Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Marek M o d r ý
Study programme:	Open Informatics
Specialisation:	Artificial Intelligence
Title of Diploma Thesis:	Learning to Rank Algorithms

Guidelines:

Design and test an algorithm for ranking URLs by relevance given a user search query. Feature vector data sets with manually assigned ranked results for training and testing will be provided.

1. Analyze the most important learning to rank algorithms adequate for the task.
2. Analyze and describe applicable precision measures for this task.
3. Analyze the training data and the feature vector signals, estimate signals discriminative power.
4. Implement the chosen algorithm.
5. Test and evaluate the designed algorithm.

Bibliography/Sources:

[1] William W. Cohen , Robert E. Schapire , Yoram Singer: Learning to order things. Journal of Artificial Intelligence Research, v.10 n.1, p.243-270, January 1999.
[2] Jun Xu , Hang Li, AdaRank: a boosting algorithm for information retrieval, Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, July 23-27, 2007, Amsterdam, The Netherlands.
[3] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, Greg Hullender: Learning to rank using gradient descent. Proceedings of the 22nd international conference on Machine learning, p.89-96, August 07-11, 2005, Bonn, Germany.
[4] Olivier Chapelle , Donald Metlzer , Ya Zhang , Pierre Grinspan: Expected reciprocal rank for graded relevance. Proceeding of the 18th ACM conference on Information and knowledge management, November 02-06, 2009, Hong Kong, China.
[5] Kalervo Järvelin and Jaana Kekäläinen: Cumulated Gain-based Evaluation of IR Techniques. ACM Transactions on Information Systems, 20:422-446, 2002.
[6] L. Page, S. Brin, R. Motwani, T. Winograd - Page, Lawrence, et al.: The PageRank citation ranking: bringing order to the web. (1999).

Diploma Thesis Supervisor: Ing. Jan Šedivý, CSc.

Valid until: the end of the winter semester of academic year 2014/2015

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, May 31, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Marek M o d r ý
Studijní program:	Otevřená informatika (magisterský)
Obor:	Umělá inteligence
Název tématu:	Algoritmy pro učení se řadit

Pokyny pro vypracování:

Navrhněte a otestujte algoritmus pro učení se řadit URL podle důležitosti na základě uživatelského dotazu. Trénovací data s ručně označenými výsledky pro trénink a testování budou poskytnuta.

1. Analyzujte nejdůležitější algoritmy pro učení se řadit, které jsou vhodné pro zadanou úlohu.
2. Analyzujte a popište vhodné metriky pro zadanou úlohu.
3. Analyzujte trénovací data a signály učícího vektoru, odhadněte diskriminativní sílu signálů.
4. Implementujte vybraný algoritmus.
5. Otestujte implementovaný algoritmus.

Seznam odborné literatury:

[1] William W. Cohen , Robert E. Schapire , Yoram Singer: Learning to order things. Journal of Artificial Intelligence Research, v.10 n.1, p.243-270, January 1999.
[2] Jun Xu , Hang Li, AdaRank: a boosting algorithm for information retrieval, Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, July 23-27, 2007, Amsterdam, The Netherlands.
[3] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, Greg Hullender: Learning to rank using gradient descent. Proceedings of the 22nd international conference on Machine learning, p.89-96, August 07-11, 2005, Bonn, Germany.
[4] Olivier Chapelle , Donald Metlzer , Ya Zhang , Pierre Grinspan: Expected reciprocal rank for graded relevance. Proceeding of the 18th ACM conference on Information and knowledge management, November 02-06, 2009, Hong Kong, China.
[5] Kalervo Järvelin and Jaana Kekäläinen: Cumulated Gain-based Evaluation of IR Techniques. ACM Transactions on Information Systems, 20:422-446, 2002.
[6] L. Page, S. Brin, R. Motwani, T. Winograd - Page, Lawrence, et al.: The PageRank citation ranking: bringing order to the web. (1999).

Vedoucí diplomové práce: Ing. Jan Šedivý, CSc.

Platnost zadání: do konce zimního semestru 2014/2015

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 31. 5. 2013

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Master's Thesis

Learning to Rank Algorithms

Bc. Marek Modrý

Supervisor: Ing. Jan Šedivý, CSc.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

Monday 12th May, 2014

Aknowledgements

Above all, I am truly grateful to my supervisor, Ing. Jan Šedivý, CSc., whose encouragement, guidance and support from the initial to the final part enabled me to fully examine the subject and write this thesis. Furthermore, I thank Seznam.cz for their cooperation and support. And lastly, I am thankful for the support from my family and especially from Matěj Eliáš.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 12. května 2014 ....

Abstract

In the recent decades and especially in the last few years, we have experienced a rapid growth of information. In the following years, the amount of data is supposed to multiply by hundreds. With growing amount of data, the need for high-quality Information Retrieval and for correct ranking of the retrieved results is rapidly increasing. Learning to Rank, as supervised machine learning methods, can help solving the issue which is present in many applications, such as web search engines, recommendation systems or misspelling corrections. This thesis provides an exhaustive listing and analysis of current state-of-the-art algorithms and it describes the necessary background for this work. Besides, it focuses on applicable performance measures and available datasets. All the hypothesis and knowledge are utilized in a thorough set of experiments. As LambdaMART was evaluated as the potentially best LTR algorithm, our own implementation of the algorithm is introduced and compared to an existing implementation. On the one hand, this thesis can server as a guide to any researcher interested in this topic and on the other it opens many new questions and issues.

Abstrakt

V nedávných desetiletích a zvláště pak v posledních letech zaznamenáváme velký nárůst informací a dat. Studie předpokládají, že v následujících několika letech se množství dat zvýší násobky sta. Vzrůstající množství dat způsobuje stále stoupající potřebu po kvalitním získávání informací a také po správném řazení dostupných získaných výsledků. Learning to Rank je metoda strojového učení s učitelem, která může nabídnout řešení těchto problémů, které se objevují v mnoha aplikacích jako internetové vyhledávače, doporučovací systémy nebo opravování překlepů. Tato diplomová práce předkládá vyčerpávající rešerši současných algoritmů a také vše doplňuje popisem základních znalostí. Mimojiné se práce zaměřuje i na vhodné metody měření kvality modelu a dostupné datové sady pro LTR. Nabyté vědomosti a předpoklady jsou zužitkovány v množství experimentů. Jelikož byl LambdaMART vyhodnocen jako potenciálně nejlepší algoritmus, zaměřili jsme se na něj v naší implementaci. Tato práce naši implementace nejdříve popíše a poté zhodnotí v porovnání s již existujícími implementacemi metod pro LTR. Tato práce jednak může sloužit jako průvodce aktuálního stavu v oblasti LTR, tak i otevírá mnoho nových otázek a poukazuje na několik nových zajímavých problémů.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the recent decades and especially in the last few years, we have experienced a rapid growth of information and knowledge and a consequent growth of the need for an efficient retrieval of relevant information. According to a study prepared by IDC[1] [22], only in 15 years, from 2005 to 2020, the digital universe will grow by a factor of 300 - from 130 exabytes to 40,000 exabytes[2] of which 33% will contain potentially valuable information.

It may not be fully obvious but *ranking* (sometimes called sorting or ordering) is tightly related to Information Retrieval. Sorting is a very natural process which seems to be simple when the set of objects to be sorted is not too large and, this is the main point, when you know *how* to sort it. Back in the ages, it was possible to manually determine the order of a set of objects. As the amount of information grew, several years ago, it would not be possible to do it manually, but it would be sufficient to find an easy key or dependency among the objects and determine what key will be used for the sorting. And finally, in the recent years, the amount of information and data is so vast, that even the dependencies and the sorting key are sometimes unknown. The only thing that left is the ability to recognize what we find being good (*relevant*) and what we consider being bad (*irrelevant*).

Issues emerge when the both aforementioned, ranking and information retrieval, is combined. When a relevant information is to be retrieved from a big amount of data and no dependencies, sorting keys nor ranking models are known. A typical example of wide-known Information Retrieval problem is search engine results retrieval and ranking in the Internet. Besides search engines, there are many other examples, such as recommendation systems, on-line advertising, misspelling correction, collaborative filtering etc., that involves ranking in a core of their solution.

To successfully solve the given problem, machine learning methods have to be applied. Learning to Rank (LTR) is a problem that can be solved by the means of supervised machine learning methods that can be effectively applied to solve the task of creating a ranking model. Given a set of queries, where each of the queries contains a list of documents and a list of relevance labels determining how relevant a document is with respect to a particular query, Learning to Rank methods construct a ranking model. The ranking model can be then used

[1]International Data Corporation

[2]1 exabyte = 1 billion gigabytes = 10^18 bytes

to score and rank a new set of documents with unknown relevance labels. The performance of the model is evaluated using a chosen performance measure.

On one hand, there is no doubt that without Learning to Rank methods, the Internet could not work as it works today and also many companies could not exist or their revenue would be much lower - without efficient recommendation systems or relevant search results in their applications. On the other hand, since LTR is a hot rapidly evolving topic, there is still a lot of space for further improvements. Although, many researchers work and publish on this topic, the best known methods are usually partially covered by secrets. The big companies, such as Microsoft, Yahoo, Google or Yandex, who are the leaders of the research do not uncover all the know-how (which is understandable). Moreover, there is a lot of various methods in LTR, but to the best of our knowledge, there is no work thoroughly comparing available means and algorithms. Most papers usually compare new algorithms only with the old ones and therefore a comparative analysis is missing.

