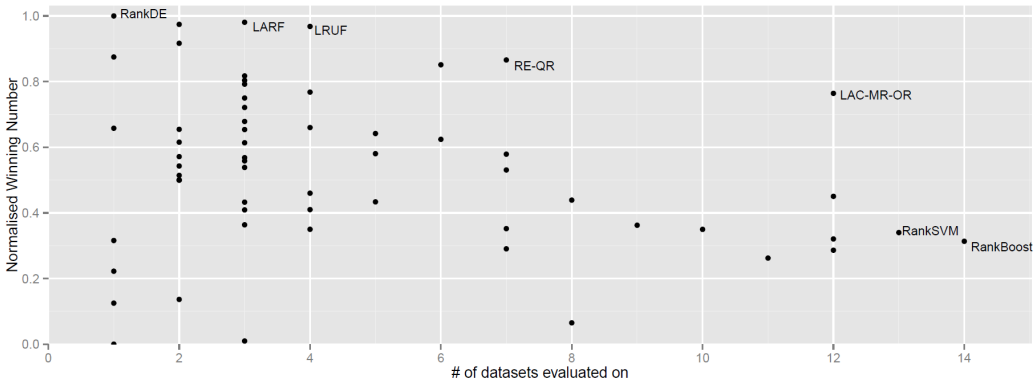

Figure 9: MAP comparison of Learning-to-Rank methods

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 Normalised Winning Number 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 Normalised Winning Number scores, but based on only relatively few datasets, 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 16 to 19 in Appendices B to E shows that LAC-MR-OR is evaluated on many more datasets for MAP compared to nDCG, which resulted in LAC-MR-OR obtaining equal certainty to ListNet with a higher Normalised Winning Number. SmoothRank performed a Normalised Winning Number of around 0.53 over 7 datasets, 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 datasets included in the comparison for the MAP-metric, but given their Normalised Winning Number 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 Normalised Winning Number as function of Ideal Winning Number for the methods described in Table 13. Table 20 in Appendix F provide the raw data plotted in Figure 10.

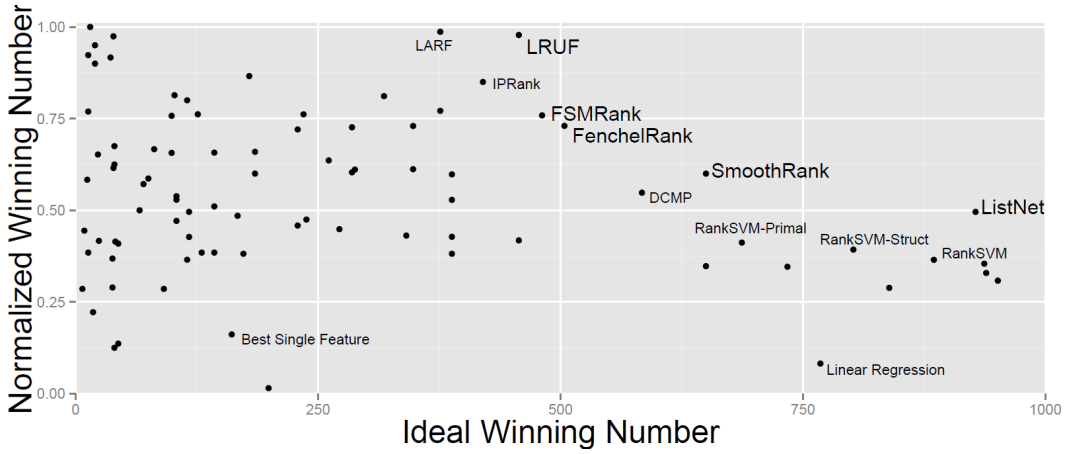

Figure 10: Cross-benchmark comparison of Learning-to-Rank methods

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 Ideal Winning Number and Normalised Winning Number. 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.

LARF, IPRank and DCMP and several variants of RankSVM 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 Normalised Winning Number measures, but have low Ideal Winning Number measures. Further evaluation runs of these methods on benchmark datasets that they are not yet evaluated on are desirable. The DirectRank paper [183] shows that the method is evaluated on more datasets than the number of datasets that we included evaluation results for in this meta-analysis. Some of the DirectRank measurements could not be used because measurements on some datasets 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 datasets. Because LAC-MR-OR is only evaluated on two datasets 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 Normalised Winning Number 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 datasets than on other datasets.

A second limitation is that the datasets 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 datasets that showed the most positive results for their Learning-to-Rank

method.

