NIEK TAX

METHODS FOR LARGE SCALE LEARNING-TO-RANK

METHODS FOR LARGE SCALE LEARNING-TO-RANK

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

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

UNIVERSITY OF TWENTE.

avanade®
Results Realized

Februari 2014 – version 0.1

Niek Tax: *Methods for Large Scale Learning-to-Rank,* A study concerning parallelization of Learning-to-Rank algorithms using Hadoop, © Februari 2014

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

LISTINGS

ACRONYMS

AP	Average Precision
CC	Cooperative Coevolution
DCG	Discounted Cumulative Gain
ERR	Expected Reciprocal Rank
FPGA	Field-Programmable Gate Array
GBDT	Gradient Boosted Decision Tree
GPGPU	General-Purpose computing on Graphical Processing Units
GPU	Graphical Processing Unit
IDF	Inverse Document Frequency
MAP	Mean Average Precision
NDCG	Normalized Discounted Cumulative Gain
RLS	Regularised Least-Squares
SGD	Stochastic Gradient Descent
SIMD	Single Instruction Multiple Data
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
URL	Uniform Resource Locator

Part I

INTRODUCTION

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 query. Research in the field of ranking models has for a long time been based on manually designed ranking functions and has been an active area of research since Luhn[20] 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 covering 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[8], the Yandex Internet Mathematics competition[1], and the LETOR dataset[25] that was build 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[9]. Liu[17] 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 [12, 7]. 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 [1, 2, 6, 29, 27], since Liu [17] first stated its growing importance back in 2007. Only several of these studies [29, 27] have explored the possibilities of efficient Learning-to-Rank through the use of parallel programming paradigms.

MapReduce[14] 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 pro-

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

gramming framework back in 2004 it has since grown to be the industry standard model for parallel programming. Lin [16] 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 chapter 2.

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 is the speed-up of existing Learning-to-Rank algorithms when executed using the MapReduce framework?
Where the definition of *relative speed-up* is used for speed-up[30]:

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

RQ2 Can we adjust those Learning-to-Rank algorithms such that the parallel execution speed-up increases without decreasing accuracy?

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 ("large scale" OR "parallel" OR "distributed")

and the following bibliographic databases:

- Scopus
- Web of Science

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. *large scale* is a term in the search query that is likely to result in a decrease of precision, because this term is also often used in highly efficient sequential approaches. However, the *large scale* term is still included in the query to prevent missing relevant results.

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

To implement the Learning-to-Rank algorithms I will use cloud based MapReduce implementation from Microsoft was used that is called HDInsight. It which is based on the popular MapReduce open source implementation Hadoop[1]. The algorithms that we include in the measurements will be determined based on experimental results on the *Yahoo! Learning to Rank Challenge*[8], the Yandex Internet Mathematics competition[2], the LETOR[25] dataset and the LETOR successors MSLR-WEB10k and MSLR-WEB30k.

[1] http://hadoop.apache.org/

[2] http://imat-relpred.yandex.ru/en/

4 THESIS OVERVIEW

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

PART III: RELATED WORK concisely describes existing work in the field of parallel machine learning and parallel Learning-to-Rank.

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

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

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

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

PART 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.

Part II

TECHNICAL BACKGROUND

This part provides a background in the Learning-to-Rank field with the goal of making the subsequential parts of this thesis understandable for non-experts in the field.

5 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.

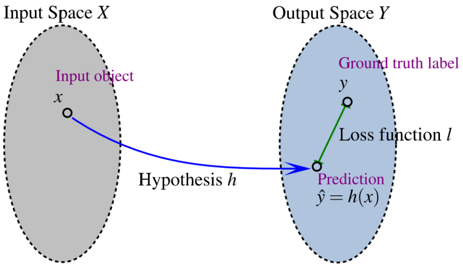

Figure 1: Machine learning framework for Learning-to-Rank, obtained from Liu[17]

Liu [17] 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 *Feature based* means that all documents under investigation are represented by feature vectors that reflect the relevance of the documents to the query.

DISCRIMINATIVE TRAINING *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, that can 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 are minimal in terms of a loss function.