The subject of this thesis is Learning to Rank algorithms. The work aims on performing a thorough analysis of Learning to Rank topic. It involves the following points:

- Introduction of general framework for Learning to Rank problem in Information Retrieval
- Thorough comparative analysis of available LTR algorithms, their categorization and explanation of the differences among the categories
- Analysis of applicable performance measures adequate for the task and a description of possible approaches to the optimization
- Discovering available datasets and performing their analysis and comparison, providing also the statistics of the datasets. Also giving an explanation of feature vectors and how it the discriminative power can be determined
- Own implementation of an algorithm
- Experimenting using different algorithms, measures and datasets and its evaluation.
- Proposal of potential improvements

Apart from publicly accessible algorithms and datasets, we were also provided by a dataset and an algorithm from Seznam.cz company which allows us to extend our experiments and analysis with the comparison to commercially used means.

The work is structured as follows. Chapter 2.1 describes the background of the task. First, it explains what is Information Retrieval and why it is important and then it provides an abstract description of Learning to Rank. This chapter also involves the best known application of LTR and then a brief description of the company Seznam.cz that co-operated with us by providing a dataset and an algorithm which will be both described in the following chapters.

Chapter 3 focuses on Learning to Rank, especially from the theoretic point of view. Sec. 3.1 introduces a general framework for LTR task, Sec. 3.2 will list and describe available datasets, Sec. 3.3 will compare and examine applicable performance measures and finally, a thorough description of state-of-the-art algorithms for LTR will be provided in Section 3.4.

Chapter 4 proposes our own implementation of LambdaMART algorithm that was considered being the potentially best algorithm. The implementation is compared to the existing implementation and the advantages of our implementation are introduced.

The knowledge obtained in the previous chapters will be verified in Chapter 5, where several experiments will be presented with their results that were sometimes as expected and sometimes surprising (e.g. an influence of a noise on the performance of a ranking model).

Finally, the thesis will be concluded in Chapter 6 where the findings and the results of the work will be sumarized.

Chapter 2

Background

Information Retrieval(IR) and *Learning to Rank*(LTR) are two main topics accompanying us throughout this work. In this Chapter, an abstract description of both will be given. In the first section, you will be provided with the history, the purpose and the basic problem explanation of Information Retrieval, followed by a Section 2.2 clarifying the meaning of Learning to Rank in relation to IR. As will be explained in the following sections, Information Retrieval in this work is presented in relation to search engines. Therefore the subject of Section 2.3 will focus on Search engines and simplified schemes of their processes will be shown. Since this work was created in cooperation with *Seznam.cz* company, the final Section 2.4 will introduce *Seznam.cz* and explain the support of the company.

2.1 Information Retrieval

Information Retrieval (IR) is a process of locating and obtaining information that is needed by a user. Generally said, IR helps satisfying a desire for a relevant information resource from a collection of available information resources. [15] The activity of information retrieval involves many subprocesses - it starts when the user identifies his need for information, then it involves searching and locating of an information resource and it ends when the information is retrieved and delivered to the user in a demanded form and the information need of the user is eventually satisfied.

Expectedly, *Information Retrieval* as such is a very old problem. The first notes about IR could be found even in 2000 B.C. when the Sumerians created a literary catalogue to list all their current literature. The first book indexes[1] appeared in a primitive form in the 16th century. Later on, in the 18th century, the book indexes became similar to today's form. Obviously, the first Information Retrieval 'systems' facilitated the search for information contained in books. IR techniques were applied mainly in libraries (and similar institutions). [18]

With the modern era of the printing (and eventually the era of digital technologies), there is a rapid increase of amount of information and data being stored. In 1944, Fremont Rider calculated that libraries will double its capacity every 16 years. [37] And note that the

[1]a list of useful phrases in the book

calculation is rather old and that the speed of growth is probably much higher nowadays. The vast amount of information resources, which is still increasing, makes it also really difficult to find the resource that fully satisfies your need. Therefore, there was a demand for a solution to the problem of the endless growth of resources. And the solution was Automated Information Retrieval systems.

To point out the growth of the amount of digital data, a few interesting facts are provided. According to a study prepared by IDC[2] [22]:

- From 2005 to 2020, the digital universe will grow by a factor of 300 - from 130 exabytes to 40,000 exabytes.

- IDC estimates that by 2020, as much as 33% of the digital universe will contain information that might be valuable if analyzed (compared with 25% today).

Automated IR systems were originally developed in the 1940s, in order to manage the scientific literature that was produced during the past decades and to reduce what was called 'information retrieval'. The systems were mainly utilized by libraries and universities. [39] However, with the rise of computer science automated IR has become its important subfield. Nowadays, the modern IR deals with data storage, analysis and retrieval of documents, algorithms etc. [18]

The modern IR works as follows. An information need of the user is defined by a query which is entered to the system (e.g. a search query in a web search engine). Since the query is not a unique identifier of any of the resources, it is likely that the query would match more than only one resource. Moreover, each of the matched resources can show different degree of relevancy with respect to the given query. To find relevant resources and to retrieve them in a convenient order is the purpose of an IR model (or of an IR algorithm). In other words, given a query and a set of documents with different levels of relevance, find the appropriate ranking of the given candidates according to their relatedness. The model that ranks the documents is one of the essential parts of automated IR system and Learning to Rank is one of the fields that provides techniques and algorithms that are capable of computing of such a ranking model.

Generally, IR models can be divided into a few categories. Set-theoretic models, algebraic models, probabilistic models and feature-based retrieval models. The category of feature-based retrieval models is also the case of models learned by Learning to Rank algorithms.

Shortly, most IR systems based on feature-based retrieval models usually evaluate documents (or information resources) with a score specifying a relevancy of the documents to a given query. Then the documents are ranked according to the values of the score. The user is then provided only with the top scored documents. The process can be also reiterated, e.g. when the user decides to refine the query.

The interconnection between IR and LTR will be further clarified in the following section which focuses on Learning to Rank.

[2]International Data Corporation

2.2 Learning to Rank

At it was noted in the previous section, Learning to Rank (LTR) is one of the methods that can be effectively applied to solve the task of creating a ranking model in Information Retrieval. It helps solving IR problems such as document retrieval, collaborative filtering, sentiment analysis, computational advertising etc. LTR method aims at learning a model that given a query and a set of candidate documents finds the appropriate ranking of documents according to their relevancy.

Nowadays, Learning to Rank (LTR) problem, and Information Retrieval in general, are one of the hottest topics in the fields of Machine Learning and Computer Science. The high interest emerged in the recent years especially because we are experiencing a rapid growth of the usage of digital technologies and the Internet. The amount of available data and information is growing, as well as the desire to efficiently use the collected data to create a new product, to increase the satisfaction of users and eventually to utilize data to increase the revenue ([23] predicted 2013 worldwide IT spending to exceed $2.1 trillion).

The problem is to rank any set of items according to its relevance (or any other ranking measure), therefore it can be used to order results of a web search engine, a recommendation system or an online advertising system. Above mentioned is the proof of the importance and the potential of the research in the fields of Learning to Rank and other problems related to Information Retrieval.

Learning to Rank is a supervised machine learning method. Given a training dataset of queries, documents and evaluations of how relevant the documents are, a LTR algorithm constructs a ranking model. The ranking model is then usually used to assign ranking scores to a new set of documents with unknown relevance. The ranking scores are finally used to order the given documents. The evaluation of the model's performance can be accomplished by a chosen performance measure.

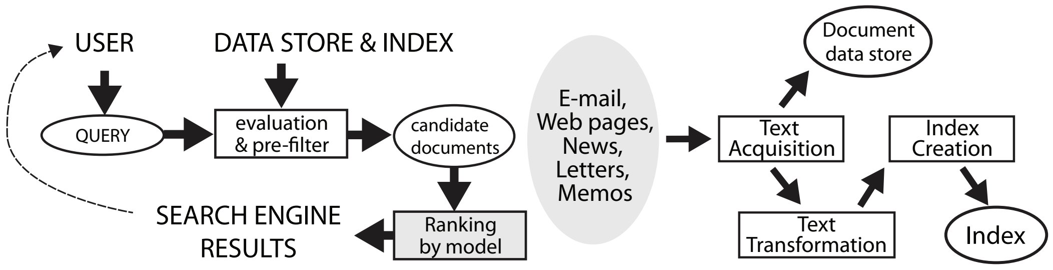
Detailed information on the architecture of Learning to rank methods, the problem definition, the formal framework and descriptions of data, algorithms and performance measures will be given in a self-standing Chapter 3. In the next section, the position and the usage of ranking models in the real-world problems, such as the web searching, will be pointed out.

2.3 Search Engines

As the web searching is one of the essential Internet activities which is known to all Internet users and it is the case which is used in the research literature most often, we decided to focus on Learning to Rank in the context of Information Retrieval in the web search. We believe that all the results of the analysis and the experiments which were performed (in the following chapters) are also applicable in other fields using Information Retrieval.

In order to point out the purpose and the position of Information Retrieval and especially of Learning to Rank, in search engines, a brief simplified description of a search engine is provided.