Another limitation lies in the meta-analysis methodology. Using evaluation results published by other researchers relies on the honesty of those researchers. Overly optimistic evaluation results published by other researchers do affect our Normalised Winning Number 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 the Ideal Winning Number into account in Figure 10 we further mitigate this limitation, as the Ideal Winning Number is loosely related with the number of studies that reported evaluation results for an algorithm.

5.6 CONCLUSIONS

We will now look back on our first research question using the knowledge gained with the cross-benchmark comparison.

RQ1 What are the best performing Learning-to-Rank algorithms in terms of accuracy on relevant benchmark datasets?

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.

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 datasets that these methods have not yet been evaluated on.

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 [35] is a listwise ranking function whose loss function is not directly related to information retrieval evaluation metrics. 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 uses the Plackett-Luce model [148, 126], which is one of the most well-known permutation probability distributions. The Plackett-Luce model defines the probability of all possible permutations π, given all document ranking scores S, as shown in Equation 1.

$$P(\pi|S) = \prod_{j=1}^{m} \frac{\phi(s_{\pi^{-1}(j)})}{\sum_{u=1}^{m} \phi(s_{\pi^{-1}(u)})} \tag{1}$$

Kullback-Leibler divergence

ListNet uses Gradient Descent to optimise a neural network such that its Cross Entropy loss compared to the Plackett-Luce distribution over the training data relevance labels is minimal. Note that some sources, including Liu [123], describe ListNet as using Kullback-Leibler divergence (KL divergence) as loss function. KL divergence and Cross Entropy are however identical up to an additive constant when comparing distribution q against a fixed reference distribution p. This follows from the definition of Cross Entropy Loss, which is shown in Equation 2.

$$H(p, q) = H(p) + D_{KL}(p||q) \tag{2}$$

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

Equation 3 describes the gradient descent step to minimise loss function $L(y^{(i)}, z^{(i)}(f_\omega))$ with respect to parameter ω.

$$\Delta\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial\omega} = -\sum_{\forall g \in \mathscr{G}_k} \frac{\partial P_{z^{(i)}(f_\omega)}(g)}{\partial\omega} \frac{P_{y^{(i)}}(g)}{P_{z^{(i)}(f_\omega)}(g)} \tag{3}$$

Algorithm 2 shows the pseudo-code of the ListNet training phase.

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

Algorithm 2: Learning algorithm of ListNet, obtained from [35]

6.2 SMOOTHRANK

SmoothRank [43] 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, I apply the SmoothRank approximation method to the nDCG evaluation measure for illustration, but the same procedure can be applied to MAP or any other information retrieval measure. The smoothing function used in SmoothRank is based on the softmax activation function [26], which is often used in neural networks. The softmax function is formally defined as shown in Equation 4.

$$p_i = \frac{e^{f_i/\sigma}}{\sum_{j=1}^{m} e^{f_j/\sigma}} \tag{4}$$

where σ is the smoothing parameter. Chapelle et al. [43] 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 5.

$$h_{i,j} = e^{-\frac{(f(x_i)) - (f(x_{d(j)}))^2}{\sigma}} \Big/ \sum_{k=1}^{m} e^{-\frac{(f(x_k)) - f(x_{d(j)})^2}{\sigma}} \tag{5}$$

It can be shown that the derivative of the smoothed nDCG version shown in Equation 5 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 [168], 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 5 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 σ 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: Learning algorithm of SmoothRank, obtained from [43]

6.3 FENCHELRANK

FenchelRank [108] 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 [159] and uses a generic algorithm framework proposed by Shalev-Shwartz and Singer [167]. FenchelRank optimises the objective function shown in Equation 6 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 $||x||_1 = \sum_i |x_i|$.

$$\min_w G(w) = \min_w I_{||w||_1 \leqslant 1}(w) + \frac{r^2}{p}\sum_{i=1}^{p} \max(0, \frac{1}{r} - (Kw)_i)^2 \tag{6}$$

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 6 is not differentiable everywhere because of its ℓ_1-norm regularisation term. Fenchel's duality theorem [159], defined in Equation 7, provides a way to approximate the optimal value of this non-differentiable optimisation problem by instead solving the Fenchel dual of the optimisation problem.

$$\min_x(f(x) - g(x)) = \max_p(f_*(p) - f^*(p)) \tag{7}$$

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 6), Lai et al. define $D(w)$ as $D(w) = -G(w)$. Equation 8 shown the resulting Fenchel dual of the ℓ_1-regularised pairwise loss function, which is the loss function that is used in FenchelRank.

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

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 ϵ, maximum number of iterations T and the radius r of the $\ell 1$ ball.