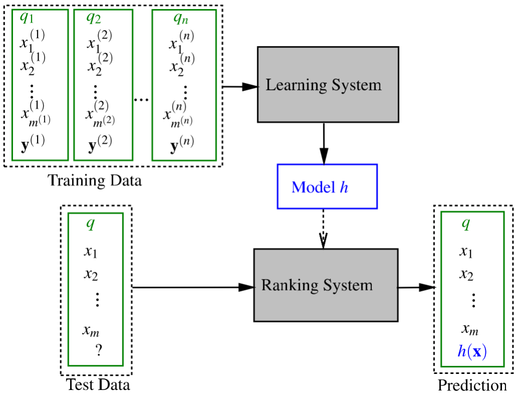

Figure 2: A typical Learning-to-Rank setting, obtained from Liu[17]

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 further on in this part.

6 EVALUATING 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 the succeeding section several frequently used evaluation metrics for ranked results will be described.

6.1 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.

6.1.1 *Discounted Cumulative Gain*

The Discounted Cumulative Gain (DCG)[3] at a position p is defined as

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

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.

6.1.2 *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. We can write the definition of nDCG down mathematically as

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

Table 1 shows an example calculation for nDCG.

	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
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

$$\text{nDCG} = \frac{732.7}{788.0} = 0.93$$

Table 1: Example calculation for nDCG

6.2 EXPECTED RECIPROCAL RANK

Expected Reciprocal Rank (ERR)[10] 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

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

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.

Listing 1: Algorithm to compute the ERR metric, obtained from [10]

```
p <- 1, ERR <- 0
for r=1 to n do
        R <- R(g[r])
        ERR <- ERR + p * R/r
        p <- p * (1-R)
end for
return ERR
```

6.3 MEAN AVERAGE PRECISION

Average Precision (AP)[41] is an often used binary relevance judgement based metric that can be seen as a trade-off between precision

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

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

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.

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

7 APPROACHES TO LEARNING-TO-RANK

7.1 POINTWISE APPROACH

Learn from each document in isolation. Through:

1. regression, when relevance is real-valued
2. classification, when relevance is categorical
3. ordinal regression

7.2 PAIRWISE APPROACH

Pointwise approach Learning-to-Rank methods 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.

7.3 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 all discontinuous and non-convex functions, what makes them very hard to optimize.

Part III

RELATED WORK

8 RELATED WORK

8.1 SEARCH 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 ("large scale" OR "parallel" OR "distributed")*. An abstract based manual filtering step is applied where I filter those results that actually use the *large scale*, *parallel* or *distributed* terms in context to the *learning to rank*, *learning-to-rank* or *machine learned ranking*. Studies focusing on efficient query evaluation instead of efficient model training are likely to meet all criteria listed. 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.

8.1.1 *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.

8.1.2 *Web of Science*

The defined search query resulted in 16 documents. Four of those documents were also present in the set of 65 documents found using Scopus, leaving 61 unique documents. Only four of those 61 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 focussed on parallel or distributed learning of ranking functions.

8.2 CCRANK

Wang et al[32, 33] propose 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 the baselines. It must be stated however that 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.

8.3 PARALLEL LISTNET USING SPARK

Shukla et al[27] 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 a linear training time reduction.

8.4 GRADIENT BOOSTED DISTRIBUTED DECISION TREES

Ye et al[38] described how to implement the Gradient Boosted Decision Tree (GBDT) process in a parallel manner using both MPI and Hadoop. GBDT's are shown to be able to achieve good accuracy in a Learning-to-Rank setting when used in a pairwise[40] or listwise[11] setting. Experiments showed the Hadoop implementation to result into 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.

8.5 NDCG-ANNEALING

Karimzadeghan el al[15] 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.

8.6 LOW COMPLEXITY LEARNING-TO-RANK

Designing Learning-to-Rank algorithms with low time complexity for training is a different another approach towards large scale Learning-to-Rank. Pahikkala et al[22] describe a pairwise Regularised Least-Squares (RLS) type of ranking function, RankRLS, with time complexity $\mathcal{O}(n^3 + n^2m)$, with n the number of features and m the number of training documents. The RankRLS ranking function showed ranking performance similar to RankSVM on the BioInfer corpus[24], a corpus for information extraction in the biomedical domain.

8.7 DISTRIBUTED STOCHASTIC GRADIENT DESCENT

Long et al[19] describe 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[42] 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.