A search engine is a complex system composed of many components and processes. [40] One process is the indexing and crawling of Internet resources. Crawlers (sometimes called spiders) search throughout the Internet and analyze and store documents, web pages, pictures and other medias they come across. When the data, that were gathered by crawlers, are analyzed, stored and indexed, a part of the data (previously annotated with relevance labels) is utilized for the creation of Learning to Rank model. The second important process is an interaction with a user. The user enters a search query which is first analyzed and evaluated by the system, e.g. misspelling correction is suggested to the user. Then the system retrieves a set of candidate documents relevant to the query. The LTR model evaluates and ranks the candidate documents by relevance scores and then the final search engine result page (SERP) with the documents order by the relevance is returned. A simplified schemes of both processes are given in Fig. 2.1. While Fig. 2.1a demonstrates the process of user interaction and emphasizes the position of LTR model, Fig. 2.1b presents the indexing process.[40]



(a) Scheme of querying process (b) Scheme of indexing process (source: [40])

Figure 2.1: Search engine system schemes

To apply Learning to Rank methods to a web search, is not as simple as it might seem to be. Besides standard machine learning issues as dataset handling, parameter search etc., there are some issues specific for the web search and the users' behavior. Some of the issues will be presented in the following sections. An algorithm's 'quality' evaluation is one of the issues. This is the issue of the choice of appropriate performance measure. It is challenging to compare and to evaluate various orderings (permutations) of a list of documents - for example, the decision whether the ranking with an irrelevant document at the third position is better than the ranking with two irrelevant document at the 9th and the 10th position. Then it depends on the approach and the particular usage.

Also the user's need and the reason for searching can be very diverse. The user's need can be classified as informational, navigational, and transactional. The first, informational query, may range from simple factoid questions such as 'What is the color of a frog?' to complex ones. The navigational query can help you to find particular web pages on the web. And finally transactional queries can represent a user's wish to accomplish some action on the web, such as the 'Tax Form download'.[3] While dealing with Learning to Rank methods, it is important to keep the aforementioned issues in mind.

The last section of this chapter is dedicated to Seznam.cz company. Section 2.4 contains an introduction of the company and also the ways how they supported this work are mentioned.

2.4 Seznam.cz

There are several publicly available datasets and there are also benchmark results provided for the datasets. But we can find a few issues related to the publicly available datasets. There are datasets that are based on real data but the feature vectors were recalculated according to feature sets that are commonly used in the research community, or there are datasets that were created just for the research purposes, or there are datasets that are rather old (i.e. 10 years and more). In all the cases, the datasets are usually far from the current reality.

The other issue is that only the algorithms that are known to the research community are benchmarked and their results published. It is difficult (or almost impossible) to compare the state-of-the-art algorithms (known in the research) to the algorithms that are used in real search engines and find out how far the research is from the reality.

To overcome the aforementioned issues, we got a support from Seznam.cz company. The main thing is that they provided us with their dataset, their algorithm and the basic explanation of how their algorithm works. Thanks to their support, it was possible to compare state-of-the-art algorithms to their RC-Rank algorithm and to analyze the performance of the algorithms on a real-world dataset.

Seznam.cz is a Czech company running a web portal *www.seznam.cz*. The web portal has over 20 servers of different focus - a search engine, community servers, maps, news, on-line TV, on-line catalogues, advertising, games, dating etc. Since the company was founded in 1996, they have provided a full-text search service which is the most interesting part of their business because of the nature of the topic of this work.

Seznam.cz is also one of a few search engines in the world which are adequate regional competitors for Google Search engine[3]. There was an analysis performed[4] which analyzed Google Search[5] and Seznam.cz search engine. The analysis confirmed the aforementioned. The usage of both engines is almost equal. Google Search held a slightly bigger share on the market though. Moreover, there was an increasing trend of Google's share. But please note, that both analysis are at least 2 years old and the shares could have changed and that the analysis concerned only Search engines and did not reflect any other services.

The next chapter concentrates on Learning to Rank in detail. Many issues related to LTR will be addressed. Chapter 3 includes sections about a general framework for Learning to Rank, LTR algorithms description and also commonly used LTR performance measures and publicly available datasets will be listed.

[3]According to an older article at [30], besides Seznam.cz, those are Baidu (China), Naver (South Korea), Yahoo Japan (Japan) and Yandex (Russia).

[4]The analysis was published by Effectix (http://www.effectix.com) at the beginning of 2013.[17]

[5]http://search.google.com

Chapter 3

Learning to Rank

This chapter provides detailed information on Learning to Rank. First, in Section 3.1, the general framework for LTR will be provided. The next section will offer description of the data format and provide a list of available datasets and the statistics. Section 3.3 examines the ways how models' performance can be compared and measured. The basic performance measures will be named and the main characteristics will be pointed out and analyzed. The last section of this chapter finally categorizes and lists LTR algorithms.

3.1 General Framework

Learning to rank process can be described as follows. A training sample is typically a query-document pair. The necessity for using the query-document pairs as the training samples comes from the fact, that many features are based on the relation between query and document. Such *query-related* feature could be, for example, whether the document contains the query expression. The training sample consists of feature values and a relevance label. The relevance labels have been manually assessed by expert annotators. Training samples with relevance labels (with respect to a given query), a particular evaluation measure and eventually a validation dataset come as an input to the LTR algorithm. The algorithm uses the training dataset to construct a model which is then used to sort a set of testing samples and the ranking performance of the model is then evaluated by given performance measure. The aim of the learning is generally minimization of a loss function, or eventually maximization of a training performance measure. The formal description of the process is proposed in the following paragraphs and the description of the data is given in Section 3.2.

3.1.1 Problem description

In the training phase of LTR, a set of queries $Q = \{q_1, q_2, \dots, q_n\}$, where n denotes the number of queries, is given. There is a set of documents $\mathbf{d_i} = \{d_1^i, d_2^i, \dots, d_{m(q_i)}^i\}$ associated with each of the queries q_i. Then there is a list of labels $\mathbf{y_i} = \{y_1^i, y_2^i, \dots, y_{m(q_i)}^i\}$ provided together with the documents $\mathbf{d_i}$, where $m(q_i)$ is the number of documents given for the query q_i. y_j^i denotes the label of the j^{th} document d_j^i of the i^{th} query q_i. A feature vector $\vec{x}_j^i \in X$

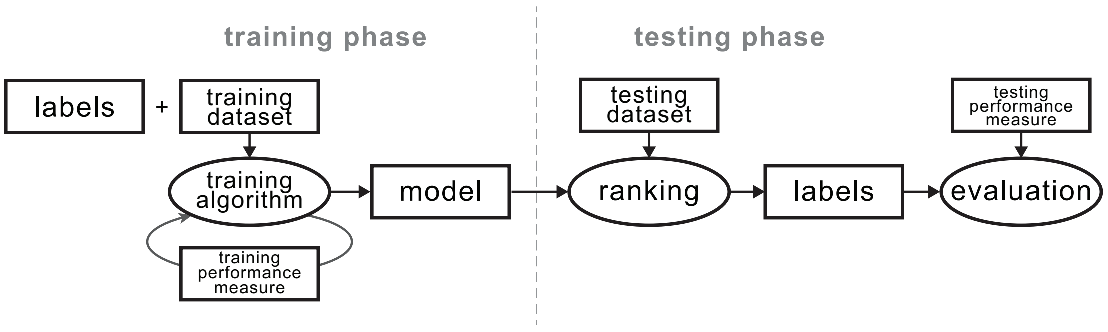


Figure 3.1: Scheme of Learning to Rank problem

is specified for each query-document pair (q_i, d^i_j), $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m(q_i)$. Finally, we can define training dataset as a set

$$S_{train} = \{(q_i, \mathbf{d_i}, \mathbf{y_i})\}_{i=1}^{n} . \tag{3.1}$$

The objective of the learning process is to construct a model optimizing the given objective function. The objective function can differ, depending on the particular approach. The measures and the approaches will be described in detail in Sec. 3.3 and Sec. 3.4. Typically, the model is then a function mapping a training sample to a score value, i.e. $f : X \to \mathbb{R}$. When using the model for ranking (e.g. in testing phase) all the query-document pairs (q_i, d^i_j) in the list of documents of query q_i are evaluated and have their score assigned. The list is then ordered according to the score in the descending order.

A permutation $\pi_i(\mathbf{d_i}, f)$ of integers is created based on a document list $\mathbf{d_i}$ and a model function $f(\vec{x}^i_j) \in \mathbb{R}$. The permutation represents a ranked list in this case. The notation $\pi_i(j)$ denotes the position of the document d^i_j in the ordering based on the specified permutation.

There are three inputs to the testing phase of the LTR problem. Those are a ranking model, a testing dataset and a performance measure. Once the ranking model (i.e. function $f(\vec{x}^i_j) \in \mathbb{R}$ in our case) has been trained, it can be used to rank elements of document lists for each single query in the testing dataset. Finally, we evaluate the performance of the model using the given measure.

See Fig. 3.1 for the basic scheme of Learning to Rank process. A brief description follows. Using a training set and relevance labels the algorithm is initiated. The algorithm runs through several iterations and optimizes given training performance measure. Once the model is constructed, it can be used for ranking of a new dataset. The labels in the testing phase are necessary only in the case, when the model's performance evaluation is desired. The labels are not necessary for the process of ranking. When the test dataset is ranked, the known relevance labels of the data samples are used, and the performance of the model is evaluated by the means of a chosen testing performance measure. Clearly, this is only a general description of the architecture of training and testing phase of LTR. Deviations can appear.