Initialize: $w_1 = 0_m$
for $t \leftarrow 1$ **to** T **do**
  //check if the early stopping criterion is satisfied
  if $||g_t||_\infty + \langle d_t, -Kw_t \rangle \leqslant \epsilon$ **then**
    //here $d_t = \nabla f^*(-Kw_t) = \frac{\partial f^*(-Kw)}{\partial(Kw)} | w = w_t$,
    // $\langle x, y \rangle$ represents the inner products of vectors x and y, and
    // $g_t = d_t^T K$
    return w_t as ranking predictor w
  end
  //greedily choose a feature to update
  Choose $j_t = \arg\max_j |(g_t)_j)|$
  //compute an appropriate step size
  Let $\mu_t = \arg\max_{0 \leqslant \mu_t \leqslant 1} D((1 - \mu_t)w_t + \mu_t \operatorname{sign}((g_t)_{j_t}) e^{j_t})$
  //update the model with the chosen feature and step size
  Update $w_{t+1} = (1 - \mu_t)w_t + \mu_t \operatorname{sign}((g_t)j_t) e^{j_t}$
end
return w_T as ranking predictor for w

Algorithm 4: Learning algorithm of FenchelRank, obtained from [108]

6.4 FSMRANK

Lai et al. [111] 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. [111] 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 [111] 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 [134]. 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 9.

$$\min_{\hat{w}} \frac{\lambda_1}{2} \hat{w}^T \hat{A} \hat{w} + \lambda_2 \|\hat{w} \oslash \hat{s}\|_1 + f(\hat{w}, (\hat{X}^{(q_k)}, Y^{(q_k)})_{k=1}^{n}) \tag{9}$$

The first term in Equation 9 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 9 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. [111] 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 $\min_w l(w) + r(w)$ where $l(w)$ is a convex function with Lipschitz gradient and $r(w)$ is convex, but non-smooth. The optimization function of Equation 9 is reformulated to match this form as presented in Equation 10.

$$\arg\min_{w_t : w_t \in \mathbb{R}} Q(w_t, z_t) = \lambda_2 \sum_{i=1}^{2d} \frac{w_i}{s_i} + \langle l'(z_t), w_t - z_t \rangle + \frac{L}{2} \|w_t - z_t\|^2 \tag{10}$$

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: λ_1, λ_2, T and L_0
Initialize: $w_0 = z_1 = 0, \alpha_1 = 1, \gamma = 2, L = L_0/\gamma^{10}$
for $t \leftarrow 1$ **to** T **do**
  Let $g = l'(z_t)$
  while *true* **do**
    //projection step
    $w_t = \arg\min_{w_t : w_t \in \mathbb{R}^{2d}_{+}} Q(w_t, z_t)$
    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**
      break
    end
    $L = \lambda L$
  end
  if $\frac{|F(w_t) - F(w_{t-1})|}{|F(w_{t-1})|} \leqslant \epsilon_s$ **then**
    //early stopping criterion
    break
  end
  $\alpha_{t+1} = \frac{1+\sqrt{1+4\alpha_t^2}}{2}$
  $z_{t+1} = w_t + \frac{\alpha_t - 1}{\alpha_t + 1}(w_t - w_{t-1})$
end

Algorithm 5: Learning algorithm of FSMRank, obtained from [111]

6.5 LRUF

LRUF [188] 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 11 when the selected action $a_i(k)$ is rewarded by the environment.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)] & \text{if } j = i \\ (1-a)p_j(k) & \text{otherwise} \end{cases} \tag{11}$$

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 [185], 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 11. 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 12.

$$g_i = \frac{1}{N_i} \sum_{d_i^j \in \underline{f_i}} a(r_i^j)^{-1} \tag{12}$$

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 [188] 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
Assume: Let A_i be the learning automaton corresponding to query q_i with action-set α_i
Action $\alpha_i^j \in \alpha_i$ is associated with document $d_i^j \in \underline{d_i}$
Let k denote the stage number
Let G be the total relevance score
Initialise: $k \leftarrow 1$, $T_i \leftarrow 0$
while $k \leqslant n_i$ **do**
    A_i chooses one of its actions (e.g. α_i^k) at random
    Document d_i^j corresponding to selected action α_i^j is ranked at K^{th} position of $\underline{R_i}$
    Configuration of A_i is updated by disabling action α_i^j
    $k \leftarrow k + 1$
end
Ranking $\underline{R_i}$ is shown to the user
$N_i \leftarrow 0$, $\underline{f_i} \leftarrow \emptyset$, $G \leftarrow 0$
repeat
    for *every document* d_i^j *visited by user* **do**
        $\underline{f_i} \leftarrow \underline{f_i} + d_i^j$
        $G \leftarrow G + a * (r_i^j)^{-1}$
    end
    $N_i \leftarrow N_i + 1$
until *query session is expired;*
$g_i \leftarrow \frac{G}{N_i}$
Configuration of A_i is updated by re-enabling all disabled actions
if $g_i \geqslant T_i$ **then**
    Reward the actions corresponding to all visited documents by Equation 11
    $T_i \leftarrow g_i$
end
for $\forall \alpha_i^j \in \underline{\alpha_i}$ **do**
    if $p_i^j < T_\epsilon$ **then**
        d_i^j is replaced by another document of the searched results
    end
end
Output $\underline{R_i}$