8.8 FPGA-BASED LAMBDARANK

Yan et al[34, 35, 36, 37] described the development and incremental improvement of a Single Instruction Multiple Data (SIMD) architecture Field-Programmable Gate Array (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 other advantages of the FPGA approach next to faster model training. In their latest publication[37] 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.

8.9 GPGPU FOR LEARNING-TO-RANK

De Sousa et al[13] proposed a General-Purpose computing on Graphical Processing Units (GPGPU) approach using the Graphical Processing Unit (GPU) both learning the ranking function and for query processing, thereby improving both training time and query time. An association rule based Learning-to-Rank approach proposed by [31] has been implemented using the GPU in such a way that the set of rules van be computed in parallel, in different threads, 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.

Part IV

BENCHMARK RESULTS

This part describes benchmark characteristics like collection size and features and gives an overview of the performance of different Learning-to-Rank methods. Performance differences for a given Learning-to-Rank method are explained in terms of differences in benchmark characteristics. This section is concluded with a selection of Learning-to-Rank methods to include in the experiments.

9 YAHOO! LEARNING TO RANK CHALLENGE

Yahoo's observation that all existing benchmark datasets were too small to draw reliable conclusions, especially in comparison with datasets used in commercial search engines, prompted Yahoo to release two internal datasets from Yahoo! search. The Yahoo! Learning to Rank Challenge[8] 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.

	Train	Validation	Test
Queries	19,994	2,994	6,983
Documents	473,134	71,083	165,660
Features	519	-	-

Table 3: Yahoo! Learning to Rank Challenge dataset characteristics, as described in the challenge overview paper[8]

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 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[8]:

WEB GRAPH Quality and popularity metrics of web documents, e.g. PageRank[21].

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.

TEXT MATCH Textual similarity metrics between query and document. Includes Term Frequency - Inverse Document Frequency (TF-IDF), BM25[26] 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.

9.1 RESULTS

Figure 4 shows the top five participants in the Yahoo! Learning to Rank Challenge in terms of ERR score. The top five participants all used decision trees and ensemble methods. Burges et al[5] created a linear combination ensemble of eight LambdaMART[4], 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[23] ensemble technique, which combines bagging and boosting. Sorokina used a similar combination of bagging and boosting that is called Additive Groves[28].

The challenge overview paper states as one of the lessons of the challenge that the simple baseline GBDT model performed very well

[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[8]

with a small performance gap to the complex ensemble submissions at the top of the table.

10 YANDEX INTERNET MATHEMATICS COMPETITION

10.1 BENCHMARK CHARACTERISTICS

10.2 RESULTS

11 LETOR

The LETOR benchmark set was first released by Microsoft Research Asia in April 2007 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. The original LETOR benchmark collection[18] as released in 2007 contained two data sets: the OHSUMED collection and 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 TREC web track in 2003 and 2004. Tables 5 and 6 provide the descriptions of the features of the .gov and the OHSUMED collections respectively.

11.1 BENCHMARK CHARACTERISTICS

11.2 RESULTS

ID	Feature Description	ID	Feature Description
1	Term Frequency (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 Uniform Resource Locator (URL)	39	LMIR.JM of URL
5	TF of whole document	40	LMIR.JM of whole document
6	Inverse Document Frequency (IDF) of body	41	Sitemap based term propagation
7	IDF of anchor	42	Sitemap based score propagation
8	IDF of title	43	Hyperlink based score propagation: weighted in-link
9	IDF of URL	44	Hyperlink based score propagation: weighted out-link
10	IDF of whole document	45	Hyperlink based score propagation: uniform out-link
11	TF-IDF of body	46	Hyperlink based feature propagation: weighted in-link
12	TF-IDF of anchor	47	Hyperlink based feature propagation: weighted out-link
13	TF-IDF of title	48	Hyperlink based feature propagation: uniform out-link
14	TF-IDF of URL	49	HITS authority
15	TF-IDF of whole document	50	HITS hub
16	Document length of body	51	PageRank
17	Document length of anchor	52	HostRank
18	Document length of title	53	Topical PageRank
19	Document length of URL	54	Topical HITS authority
20	Document length of whole document	55	Topical HITS hub
21	BM25 of body	56	In-link number
22	BM25 of anchor	57	Out-link number
23	BM25 of title	58	Number of slashes in URL
24	BM25 of URL	59	Length of URL
25	BM25 of whole document	60	Number of child page
26	LMIR.ABS[39] 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 5: Features of the LETOR .GOV dataset, obtained from Qin et al[25]