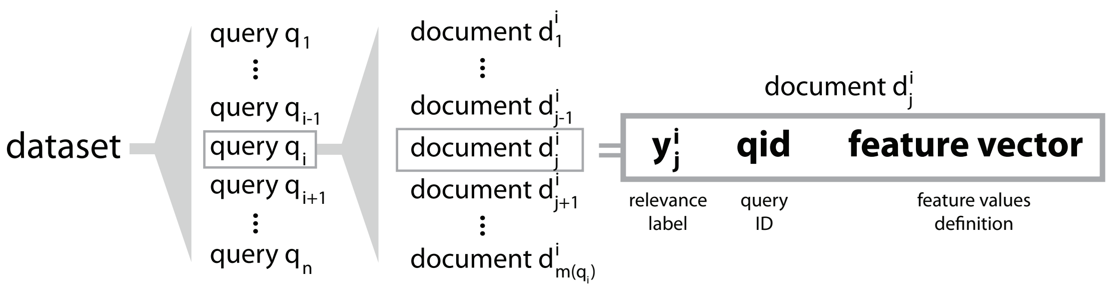


Figure 3.2: Scheme of the general data structure - Description: The dataset contains n queries. Each of the queries involves $m(q_i)$ documents in the list. Each document d^i_j is defined by a relevance label y^i_j, by a related query q_i and by a list of feature values.

3.2 Data

To properly train and test a ranking model, it is necessary to use a dataset of a sufficient quality and size. As small dataset can cause a lot of problems, e.g. increase the variance error as proven in [1]. It is quite difficult to get an access to a current and real-world dataset of a proper size, because usually only big companies can provide such datasets and they keep the datasets as their valuable treasures. However, time to time, there is a dataset release initiated by one of the big companies (e.g. Microsoft, Yahoo, Yandex) to support the research community or they provide datasets for various competitions. The available datasets will be listed and their statistics will be provided in Section 3.2.3. Fortunately, there is only one widely used format of datasets for LTR, which makes the work easier. The format will be described in the following paragraphs and is also presented in Fig. 3.2.

Each dataset consists of many query associated document lists. As a matter of fact, it means that the dataset consists of query-document pairs. Each of the document lists is associated with a different query and there is usually more than one document belonging to the document list of a particular query.

For more information and experiments analyzing the characteristics of data, see Chapter 5 on page 53.

3.2.1 Data file format

Each row in the data file corresponds to one of the query-document pairs. Each row is composed of three (optionally four) parts. In the given order, it is an integer or float relevance label y^i_j for the query-document pair (q_i, d^i_j), followed by a string in the format `qid:queryid` specifying query of the pair (q_i, d^i_j), where `queryid` denotes an integer representing the query (`qid` is a keyword that remains unchanged), e.g. `qid:1382` or `qid:913`. Then there is a sequence specifying the feature vector. Optionally, the row can be finished by a hash tag and auxiliary information (e.g. document id, various pre-calculated numbers). Although, the format is generally simple, the definition of a feature vector is a tricky part and different dataset sources use different approaches.

3.2.1.1 Labels

First, a relevance label y_j^i will be described. Since the methods, this paper is referring to, are supervised, it is necessary to provide the algorithm with labels associated with particular query-document pairs (q_i, d_j^i). The labels are usually either boolean values (i.e. zeros and ones), or multi-level graded relevance evaluations (i.e. from zero up to the maximal relevance grade, which is usually number 4). In the most cases, the labels are assessed by professional assessors according to the document relevancy with respect to the given query. In the case of boolean labels, the mapping to a word representation is simple. `Zero`, resp. `one`, represents `irrelevant`, resp. `relevant`, document. In the case of multi-level relevance assessment, the mapping can differ, but generally the word representation of the labels $y_j^i \in Y; Y = (0, 1, 2, 3, 4)$ keeps a similar meaning and can be mapped to

$$G = (useless, slightly\, relevant, relevant, useful, perfectly\, relevant).$$

However, there are also cases when the labels are slightly changed. WLC2R dataset has number 3 as a maximum relevance label, the maximum relevance label of LETOR 4.0 dataset is number 2 and the dataset provided by Seznam.cz contains number 5 as the maximum relevance grade. When the set of labels is extended to $Y = (0, 1, 2, 3, 4, 5)$ which is the case of Seznam.cz dataset then it can be mapped to

$$G = (unlabeled, off\, topic, irrelevant, relevant, useful, vital).$$

3.2.1.2 Feature vector

In the dataset file, a feature vector is specified by feature IDs and values in a given form of `fid:value` (e.g. `3:0.5621`), having all the features separated by a space character. What can be handled in a different manner is the handling of undefined values. As `0.0` value really means the value, it cannot be used for assigning *undefined* value. Usually, this issue is solved by leaving out the feature. Thus `1:0.5 3:0.5621` means that the feature with id 2 is *undefined* for the particular document.

There are two main approaches to the data definition. First, the feature definition includes all the ids and *undefined* value is substituted by a special value or zero. Therefore, all the feature vectors are of the same length. Second approach is then leaving out *undefined* values and defining feature vectors of different lengths.

3.2.1.3 File

Data are typically specified in a text file, formated in a commonly used *libSVM* (sometimes referred also as *svmlight*) format (originating in LIBSVM[1][9] and svmlight libraries[2]). The definition of a data row follows.

```
relevance qid:query_id fid:value ... fid:value # auxiliary
```

To make it clear, there is also one simple example of a few data rows (leaving out *undefined* values) provided:

[1]http://www.csie.ntu.edu.tw/ cjlin/libsvm/
[2]http://svmlight.joachims.org/

```
3 qid:19876 1:0.421 2:1.0 3:0.3123 4:11.4 # doc1
1 qid:19876 1:1.312 2:0.01 4:15.0 5:0.0 # doc2
4 qid:19876 1:0.0 2:0.31 4:7.41 # doc3
2 qid:1321 2:0.0 3:0.0 4:10.213 5:0.1 # doc4
0 qid:1321 1:0.023 2:0.413 3:0.792 4:3.45 # doc5
4 qid:1321 1:0.312 2:1.0 3:0.870 4:21.234 # doc6
```

Since all publicly accessible data are thoroughly anonymised, all the queries, document names and feature names are obviously substituted by ids without particular relation to a real object. Therefore, it is hard (or almost impossible) to search for any interconnection among documents or queries.

3.2.2 Features

As already mentioned, the datasets are usually released by the big companies because there are not many other (and maybe none) subjects who could provide a dataset of a similar size. Although, the datasets are publicly available, all additional information is anonymised. It means that we do not know expressions in queries, we do not know where documents come from or even what the labels of features are. We initially intended to analyze dataset provided by Seznam.cz and perform an analysis of features, their discriminative power and suggest possible improvements, such which other features could be used in the dataset. Finally, it was not possible to perform the analysis because the Seznam.cz dataset was anonymized as well, and the feature labels and their meanings, relations and dependencies were unknown to us. From this point of view, the analysis of features with no labels would not bring any benefit.

However, to provide a better insight how could we perform such an analysis, there are two ways how to analyze the significance of features.

- Analyze one feature after another and measure how the feature can discriminate the samples in the dataset. Using calculations of expected information gain or the possible reduction in the entropy achieved by learning the state of feature.
- To find a single well-discriminating feature is not always enough. Since there can be subsets of features with bigger discriminative power than only one particular feature can provide, it is necessary to use a more sophisticated methods. One of the possibilities is to utilize a decision tree or a whole forest that have been built above the dataset. The decision tree can be then used to analyze the features and the subsets of features and their discriminative power.

3.2.3 Available datasets

Learning to Rank is a hot topic nowadays. Therefore, there is a high demand for big datasets. However, only big corporations are able to create datasets of sufficient size. It can be tricky to find a sufficient high-quality dataset because big datasets are not usually publicly released. Usually, the datasets are released by one of the biggest corporations as Google, Microsoft, Yahoo or Yandex to support the research or their public learning to rank contests. Though, due to the security reasons the datasets are anonymised.

In the following sections, a few sources of useful datasets of higher quality will be described. Statistics, brief descriptions and comments will be provided in Table 3.1 on page 18.

3.2.3.1 LETOR

LETOR[3] is a benchmark collection for research on learning to rank for information retrieval managed by Microsoft Research group. Besides meta-data, sitemaps, link graphs etc., LETOR contains a set of datasets that can be used for Learning-to-rank task. Baseline algorithms' description, benchmark performances, features descriptions, data partitioning (train, test and validation sets) and evaluation tools are also provided. Basically, LETOR contains 2 main datasets. The first one *LETOR 3.0* was created based on the previous releases composed from *.gov* data collection and *OHSUMED* data collection. The set is altogether containing 6 datasets. The second, *LETOR 4.0* was composed from *Gov2* web page collection and two query sets from *Million Query track* of *TREC 2007* and *TREC2008.*

3.2.3.2 Microsoft Learning to Rank datasets

Two datasets, *MSLR-WEB30K* and *MSLR-WEB10K*[4], for research on LTR which were released by Microsoft. The datasets consists of query-document pairs saved in a usual data format as mentioned above. The relevance evaluations were acquired from a commercial web search engine Microsoft Bing retired labeling. Range of values goes from 0 - irrelevant to 4 - perfectly relevant. The feature vector was created on the basis of a commonly known set of features widely used in the research community.

3.2.3.3 Yahoo! Learning to Rank Challenge Datasets

In 2010, Yahoo! Labs organized a learning to rank contest. For this occasion, they also released two datasets[5]. The datasets originates from a web search ranking which was used to train a ranking function. However, no feature labelings, explanations or descriptions are provided for neither urls, queries nor features. It is due to the fact that Yahoo! was worrying about a reverse engineering. Yahoo! provided two different datasets - each corresponding to a search log from a different country. Both datasets are related, but also different to some extent. This dataset is again provided in partitioned form.