Algorithm 6: LRUF algorithm, obtained from [188]

7

IMPLEMENTATION

Describes implementation details of the Learning-to-Rank algorithm parallel implementations in HDInsight that are either Hadoop, Microsoft Azure, or HDInsight and are not part of the algorithm specification itself.

8

RESULTS & DISCUSSION

9

CONCLUSIONS

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

Table 14: Features of the LETOR OHSUMED dataset, obtained from Qin et al [152]

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-lin
12	TF-IDF of anchor	47	Hyperlink based feature propagation: weighted out-li
13	TF-IDF of title	48	Hyperlink based feature propagation: uniform out-lin
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 [230] 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 15: Features of the LETOR .GOV dataset, obtained from Qin et al [152]

B

RAW DATA FOR NDCG@3 COMPARISON

Method	Normalised Winning Number	# of datasets
AdaRank-MAP	0.34862385	12
AdaRank-nDCG	0.30733945	12
ApproxnDCG	0.79487179	1
BagBoo	0.84782609	2
BL-MART	0.95238095	2
BoltzRank-Pair	0.83333333	4
BoltzRank-Single	0.75490196	4
BT	0.72727273	3
CoList	1.00000000	1
DCMP	0.54591837	9
EnergynDCG	0.36363636	2
FBPCRank	0.41463415	3
FenchelRank	0.77419355	5
FocusedBoost	0.37500000	2
FocusedNet	0.45833333	2
FocusedSVM	0.25000000	2
FRank	0.30845771	11
FSMRank	0.83333333	4
FSMSVM	0.22916667	4
GAS-E	0.37500000	4
GPRank	0.87096774	3
GroupCE	0.73118280	3
GroupMLE	0.52688172	3
IntervalRank	0.58974359	1
IPRank	0.93548387	3
KL-CRF	0.59459459	2
LambdaMART	0.57142857	2
LambdaNeuralRank	1.00000000	1
LambdaRank	0.20000000	1
LARF	0.98924731	3
Linear Regression	0.07142857	9
ListMLE	0.00000000	1
ListNet	0.44954128	12

ListReg	0.73118280	3
LRUF	0.98230088	4
MHR	0.75000000	1
NewLoss	0.51612903	3
OWPC	0.65000000	6
PERF-MAP	0.38938053	4
RankBoost	0.32110092	12
RankELM (pairwise)	0.61538462	1
RankELM (pointwise)	0.69230769	1
RankNet	0.18867925	1
RankSVM	0.30097087	12
RankSVM-Primal	0.39204545	8
RankSVM-Struct	0.35204082	9
REG-SHF-SDCG	0.38461538	1
Ridge Regression	0.40880503	7
RSRank	0.57291667	4
SmoothRank	0.60377358	7
SoftRank	0.23076923	1
SortNet	0.25000000	2
SparseRank	0.82242991	4
SVMMAP	0.28930818	7
TGRank	0.54166667	4
TM	0.59090909	3

Table 16: Raw data of Normalised Winning Number comparison on nDCG@3

C

RAW DATA FOR NDCG@5 COMPARISON

Method	Normalised Winning Number	# of datasets
AdaRank-MAP	0.3883929	12
AdaRank-nDCG	0.3258929	12
ApproxnDCG	0.7500000	1
BagBoo	0.8400000	1
BL-MART	0.7200000	1
BoltzRank-Pair	0.8349515	4
BoltzRank-Single	0.7184466	4
BT	0.7878788	3
C-CRF	0.9500000	2
DCMP	0.5078534	9
EnergynDCG	0.3777778	2
FBPCRank	0.5529412	3
FenchelRank	0.7500000	5
FocusedBoost	0.4545455	2
FocusedNet	0.6363636	2
FocusedSVM	0.2727273	2
FPRank	0.9000000	1
FRank	0.2849462	10
FSMRank	0.8775510	4
FSMSVM	0.4081633	4
GAS-E	0.4693878	4
GPRank!	0.7252747	3
IntervalRank	0.3750000	1
IPRank	0.8131868	3
Kernel-PCA	0.2857143	3
KL-CRF	0.5789474	2
LambdaNeuralRank	1.0000000	1
LambdaRank	0.2000000	1
LARF	0.9890110	3
Linear Regression	0.1099476	9
ListNet	0.4910714	12
ListReg	0.6923077	3
LRUF	0.9816514	4