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

Table 6: Features of the LETOR OHSUMED dataset, obtained from Qin et al[25]

12 MSLR-WEB10K/MSLR-WEB30K

Contrary to the Yahoo! Learning to Rank Challenge dataset, the MSLR-WEB30k provides detailed feature descriptions. The MSLR-WEB30k dataset however contains no proprietary features but only features that are commonly used in the research community.

12.1 benchmark characteristics

12.2 results

13 SELECTING LEARNING-TO-RANK METHODS

The Accuracies of the Learning-to-Rank methods described in the preceding chapters must 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[6] found that nDCG evaluation tools of benchmark data sets produced different scores

Part V

SELECTED LEARNING-TO-RANK METHODS

Algorithms and details of the well-performing Learning-to-Rank methods as selected in the aforegoing part are presented and explained in this part.

Part VI

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.

Part VII

RESULTS & DISCUSSION

Part VIII

CONCLUSIONS

Part IX

APPENDIX

BIBLIOGRAPHY

[1] Asadi, N., and Lin, J. Training Efficient Tree-Based Models for Document Ranking. In *Advances in Information Retrieval SE - 13*, P. Serdyukov, P. Braslavski, S. Kuznetsov, J. Kamps, S. Rüger, E. Agichtein, I. Segalovich, and E. Yilmaz, Eds., vol. 7814 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 146–157.

[2] 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.

[3] 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), ACM, pp. 89–96.

[4] Burges, C. J. C. From ranknet to lambdarank to lambdamart: An overview. *Learning 11* (2010), 23–581.

[5] 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.

[6] 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 *ECAI 2012-20th European Conference on Artificial Intelligence* (2012), vol. 242.

[7] 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* (2007), pp. 257–264.

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

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

[10] 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 '09 (New York, New York, USA, Nov. 2009), ACM Press, p. 621.

[11] 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* (2008), ACM, pp. 1143–1152.

[12] 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. *Advances in neural information processing systems 19* (2007), 281.

[13] 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 *Web Information Systems Engineering-WISE 2012*. Springer, 2012, pp. 526–537.

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

[15] 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), ACM, pp. 377–386.

[16] 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.

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

[18] 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 SIGIR 2007 workshop on learning to rank for information retrieval* (2007), pp. 3–10.

[19] 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), ACM, pp. 113–122.

[20] 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.

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

[22] 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.

[23] 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* (2010), ACM, pp. 1897–1900.

[24] 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.

[25] 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.

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

[27] 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* (New York, NY, USA, 2012), SIGIR '12, ACM, pp. 1127–1128.

[28] Sorokina, D., Caruana, R., and Riedewald, M. Additive groves of regression trees. In *Machine Learning: ECML 2007*. Springer, 2007, pp. 323–334.

[29] Sousa, D., Rosa, T., Martins, W., Silva, R., and Gonçalves, M. Improving On-Demand Learning to Rank through Parallelism. In *Web Information Systems Engineering - WISE 2012 SE - 38*, X. Wang, I. Cruz, A. Delis, and G. Huang, Eds., vol. 7651 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 526–537.

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

[31] 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), ACM, pp. 267–274.

[32] Wang, S., Gao, B. J., Wang, K., and Lauw, H. W. CCRank: Parallel Learning to Rank with Cooperative Coevolution. In *AAAI* (2011).

[33] 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), ACM, pp. 1083–1084.

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

[35] 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 (Abstract Only). In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2010), FPGA '10, ACM, p. 285.

[36] 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.

[37] 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 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines* (Washington, DC, USA, 2012), FCCM '12, IEEE Computer Society, pp. 97–100.

[38] 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* (2009), ACM, pp. 2061–2064.

[39] 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), ACM, pp. 334–342.

[40] 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), ACM, pp. 287–294.

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

[42] Zinkevich, M., Weimer, M., Smola, A. J., and Li, L. Parallelized Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems (NIPS)* (2010), vol. 4, p. 4.

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