3.2.3.4 Yandex Internet Mathematics 2009 contest

Yandex.ru search engine company released a dataset[6] for the purpose of Internet Mathematics 2009 contest. The datasets contains a tables providing query-document pairs and corresponding feature vectors and assessed relevance judgements. Documents, original queries and labelings of the features are not known for obvious security reasons. However, Yandex claimed that there are TF*IDF, PageRank or query length features involved. The data are said to be real, as they were used to learn the ranking function in Yandex.

[3]http://research.microsoft.com/en-us/um/beijing/projects/letor//
[4]http://research.microsoft.com/en-us/projects/mslr/
[5]http://webscope.sandbox.yahoo.com/
[6]http://imat2009.yandex.ru/en/datasets

3.2.3.5 WCL2R

WCL2R[7] is a benchmark collection for LTR to be used preferably for research with Clickthrough Data. Chilean Learning to rank dataset created based on TodoCL search engine. Data were crawled in 2003 and 2004. Differently from LETOR or Yahoo dataset, WCL2R is mainly based on clickthrough features. The description of all the features is provided - especially of those synthesized from clickthrough data.

3.2.3.6 Seznam.cz dataset

For the purpose of this work, we were provided with a dataset[8] created by Seznam.cz company. The dataset was collected using their commercial web search engine. Seznam's relevance judgements are mapped from $Y = (0, 1, 2, 3, 4, 5)$ to

$$G = (unlabeled, off\,topic, irrelevant, relevant, useful, vital).$$

Although, it means that there is one more grade defined in comparison to other datasets (i.e. $y = 0$ which is mapped to $g = unlabeled$), there is no document marked as *unlabeled* in this particular dataset. There is no *undefined* value in the feature vectors, either. Therefore 0.0 means zero, not *undefined*.

3.2.4 Dataset statistics

Previously listed datasets were analyzed and compared in our experiments. The statistics are provided in Tab. 3.1 (page 18). The table consists of several columns. The description of the columns follows:

- `Dataset` Name of the dataset
- `Subpart` Name of the subpart of the dataset
- `Query` Number of queries
- `Docs` Number of query-document pairs
- `Feat` Number of unique features
- `AvgD/Q` Average number of documents per query
- `ModusD/Q` Most frequent number of docs per query
- `MinD/Q` Minimal number of docs per query
- `MaxD/Q` Maximal number of docs per query
- `AvgR(norm)` Mean relevance grade (normalized to $< 0; 1 >$ range)
- `MaxR` Maximal relevance grade
- `Undef` *Yes*, if the dataset contains undefined feature values, otherwise *No*

[7]http://www.latin.dcc.ufmg.br/collections/wcl2r/

[8]Not publicly accessible

Dataset	Subpart	Query	Docs	Feat	AvgD/Q	ModusD/Q	MinD/Q	MaxD/Q	AvgR(norm)	MaxR	Undef
LETOR 4.0	MQ2007	1692	69623	46	41.15	40	6	147	0.1569	2	No
LETOR 4.0	MQ2008	784	15211	46	19.40	8	5	121	0.1270	2	No
LETOR 3.0	OHSUMED	106	16140	45	152.26	153	35	320	0.2196	2	No
LETOR 3.0	Gov03td	50	49058	64	981.16	1000	525	1000	0.0083	1	No
LETOR 3.0	Gov03np	150	148657	64	991.05	1000	7	1000	0.0010	1	No
LETOR 3.0	Gov03hp	150	147606	64	984.04	1000	112	1000	0.0012	1	No
LETOR 3.0	Gov04td	75	74146	64	988.61	1000	146	1000	0.0151	1	No
LETOR 3.0	Gov04np	75	73834	64	984.45	1000	181	1000	0.0010	1	No
LETOR 3.0	Gov04hp	75	74409	64	992.12	1000	409	1000	0.0010	1	No
Yahoo	Yahoo1	29921	709877	519	23.73	9	1	139	0.3087	4	Yes
Yahoo	Yahoo2	6330	172870	595	27.31	20	1	120	0.2829	4	Yes
MSLR	MSLR10k	10000	1200192	136	120.02	91	1	908	0.1676	4	No
WCL2R	–	79	5200	29	65.82	72	19	82	0.1625	3	No
Yandex IMAT	–	21701	97290	245	4.48	1	1	31907	0.2601	4	Yes
Seznam	–	20533	567689	106	27.68	18	1	217	0.492	5	No

Table 3.1: Dataset statistics

3.3 Performance Evaluation

Similarly to other machine learning problems, it is necessary to decide how the performance of the final model will be evaluated. In many machine learning methods, the objective function which is being optimized during a learning phase is the same as the final measure evaluating the resulting model. For example, MSE[9] can be used in both cases, as an objective function during the training of a regression model and when the performance of the resulting model is being evaluated.

Unfortunately, Learning to Rank is not the same case. Since a LTR performance measure involves sorting and it is non-smooth, it cannot be differentiated and thus it is very challenging to optimize the measure directly. Only a very few algorithms actually optimize the performance measure directly. Therefore, it is important to distinguish between an objective function and a performance measure in LTR. In the following sections, performance measures will be listed, described and analyzed. While the ways how to deal with the issues concerning 'what to optimize' will be further addressed in Section 3.4 and mainly in Section 3.4.3.

3.3.1 Introduction to Performance Measures

It is important to note that the definitions of the measures are based on the data format, i.e. a set of lists of documents where a list of documents is always related to one of the queries. The measures are usually list-wise, therefore you use the measure to evaluate ranked lists one after the other and then the final performance is obtained as an average of the values returned for each list. For more information about the data format and the general framework of LTR problem, see Section 3.2 and Section 3.1.

A few assumptions, used in the following text, were made. Given a ranked list (a permutation) of documents $\pi_i = (d_1, d_2, d_3)$, d_1 is considered being the top ranked, the document d_2 is ranked as the second and d_3 is ranked as the last. Moreover, the top ranked (at 1^{st} position) document is assumed to be the most relevant document, while the bottom ranked (at 3^{rd} position) document is assumed to be the least relevant. Thus, the higher the document is ranked, the lower its index is. The length of the list π_i depends on the particular dataset and it can be arbitrarily long.

There are also cases in this work, when the list is represented only as a list of relevance labels (grades) instead of a list of documents. Therefore, $\pi_i = (d_1, d_2, d_3)$ can be also represented by $\pi_i = (y_1, y_2, y_3)$, where y_j denotes a relevance label of document d_j. Please note, that the indexing of lists usually starts from 1 and not from 0. Also the expressions performance measure and evaluation measure can be used interchangeably with no change of the meaning. Actually, there are research papers ([27, 41]) using the term *metric* which we find misleading, as the mathematical definition of *metric* is referring to a function of distance and the reader could be confused.

In the next sections, commonly used performance measures will be first listed and described and then analyzed. Finally, a short summary will be provided.

[9]Mean Squared Error

3.3.2 Performance Measures

In this section, commonly used performance measures will be listed and the basic interaction models that are considered by some of the measures will be provided in the following paragraphs (as divided by [11]). Besides other measures, we can name MAP(Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) among the most known measures. In the recent years, a new state-of-the-art measure has come up - Expected Reciprocal Rank (ERR), as well as p-found which is a measure used by *Yandex* web search engine[10].

Performance measures can be basically divided into two main categories. The first category expect binary relevance labels (grades), i.e. the documents can be labeled only `relevant` or `irrelevant`. Measures in the second category can handle multi-level relevance labels (see Sec. 3.2.1.1 for more information on labels). For example, apart from MAP or WTA measures which are the members of the first category, NDCG and ERR from the second category can be used in the cases when the labels are graded by more relevance levels.

3.3.2.1 Basic User Interaction Models

The measures, that are used by the research community most frequently, are related to one of the following user interaction models. In order to properly evaluate the performance of the model, a few assumptions on a user's behavior has to be made. One of the issues is the approach to the users' perception of the importance of the documents on different positions of a ranked list. The questions can be put: 'How important is it, having a relevant document as the top ranked element of the list?' and 'How will change the performance when the relevant document moves from the top position to the second position?' Those questions try to be answered by the following performance measures and the following approaches.

Position model Position-based model assumes that user interacts (clicks) with the document (URL) in the list under two conditions: first, it is *relevant* and second, it is examined, where the examination probability is dependent only on the position on the document in the ranked list (it is not influenced by any other document in the list). It means that it is more likely that the first document in the list will be clicked than the eleventh document because the probability of examination is much lower at 11th position. The position model is implemented, for example, by NDCG or MAP measures.

Cascade model The cascade model is an extension of the *position* model. Apart from *position* model, the probability of interaction with the document d_i also depends on the documents that have been ranked above d_i and the relevance grades of those documents. It means that if there is a perfect match on the first position of a retrieved list of documents, it's not fully important how relevant the document on the further positions are, because the needs of the user would be satisfied by the top ranked document and the probability of further examination is rapidly decreasing. On the other hand, if there are not really

[10]Yandex is Russian web search engine. As mentioned in Sec. 2.4, it is one of the regional search engines competing with Google Search.

relevant results at the top of the list then the importance of the ranking on further positions is increasing.