MHR	0.6000000	1
NewLoss	0.4285714	3
PERF-MAP	0.2660550	4
Q.D.KNN	0.3205128	3
RankAggnDCG	0.5000000	3
RankBoost	0.2794118	10
RankDE	0.5384615	1
RankELM (pairwise)	0.6500000	1
RankELM (pointwise)	0.7000000	1
RankNet	0.2857143	3
Rank-PMBGP	0.7692308	1
RankSVM	0.3612565	11
RankSVM-Primal	0.4508671	8
RankSVM-Struct	0.4136126	9
RCP	0.5757576	3
REG-SHF-SDCG	0.4500000	1
Ridge Regression	0.3333333	7
RSRank	0.5306122	4
SmoothRank	0.6339869	7
SoftRank	0.2750000	1
SortNet	0.5147059	4
SparseRank	0.8173077	4
SVD-RankBoost	0.2727273	3
SVMMAP	0.3801170	8
SwarmRank	0.1538462	1
TGRank	0.6122449	4
TM	0.7575758	3

Table 17: Raw data of Normalised Winning Number comparison on nDCG@5

D

RAW DATA FOR NDCG@10 COMPARISON

Method	Normalised Winning Number	# of datasets
AdaRank-MAP	0.36480687	13
AdaRank-nDCG	0.31578947	16
ADMM	0.44444444	1
ApproxnDCG	0.86111111	1
Best Single Feature	0.16149068	8
CA	0.65217391	4
CoList	0.16666667	1
Consistent-RankCosine	0.76923077	2
DCMP	0.58883249	9
DirectRank	0.92307692	2
EnergynDCG	0.41463415	2
FenchelRank	0.76229508	5
FocusedBoost	0.68627451	2
FocusedNet	0.86274510	2
FocusedSVM	0.60784314	2
FRank	0.30288462	11
FSMRank	0.86206897	5
FSMSVM	0.54255319	4
GAS-E	0.45744681	4
GP	0.66666667	2
GPRank	0.65909091	3
GRankRLS	0.28947368	2
GroupCE	0.72727273	3
GroupMLE	0.62500000	3
IPRank	0.79545455	3
LAC-MR-OR	0.66666667	2
LambdaMART	1.00000000	1
LambdaNeuralRank	1.00000000	1
LambdaRank	0.57142857	2
LARF	0.98863636	3
Linear Regression	0.08287293	8
ListMLE	0.02127660	4
ListNet	0.59821429	12

LRUF	0.98181818	4
MHR	0.62500000	1
NewLoss	0.39772727	3
PERF-MAP	0.20000000	4
Q.D.KNN	0.50000000	3
RandomForest	0.42236025	8
RankAggnDCG	0.87837838	3
RankBoost	0.39357430	17
RankDE	0.18181818	1
RankELM (pairwise)	0.69444444	1
RankELM (pointwise)	0.80555556	1
RankNet	0.59154930	5
Rank-PMBGP	0.27272727	1
RankRLS	0.36842105	2
RankSVM	0.44957983	17
RankSVM-Primal	0.45911950	7
RankSVM-Struct	0.44670051	9
RCP	0.74074074	3
Ridge Regression	0.36477987	7
RSRank	0.62765957	4
SmoothGrad	0.38461538	2
SmoothRank	0.64150943	7
SoftRank	0.61111111	1
SortNet	0.56666667	4
SparseRank	0.79439252	4
SVD-RankBoost	0.55555556	3
SVMMAP	0.35911602	8
SwarmRank	0.09090909	1
TGRank	0.50000000	4

Table 18: Raw data of Normalised Winning Number comparison on nDCG@10

E

RAW DATA FOR MAP COMPARISON

Method	Normalised Winning Number	# of datasets
Method	Normalised Winning Number	# of datasets
AdaRank-MAP	0.320610687	12
AdaRank-nDCG	0.286259542	12
ApproxAP	0.500000000	2
BagBoo	0.654545455	2
BL-MART	0.803571429	3
BoltzRank-Pair	0.580419580	5
BoltzRank-Single	0.433566434	5
BT	0.750000000	3
CCRank	0.615384615	2
FenchelRank	0.641791045	5
FRank	0.262295082	11
FSMRank	0.578947368	7
FSMSVM	0.350000000	4
GAS-E	0.410000000	4
GP	0.500000000	2
GPRank	0.817307692	3
GroupCE	0.721153846	3
GroupMLE	0.653846154	3
IntervalRank	0.315789474	1
IPRank	0.851351351	6
KeepRank	0.538461538	3
LAC-MR-OR	0.764192140	12
LambdaMART	0.678571429	3
LARF	0.980769231	3
Linear Regression	0.065000000	8
ListMLE	0.009615385	3
ListNet	0.450381679	12
ListReg	0.432692308	3
LRUF	0.968000000	4
MCP	0.571428571	2
MHR	0.000000000	1
MultiStageBoost	0.136363636	2

OWPC	0.624113475	6
PERF-MAP	0.768000000	4
PermuRank	0.409090909	3
Q.D.KNN	0.558441558	3
RandomForest	0.438888889	8
RankAggnDCG	0.792207792	3
RankBoost	0.313432836	14
RankCSA	0.916666667	2
RankDE	1.000000000	1
RankELM (pairwise)	0.514285714	2
RankELM (pointwise)	0.542857143	2
RankMGP	0.222222222	1
Rank-PMBGP	0.875000000	1
RankSVM	0.340000000	13
RankSVM-Primal	0.351955307	7
RankSVM-Struct	0.362385321	9
RCP	0.363636364	3
REF-SHF-SDCG	0.657894737	1
RE-QR	0.865921788	7
Ridge Regression	0.290502793	7
RSRank	0.660000000	4
SmoothRank	0.530726257	7
SortNet	0.500000000	2
SVD-RankBoost	0.568181818	3
SVMMAP	0.349775785	10
SwarmRank	0.125000000	1
TGRank	0.460000000	4
TM	0.613636364	3
VFLR	0.974358974	2

Table 19: Raw data of Normalised Winning Number comparison on MAP

F

RAW DATA ON NORMALISED WINNING NUMBER FOR CROSS-COMPARISON

Method	Winning Number	Ideal Winning Number	Normalized Winning Number
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 20: Raw data of Normalised Winning Number comparison 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. *Knowledge and Data Engineering, IEEE Transactions on 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] 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.

[7] 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).

[8] 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.

[9] 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 1*, 3 (2010), 551.

[10] 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.

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

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

[13] ASADI, N., AND LIN, J. Training Efficient Tree-Based Models for Document Ranking. In *Proceedings of the 25th European Conference on Advances in Information Retrieval* (2013), vol. 7814, pp. 146–157.

[14] ASADI, N., LIN, J., AND DE VRIES, A. Runtime Optimizations for Prediction with Tree-Based Models. *IEEE Transactions on Knowledge and Data Engineering PP*, 99 (2013), 1.

[15] BANERJEE, S., DUBEY, A., MACHCHHAR, J., AND CHAKRABARTI, S. Efficient and accurate local learning for ranking. In *SIGIR workshop on Learning to rank for information retrieval* (2009), pp. 1–8.

[16] BARTLETT, P. L., JORDAN, M. I., AND MCAULIFFE, J. D. Convexity, classification, and risk bounds. *Journal of the American Statistical Association 101*, 473 (2006), 138–156.

[17] BEN-HAIM, Y., AND TOM-TOV, E. A streaming parallel decision tree algorithm. *The Journal of Machine Learning Research 11* (2010), 849–872.

[18] 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.

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

[20] 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* (2010).

[21] BIDOKI, A., AND THOM, J. 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.

[22] 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* (2011), pp. 1771–1778.

[23] BOYD, S., CORTES, C., JIANG, C., MOHRI, M., RADOVANOVIC, A., AND SKAF, J. Large-scale Distributed Optimization for Improving Accuracy at the Top. In *NIPS Workshop on Optimization for Machine Learning* (2012).

[24] BREIMAN, J. H., FRIEDMAN, R. A., STONE, J. C., AND OLSHEN, L. *Classification and regression trees*. Wadsworth International Group, 1984.

[25] BREIMAN, L. Bagging predictors. *Machine learning 24*, 2 (1996), 123–140.

[26] 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.

[27] 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.

[28] 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* (2005), pp. 89–96.

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

[30] 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.

[31] 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-Proceedings Track 14* (2011), 25–35.

[32] 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.

[33] 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.

[34] 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.

[35] 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* (2007), pp. 129–136.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] Chapelle, O., Chang, Y., and Liu, T.-Y. Future directions in learning to rank. *JMLR Workshop and Conference Proceedings 14* (2011), 91–100.

[42] 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.

[43] Chapelle, O., and Wu, M. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval 13*, 3 (2010), 216–235.

[44] 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* (2008), pp. 101–107.

[45] 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.

[46] 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.

[47] Chen, M., Weinberger, K. Q., Chapelle, O., Kedem, D., and Xu, Z. Classifier cascade for minimizing feature evaluation cost. In *International Conference on Artificial Intelligence and Statistics* (2012), pp. 218–226.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

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

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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* (2011), pp. 17–20.