According to recent research papers concerning LTR measures that are used in search engines[11], and information about Yandex's MatrixNet [24, 34, 35] , the quality of the ranking is not given only by an ordering and by a position of a document. The papers confirm the validity of cascade model. Cascade model is utilized, for example, by p-found, ERR and RBP.

3.3.2.2 Winner Takes All (WTA)

Very simple and clear evaluation measure is Winner Takes All (WTA) measure which is defined as follows.

$$WTA(f;D,Y) = \begin{cases} 1 & \text{: the top document of the list is relevant} \\ 0 & \text{: the top document of the list is irrelevant} \end{cases}$$

, where f is the ranking function, D is a set of documents and Y is a set of relevance labels corresponding to the documents in the set D. There are only two possible outcomes of WTA. Either it is 1 or 0. The value depends only on the document that is ranked as the very first document in the ranked list. If the first document is relevant, the value of WTA is 1. It is 0, otherwise.

3.3.2.3 Precision

Precision (P) is another simple measure. It is a fraction of the retrieved documents that are labeled as relevant. This measure is not often used. As in our case, it is usable only in the cut-off version, i.e. `P@k` (see Sec. 3.3.2.10). Note that this *precision* differs from *precision* and *accuracy* that are defined in statistics.

Average Precision (AP) will be skipped as it is covered by Mean Average Precision (Sec. 3.3.2.4) in our case.

3.3.2.4 Mean Average Precision (MAP)

Mean Average Precision is a next representative of measures determined for only two-level relevance grades - *irrelevant* and *relevant*. It is based on Average Precision (AP). The actual difference is that MAP is modified for the use in multiple queries problems - $MAP = \sum_i^n AP_i/n$ (AP_i is Average Precision computed for query q_i and n is the number of queries). MAP can be defined as follows. [29]

$$MAP(f;D,Y) = \frac{1}{n_{rel}} \sum_{s:y(d_{\pi_f(s)})=1} \frac{\sum_{i \leq s} I\{y(d_{\pi_f(i)}) = 1\}}{s}, \tag{3.2}$$

where n_{rel} is the number of documents labeled as *relevant*, $I\{\cdot\}$ is an indicator function (returning 1 when the condition is fulfilled, 0 otherwise), $d_{\pi_f(s)}$ denotes the document ranked by the ranking function f as s-th in the ranked list(permutation) π_f and $y(d_j)$ is the integer representation of the relevance label of document d_j.

3.3.2.5 Normalized Discounted Cumulative Gain (NDCG)

Apart from MAP, NDCG measure [26] is based on a multi-graded relevance.

$$DCG(f;D,Y) = \sum_{i=1}^{m} G(y(d_{\pi_f(i)}))disc(i), \tag{3.3}$$

where G is a increasing function called the *gain function*, *disc* is a decreasing function called the *position discount function*, and π_f is the resulting ranking list.

$$IDCG(f;D,Y) = \max_{\pi} \sum_{i=1}^{m} G(y(d_{\pi_f(i)}))disc(i), \tag{3.4}$$

$$NDCG(f;D,Y) = \frac{DCG(f;D,Y)}{IDCG(f;D,Y)}, \tag{3.5}$$

Mostly, the *gain function* G is set to $G(z) = 2^z - 1$ and *discount function disc* is set to $disc(z) = \frac{1}{\log_2(1+z)}$ if $z \leq C$, and $disc(z) = 0$ if $z > C$ (C is a fixed integer)..

3.3.2.6 Rank Biased Precision (RBP)

The measures defined above were applying only *position model*. RBP is a measure applying *cascade model*. The previous measures were missing one important part which could be very important especially for web search engines, i.e. assumptions on user behavior related to the preceding documents. RBP [33] is reflecting the user browsing behavior and their *persistence* in looking through the ranked list. It means that less persistent user will look only through limited amount of elements in the list while more persistent user will thoroughly explore almost all the list. RBP, similarly to MAP, is also designed only for binary relevance labels. RBP is defined as follows,

$$RBP(f;D,Y) = (1-p) \times \sum_{i=1}^{n} \left(y(d_{\pi_f(i)}) \times p^{i-1} \right), \tag{3.6}$$

where $y(d_{\pi_f(i)})$ is a relevance grade of the document ranked at i-th position and p is a persistence coefficient of a user.

3.3.2.7 Expected Reciprocal Rank (ERR)

ERR is a state-of-the-art performance measure proposed by [11]. Similarly to RBP, ERR is also reflecting *cascade* user behavior model which is the extension of *position* model. First, we need to define $\mathcal{R}$ as a mapping from relevance grades to probability of relevance. $\mathcal{R}$ is a subject of choice but in our case we will use the *gain function* that is usually used for DCG.

$$\mathcal{R}(g) := \frac{2^g - 1}{2^{g_{max}}}, g \in \{0, \ldots, g_{max}\} \tag{3.7}$$

$$ERR := \sum_{r=1}^{n} \frac{1}{r} \prod_{i=1}^{r-1} (1 - R_i) R_r. \tag{3.8}$$

Unfortunately, a naive way how to compute ERR has the complexity of $O(n^2)$, but as shown in [11] it can be easily adjusted to be computed in $O(n)$.

3.3.2.8 Probability of User Satisfaction (p-found)

$pFound$ [35] is a measure designed by *Yandex* company which is similar to ERR measure proposed by [11].

$$pFound = \sum_{r=1}^{n} (1 - pBreak)^{r-1} pRel_r \prod_{i=1}^{r-1} (1 - pRel_i), \tag{3.9}$$

where $pBreak$ is a probability of abandonment at each position, $pRel$ is a probability of user satisfaction at a given position and $pFound$ meaning is explained as *probability of an answer to be found*. This performance measure has not been used in any of the research works which have been examined. However, it is important to compare a measure used in the search engines industry to those being used in the research.

3.3.2.9 Seznam Rank (Seznam.cz)

The performance measure used by Seznam.cz company. From now on, it will be called *Seznam Rank* and often abbreviated as SR in the following text. The measure takes into account only the top 20 documents for each query, i.e. it is SR@20 by default. The performance score for each query is given by following equation:

$$SR = \min \left(1, \sum_{k=1}^{20} w^{pos}(k) \cdot w^{rel}(y\left(d_{\pi_f(k)}\right)) \right), \tag{3.10}$$

where $w^{pos}(k)$ is the weight given by the position k (specifying that top document are more important than bottom documents), $y\left(d_{\pi_f(k)}\right)$ is a relevance grade of the document ranked at the k^{th} position and $w^{rel}(y)$ is the weight according to the relevance grade y. The values given by (3.10) are summed over the top 20 documents and the lower value is saved - either 1 or the result of the summation. The weights w^{pos} and w^{rel} are constants provided by Seznam.cz in range $< 0.0; 1.0 >$.

SeznamRank4 (SR4) Please note, that SeznamRank is originally defined for the relevance labels in the range $< 1; 5 >$. In contrast to SR, the other measures that allow multi-level relevance grading also involve 0 relevance label. To facilitate the comparison of the results and the measures, SeznamRank4 (SR4) measure was derived. It is exactly the same measure, only the mapping of relevance labels is shifted by 1. Therefore, the relevance labels range of SR4 is $< 0; 4 >$ and the weights have been shifted accordingly.

3.3.2.10 Top K documents (@k)

There are measures (e.g. ERR, NDCG) that can be computed based only on top k elements of the ranked list. This type of setting can be marked by *@k* at the name of the measure, specyfing that measure will be computed based just on the first k elements. For example, *ERR@20* would be then

$$ERR@20 := \sum_{r=1}^{20} \frac{1}{r} \prod_{i=1}^{r-1} (1 - R_i) R_r. \tag{3.11}$$

Those cut-off @k measures can be used especially in the cases when the model is trained for a specific use. For example, when only the top three documents will be displayed - it is important to learn such a model that returns relevant documents at the top three positions and the relevance of the documents ranked below does not really matter.

The following section Measures Analysis will cover the analysis of the aforementioned performance measures.

3.3.3 Measures Analysis

In order to analyze the characteristics of listed measures, a few experiments were performed. Two sets of documents are introduced for the purpose of the experiments.

$$D_1 = \{y_1, y_2, y_3, y_4\} = \{4, 3, 2, 1\},$$

where y_i denotes a relevance label of a document d_i in a set of documents D_1. And then

$$D_2 = \{y_1, y_2, y_3, y_4, y_4\} = \{1, 1, 1, 0, 0\},$$

with same notation as the previous set of documents. All possible permutations of documents on both sets were generated. The duplicate lists of labels for D_2 were removed.

Relevance Scores Comparison All the ranked lists, that had been generated, were evaluated by appropriate performance measures. D_1 includes multi-level relevance grades, while D_2 uses only binary relevance labels. Therefore, NDCG, ERR and SR[11] measures were used to evaluate the permutations of D_1 and WTA, RBP and MAP were used for D_2 permutations. Results of the evaluation of D_1 permutations can be observed in Tab. 3.2 and results given by the measures evaluated on D_2 permutations are presented in Tab. 3.3.

Measures Correlation The following experiment is focused on the correlation of the measures. The outcomes of the measures for different lists were used to calculate correlation coefficients. The coefficients for both groups of performance measures are provided in Table 3.4. The table on the left presents the correlation coefficients computed for WTA, RBP and MAP measures, while the table on the right shows the coefficients for NDCG, ERR and SR measures.

[11]SR4 modified version of SR was used in the experiment (see Sec. 3.3.2.9 for more information.))