[60] 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.

[61] 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.

[62] 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.

[63] Dean, J., and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2004), 107–113.

[64] 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.

[65] 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.

[66] 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.

[67] 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* (2009), pp. 9–16.

[68] 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.

[69] 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.

[70] 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.

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

[72] 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).

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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 55*, 1 (1997), 119–139.

[78] Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* (2001), 1189–1232.

[79] Friedman, J. H. Stochastic gradient boosting. *Computational Statistics & Data Analysis 38*, 4 (2002), 367–378.

[80] Ganjisaffar, Y. *Tree ensembles for learning to rank*. PhD thesis, University of California, Irvine, 2011.

[81] Ganjisaffar, Y., Caruana, R., and Lopes, C. V. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 85–94.

[82] 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.

[83] 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* (2014), pp. 63–72.

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

[85] Geng, X., Liu, T.-Y., Qin, T., Arnold, A., Li, H., and Shum, H.-Y. Query dependent 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.

[86] 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.

[87] Geng, X., Qin, T., Liu, T.-Y., Cheng, X.-Q., and Li, H. Selecting optimal training data for learning to rank. *Information Processing & Management 47*, 5 (2011), 730–741.

[88] Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine learning 63*, 1 (2006), 3–42.

[89] Geurts, P., and Louppe, G. Learning to rank with extremely randomized trees. In *JMLR: Workshop and Conference Proceedings* (2011), vol. 14.

[90] 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 4*, 1 (2013), 57.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] Herbrich, R., Graepel, T., and Obermayer, K. Large margin rank boundaries for ordinal regression. pp. 115–132.

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

[97] 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.

[98] Holland, J. H. *Hidden Order: How Adaptation Builds Complexity*. Ulam lecture series. Perseus Books, 1995.

[99] 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.

[100] Huber, P. J. *Robust statistics*. John Wiley and Sons, New York, 1981.

[101] Järvelin, K., and Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS) 20*, 4 (2002), 422–446.

[102] Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2002), pp. 133–142.

[103] Kao, C.-Y., and Fahn, C.-S. A multi-stage learning framework for intelligent system. *Expert Systems with Applications 40*, 9 (2013), 3378–3388.

[104] 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* (2011), pp. 377–386.

[105] 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.

[106] 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.

[107] 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.

[108] 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 62*, 6 (2013), 1221–1233.

[109] 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 49* (2013), 190–198.

[110] Lai, H., Tang, Y., Luo, H., 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.

[111] 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 24*, 6 (2013), 940–952.

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

[113] Le, Q., and Smola, A. Direct optimization of ranking measures. Tech. rep., NICTA, 2007.

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

[115] 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.

[116] 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.

[117] 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.

[118] 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.

[119] 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.

[120] 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.

[121] 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.

[122] 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.

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

[124] 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.

[125] 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* (2012), pp. 113–122.

[126] Luce, R. D. Individual choice behavior, a theoretical analysis. *Bulletin of the American Mathematics Society 66* (1959), 2–9904.

[127] 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.

[128] 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* (2010), pp. 135–146.

[129] 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.

[130] Metzler, D., and Croft, W. B. Linear feature-based models for information retrieval. *Information Retrieval 10*, 3 (2007), 257–274.

[131] 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.

[132] 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.

[133] 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* (2010), pp. 151–160.

[134] Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, vol. 87. Springer, 2004.

[135] 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.

[136] Niu, S., Guo, J., Lan, Y., and Cheng, X. Top-k learning to rank: labeling, ranking and evaluation. In *Proceedings of the 35th international ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), pp. 751–760.

[137] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank citation ranking: Bringing order to the web.

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

[139] 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.

[140] 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* (2013).

[141] Pan, Y., Luo, H.-X., Tang, Y., and Huang, C.-Q. Learning to rank with document ranks and scores. *Knowledge-Based Systems 24*, 4 (2011), 478–483.

[142] 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).

[143] 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.

[144] 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.

[145] 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)* (Berlin, Heidelberg, 2010), pp. 114–126.

[146] 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.

[147] 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.

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

[149] 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.

[150] 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.

[151] 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.

[152] Qin, T., Liu, T.-Y., 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.

[153] 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.

[154] 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* (2008), pp. 407–416.

[155] 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 44*, 2 (2008), 838–855.

[156] 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.

[157] Ravikumar, P. D., Tewari, A., and Yang, E. On NDCG consistency of list-wise ranking methods. In *International Conference on Artificial Intelligence and Statistics* (2011), pp. 618–626.

[158] 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.

[159] Rifkin, R. M., and Lippert, R. A. Value regularization and Fenchel duality. *The Journal of Machine Learning Research 8* (2007), 441–479.