List	NDCG	ERR	SR4	List	NDCG	ERR	SR4
(4,3,2,1)	1.00000	0.95382	0.91935	(2,4,3,1)	0.76800	0.57621	0.41874
(4,3,1,2)	0.99351	0.95345	0.91611	(2,4,1,3)	0.74851	0.57462	0.41226
(4,2,3,1)	0.97547	0.95121	0.89586	(2,3,4,1)	0.71893	0.50850	0.36915
(4,2,1,3)	0.95598	0.94962	0.88938	(2,3,1,4)	0.67347	0.47518	0.35583
(4,1,3,2)	0.95671	0.94954	0.86913	(2,1,4,3)	0.66265	0.45613	0.31569
(4,1,2,3)	0.94372	0.94832	0.86589	(2,1,3,4)	0.63667	0.42440	0.30885
(3,4,2,1)	0.86169	0.70382	0.59559	(1,4,3,2)	0.71466	0.51204	0.23865
(3,4,1,2)	0.85519	0.70345	0.59235	(1,4,2,3)	0.70167	0.51082	0.23541
(3,2,4,1)	0.78809	0.63350	0.52251	(1,3,4,2)	0.66559	0.43392	0.18906
(3,2,1,4)	0.74262	0.60018	0.50919	(1,3,2,4)	0.62662	0.40096	0.17898
(3,1,4,2)	0.76933	0.62142	0.49578	(1,2,4,3)	0.62807	0.39363	0.16233
(3,1,2,4)	0.73036	0.58846	0.48570	(1,2,3,4)	0.60209	0.36190	0.15549

Table 3.2: Measure scores for lists consisting of multi-level relevance labels.

List	WTA	RBP	MAP
(1,1,1,0,0)	1.00000	0.87500	1.00000
(1,1,0,1,0)	1.00000	0.81250	0.91667
(1,1,0,0,1)	1.00000	0.78125	0.86667
(1,0,1,1,0)	1.00000	0.68750	0.80556
(1,0,1,0,1)	1.00000	0.65625	0.75556
(1,0,0,1,1)	1.00000	0.59375	0.70000
(0,1,1,1,0)	0.00000	0.43750	0.63889
(0,1,1,0,1)	0.00000	0.40625	0.58889
(0,1,0,1,1)	0.00000	0.34375	0.53333
(0,0,1,1,1)	0.00000	0.21875	0.47778

Table 3.3: Measure scores for lists consisting of binary relevance labels.

To further illustrate the correlations of the measures, Figure 3.3 demonstrates scores given by the measures. Please note, that the figure can be slightly confusing, as there is no natural ordering defined among the lists. On the other hand, the permutations of the labels were ordered in descending order (to create a self-ordered set) and we can assume that the scores are likely to decrease. The correlation computation is inspired by [11], where the authors used correlation to prove and compare the quality and adequacy of freshly proposed Expected Reciprocal Rank.

The correlation coefficients and the figure confirmed our expectations. WTA is clearly different from RBP and MAP, on the other hand RBP and MAP are similar to each other. MAP seems to be more linear than RBP, but it is important to keep in mind that RBP can be influenced by the persistence coefficient. The results for the measures allowing multi-level relevance grading are more interesting. Although the correlation coefficients show that NDCG and ERR are more similar to each other, the coefficients for SR are also high. The biggest difference (which can also be observed in Fig. 3.3a) is between NDCG and SR. There are two points to be noted in the figure. The first interesting point is the step between 6th and 7th lists. It is caused by the fact that the most relevant document was moved to 2nd

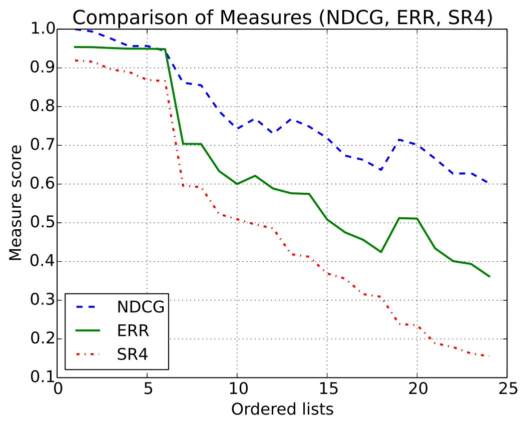
	WTA	RBP	MAP
WTA	1.00000	0.89821	0.84704
RBP	0.89821	1.00000	0.98810
MAP	0.84704	0.98810	1.00000

(a) WTA, RBP and MAP measures - binary relevance grading

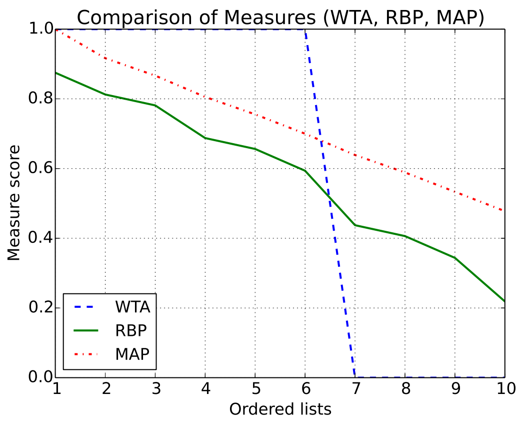
	NDCG	ERR	SR4
NDCG	1.00000	0.99063	0.97431
ERR	0.99063	1.00000	0.98545
SR4	0.97431	0.98545	1.00000

(b) NDCG, ERR and SR4 measures - multi-level relevance grading

Table 3.4: Correlation matrices of performance measures



(a) Training and testing ERR@20 for three different parameter setups. Averaged over 5 runs using 5 different folds.



(b) Learning curves of 5 runs using 5 different folds. Experiments used trees with 10 leaves.

Figure 3.3: Learning curves of LambdaMART for 3 different tree depths

position, which really influenced ERR and SR. The second interesting point is between 18th and 19th positions. NDCG and ERR reflect the fact that the most relevant document moved up, while SR measure considered other changes in the lists worse than the improvement of the position of the most relevant document.

What does influence the measures? This experiment focused on the measures allowing multi-level relevance grading, i.e. NDCG, ERR and SR. p-found is not considered as it is similar to ERR and it depends on a particular setting of the probability coefficients.

SeznamRank (Sec. 3.3.2.9) does not consider the above mentioned points that are typical for *cascade model*. It takes into account only the position and relevance of the document. Although, the metric is reflecting the relevancy of the documents in the list, as demonstrated later, it does not implicitly consider correct ordering.

To illustrate the issue, several lists of labels are introduced: $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7$. The lists are further defined in Tab. 3.5.

SeznamRank measure leads to similar results as when using other measures, e.g. NDCG or ERR. The maximum (optimized) value of SeznamRank measure is reached when the list

Lists	Relevance lists	NDCG	ERR	SR4
π_1	(4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4)	1.000000	0.968078	1.170000
π_2	(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)	1.000000	0.385664	0.550000
π_3	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	0.000000	0.000000	0.000000
π_4	(4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0)	1.000000	0.968077	1.112841
π_5	(0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4)	0.492727	0.103546	0.017379
π_6	(2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3)	0.816360	0.391034	0.626500
π_7	(2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4)	0.723180	0.391040	0.340290
π_8	(2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)	1.000000	0.367744	0.450380

Table 3.5: Comparison of performance scores of NDCG, ERR and SR

is perfectly ranked (π_1 and π_4) and the most relevant documents are on the top of the list. All the measures return the same score for π_3 and very similar for π_5. NDCG returns higher score though, which is caused by the fact that particular subparts are ordered in the correct order (e.g. the subpart of relevance grades 2). The main issue we want to point out is the difference between π_5 and π_8. Although π_8 is perfectly ranked list and π_6 is ranked in the worst possible way, ERR and SR4 prefers π_6. The consequence of this issues is that the lists are incomparable and does not reflect the fact whether the list is correctly ranked.

To sum it up, on the one hand NDCG reflects the correctness of the order, on the other hand it does not take into account the relations among documents according to relevance. Although ERR calculates with the relevances of preceding documents, it does not reflect the correctness of the order. SR4 does not calculate neither with the preceding documents nor take into account the correct order. But SR4 is the one which is easiest to be calculated and it also modifies the score according to whether the list contains the perfectly relevant document (i.e. relevance grade 4) which can be observed in the difference between π_6 and π_7.

Last but not least, it is very complicated and almost impossible to compare the performances measured on two different datasets, even though originating from the same probability distribution. The numbers and the distributions of particular relevance grades differs a lot and thus, for example, ERR@20 is totally incomparable, as the score depends on the relevance labels in the list.

A few suggestions are provided, however the ideas have not been neither verified nor tested. NDCG and ERR both went through math experiments and analysis in several research papers and such a verification would be beyond the scope of this work. It could be good idea to use a normalized version of ERR which could help to reflect the correctness of a list ordering. ERR seems to be the one corresponding to the expected users' behavior, however, the idea of p-found to use coefficients to adjust the measure could move it closer to the reality and to the real-world behavior of users.

Please, take a final note, that none of the measures are bad or good. All three of the analyzed measures put a pressure on the relevant documents to be moved up the list and thus maximize the score of the performance measure. In the ideal case, all three measures would retrieve the perfectly ranked list. However, the issue remains the same. Which of the incorrectly ranked lists do we prefer?