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

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

[162] 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.

[163] Rudin, C. The P-Norm Push: A simple convex ranking algorithm that concentrates at the top of the list. *The Journal of Machine Learning Research 10* (2009), 2233–2271.

[164] 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.

[165] Schapire, R. E. The strength of weak learnability. *Machine learning 5*, 2 (1990), 197–227.

[166] Sculley, D. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking* (2009), pp. 1–6.

[167] 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.

[168] Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University, 1994.

[169] Shivaswamy, P. K., and Joachims, T. Online learning with preference feedback. *arXiv preprint arXiv:1111.0712* (2011).

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

[171] 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.

[172] 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* (2013), pp. 354–369.

[173] Song, Y., Ng, W., Leung, K.-T., and Fang, Q. SFP-Rank: significant frequent pattern analysis for effective ranking. *Knowledge and Information Systems* (2014), 1–25.

[174] Sorokina, D., Caruana, R., and Riedewald, M. Additive groves of regression trees. In *Proceedings of the 18th European Conference on Machine Learning* (2007), pp. 323–334.

[175] Sousa, D., Rosa, T., Martins, W., Silva, R., and Gonçalves, M. Improving On-Demand Learning to Rank through Parallelism. In *Proceedings of the 13th International Conference on Web Information Systems Engineering* (2012), vol. 7651, pp. 526–537.

[176] 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.

[177] 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.

[178] Sun, X.-H., and Gustafson, J. L. Toward a better parallel performance metric. *Parallel Computing 17*, 10 (1991), 1093–1109.

[179] Sun, Z., Qin, T., Tao, Q., and Wang, J. Robust sparse rank learning for non-smooth ranking measures. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 259–266.

[180] 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).

[181] 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.

[182] 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.

[183] 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* (2013), pp. 856–864.

[184] 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.

[185] Thathachar, M. A. L., and Harita, B. R. Learning automata with changing number of actions. *Systems, Man and Cybernetics, IEEE Transactions on 17*, 6 (1987), 1095–1100.

[186] 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* (2009), vol. 5552, pp. 345–353.

[187] Torkestani, J. A. An adaptive learning automata-based ranking function discovery algorithm. *Journal of Intelligent Information Systems 39*, 2 (2012), 441–459.

[188] Torkestani, J. A. An adaptive learning to rank algorithm: Learning automata approach. *Decision Support Systems 54*, 1 (2012), 574–583.

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

[190] 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.

[191] 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* (2011), pp. 387–396.

[192] Usunier, N., Buffoni, D., and Gallinari, P. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), pp. 1057–1064.

[193] Veloso, A., Gonçalves, M. A., Meira Jr, W., and Mossri, H. Learning to rank using query-level rules. *Journal of Information and Data Management 1*, 3 (2010), 567.

[194] 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.

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

[196] 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* (2009), pp. 1089–1096.

[197] 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* (2012), pp. 479–488.

[198] 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.

[199] 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.

[200] 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.

[201] 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 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2012), pp. 490–495.

[202] 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.

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

[204] 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* (2011).

[205] 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* (2011).

[206] Wang, S., Gao, B. J., Wang, K., and Lauw, H. W. Parallel learning to rank for information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (2011), pp. 1083–1084.

[207] 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.

[208] 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.

[209] 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* (July 2008), vol. 5, pp. 2629–2634.

[210] Wolpert, D. H. Stacked generalization. *Neural networks 5*, 2 (1992), 241–259.

[211] 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 international ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 1091–1092.

[212] 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.

[213] Wu, Q., Burges, C. J. C., Svore, K. M., and Gao, J. Ranking, boosting, and model adaptation. Tech. rep., Microsoft Research, 2008.

[214] 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* (2008), pp. 1192–1199.

[215] 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.

[216] 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.

[217] 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 International Conference on Field-Programmable Technology (ICFPT)* (2007), pp. 33–40.

[218] 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.

[219] 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.

[220] 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.

[221] 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 FPL International Conference on Field Programmable Logic and Applications* (2009), pp. 662–665.

[222] 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* (2010), p. 285.

[223] 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.

[224] 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* (2012), pp. 97–100.

[225] 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.

[226] 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 Workshop on Learning to Rank for Information Retrieval (LR4IR)* (2007).

[227] 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.

[228] Yue, Y., Finley, T., Radlinski, F., and Joachims, T. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 271–278.

[229] 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).

[230] 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.

[231] 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.

[232] 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.

[233] 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.

[234] 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.

[235] Zhu, M. Recall, precision and average precision. Tech. rep., Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, 2004.

[236] 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.

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

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

DECLARATION

Put your declaration here.

Enschede, Februari 2014

Niek Tax