3.3.4 Summary

The choice of the evaluation measure is very important, because a performance measure is used to determine the quality of a ranking model. The main point of the decision should be the purpose and the use of the ranking model. A recommendation system, a question answering system or a search engine will most likely use different measures. User interaction models will be different and also the number and the structure of displayed results will play an important role when choosing the measure. Finally, also the training dataset could influence the choice of the particular algorithm and eventually the cut-off position (@k position).

It is usually clear how to select a evaluation (objective) function in regression and classification tasks. A objective (loss) function which is dependent on the distance between predicted and target values is selected in regression tasks, while in classification tasks, a objective (loss) function which is dependent on a number of misclassified samples is selected. The selection of a proper measure for LTR task is much more challenging. It is easy to determine when the documents (elements of the list) are perfectly ranked and the performance is 100% successful. However, it is not so clear, how to evaluate and compare losses caused by a few misranked documents. An example is provided to make the problem clear.

Example: Issue of ranked lists comparison Consider the following case. Let

$$S = \{(d_i, y_i)\} = \{(d_1, 0); (d_2, 0); (d_3, 1); (d_4, 1)\}$$

be the list of documents to be ranked, y_i be a relevance label of the document d_i and $y_i \in \{0, 1\}$ (0 means irrelevant, while 1 represents relevant). Now, compare two different rankings π_1 and π_2 of the list S, where the first element of the permutation is the top ranked document.

$$\pi_1 = ((d_1, 0); (d_3, 1); (d_4, 1); (d_2, 0)) \,; \pi_2 = ((d_3, 1); (d_1, 0); (d_2, 0); (d_4, 1)) \,.$$

It is difficult to compare the quality of the rankings. Both rankings can be changed to perfectly ranked lists only in two steps (i.e. moving d_1 two positions down in π_1 or moving d_4 two positions up in π_2). From this point of view, π_1 and π_2 are equally good. But what if the relevance of the top ranked document is more important than the relevance of the documents ranked at lower positions, as it happens in search engines? Then π_2 is ranked in a better way than π_1 because there is a relevant document at the top position in π_2. And what if the relevance of the first three documents is equally important while the relevance of 4^{th} document is not important (e.g. when the result of the ranking is used to recommend a relevant product in an electronic shop and only the top three positions of the ranked list are displayed)? π_1 is clearly more appropriate than π_2, because the top three elements of the ranked list π_1 contain more relevant documents.

There is no best evaluation measure. In order to properly compare the rankings and decide which one is better, it is necessary to fully understand the background and all the consequences of the problem that is being solved. Since the choice of the best model depends on the performance of the model, the choice of the evaluation measure is essential, e.g. WTA and MAP can point at different models when selecting the best model. WTA will be used in the cases when only the top ranked document is crucial for the performance of our algorithm, while for example NDCG@10 could be used in the cases when top 10 ranked documents are important or ERR and p-found could be preferred in search engines.

User behavior analysis This part of the issues is out of the scope of this work, however, it can be considered being an interesting potential direction of future research. As it was emphasized above, it is necessary to fully understand the problem and the data, in order to select the proper evaluation measure. The evaluation measure should reflect the desired ideal result. But how could it be considered what the ideal result is? Although, it is simple to specify what the perfect ranking (resp. 100% success) means, it is challenging to define the quality of imperfect rankings. As the main objective is to satisfy the user, the user should be the one who decides which of the imperfectly ordered lists is the one he prefers. The user cannot be asked to evaluate the imperfectly ordered lists but his 'opinion' could be deduced from his behavior in the system. The user behavior data can be utilized to create or to adjust one of the existing performance measures (e.g. re-weight the importance of the positions of the list).

The analysis and the use of the user behavior data in LTR is a whole research issue. It is challenging to discover a useful information in the user behavior data and handle all the issues as bias, noise or behavior significance. There are several works on the topic: [10, 16, 46, 48]. And there are also approaches using correlation between user behavior statistics and LTR evaluation measure to verify the quality of a measure [11].

3.4 Algorithms

There are plenty of algorithms available to solve Learning to Rank problems. The differences among them are not often significant because some of them differ only in loss functions (e.g. while one algorithm uses logistic loss function, the other algorithm uses hinge loss function).

First, an issue of direct optimization will be referred in this section. Then the categorization of algorithms will be explained and finally the chosen algorithms will be listed and described. As there is a lot of algorithms, often different only in small details, only the important and frequently referred algorithms have been chosen to be included in the list. Not the same amount of information is provided for all algorithms. The algorithms that are a good educative example, an important milestone in the research or a frequently used and referred algorithms have got a special attention and have been described thoroughly.

3.4.1 Direct Optimization Issue

A machine-learning method generally aims to minimize or maximize a given objective function, which usually amounts to direct minimization of a loss function.

There are several performance measures, that are commonly used in IR, e.g. MAP, NDCG, ERR (described in Sec. 3.3). Note that the evaluation measures can be easily transformed into a loss function, e.g. $L_{MAP} = 1-MAP$. Unfortunately, it is very challenging to directly optimize the evaluation measures because the measures are usually non-smooth and non-differentiable. Therefore, it is not possible to optimize a value given by the measure in a straightforward manner, e.g. by gradient descent.

The methods can be therefore divided by the approach to the optimization. There are generally two possibilities.

Direct optimization The first approach is to 'directly' optimize an original loss function. It means that an original loss function (resp. evaluation measure) is optimized by the means of upper or lower bounds, or by the optimization of the function itself which is challenging and nowadays not usually used (as the functions are non-smooth and non-differentiable). A smooth and differentiable upper-bounding (resp. lower-bounding) function is usually defined and then optimized by the algorithm. This approach is applied, for example, applied by *AdaRank*. The authors of [45] and [12] analyzed and proved the relations between the loss functions and the performance measures and pointed out many facts, e.g. that the regression based pointwise loss is an upper bound of (1-NDCG). Several important loss functions will be given below. *Lambda* algorithms, such as *LambdaRank* or *LambdaMART*, deal with the problem in an innovative way which allows to directly optimize the measure (not a bounding) and will be described in following text.

Indirect optimization The second possibility is to use a surrogate formulation of the problem and approximating the original loss function. The advantage of this approach is the smoothness and differentiability of the surrogate functions that are usually used. For example, the problem can be then defined as a classification problem on pairs of the query-document pairs or a regression problem trying to predict relevance labels of the documents. This approach is applied for example by RankBoost that minimizes a loss function based on classification error in document pairs.

3.4.2 Categorization of algorithms

One possible approach to the categorization has been already mentioned in Sec. 3.4.1. The second categorization is based on the approach to data samples.

There are three main categories of algorithms, i.e. point-wise, pair-wise and list-wise approaches. The main difference among the algorithms is the way how they approach the problem and the data samples.

- **Point-wise** approach handles the problem by transforming ranking into regression or classification of single objects. The model then takes only one sample at a time and either it predicts its relevance score or it classifies the sample into one of the relevancy classes (e.g. a class of *slightly relevant* documents).
- **Pair-wise** approach tackles the problem as a classification of object pairs, i.e. deciding whether the pair belongs to the class of the pairs with the more relevant document at the first position or vice versa.
- **List-wise** handles the problem directly, by treating a whole document list as a learning sample. For example, by using all the relations among all the documents belonging to one particular query and not only by comparing pairs or single samples.

More detailed characteristics of the approaches will be given in the dedicated sections - Sections 3.4.4, 3.4.5 and 3.4.6.

3.4.3 Loss functions

As mentioned before, the performance measures can be easily transformed into loss functions. However, the loss functions used in the algorithms are typically bounding functions of the original loss function or loss functions based on a surrogate formulation of the problem. Each of the LTR approaches (point-wise, pair-wise, list-wise) use specific loss functions appropriate to the particular approach.

Please note, that a loss function (resp. evaluation measure) used during the learning and a evaluation measure used to evaluate the performance of the model can be two completely different functions. Although there is a lot of different loss functions used by various algorithms, the performance measures described in Section 3.3.2 are commonly used for the final evaluation and the comparison of the performances of different algorithms.

To provide a better understanding what is meant by surrogate loss function, an example is provided. Let $D = d_1, \ldots, d_m$ be m documents to be ranked, then let $G = g_1, \ldots, g_K$ be K-level relevance grades and let $Y = y_1, \ldots, y_m$ be the known labels of the documents D, where $y_j \in \{0, 1, \ldots, K-1\}$. The aim of the LTR algorithm is to search for the best ranking function f, optimizing the given loss function. In this particular example, the ranking function f predicts the relevance grade of the document d_i and the algorithm minimizes mean squared error (MSE) between the predicted values and the values given by y_i, as it can be seen in (3.12). This approach belongs to the class of point-wise algorithms that define the loss function on the basis of single objects.

$$L(f; D, Y) = \sum_{i=1}^{m} (f(d_i) - y(d_i))^2. \tag{3.12}$$

More examples of surrogate functions appropriate for particular approaches will be given in the following paragraphs. Clearly, it is not possible to cover all possible loss functions but representative and important functions according to [12] will be mentioned.

3.4.3.1 Loss functions for point-wise methods

The best example of loss function for point-wise methods has already been used in the previous text in (3.12). Classification error could be another example of the loss function for point-wise methods.

$$L(f; D, Y) = \frac{\sum_{i=1}^{m} (I\{f(d_i) = y(d_i)\})}{m}, \tag{3.13}$$

where $I\{\cdot\}$ denotes an indicator function.

3.4.3.2 Loss functions for pair-wise methods

In this case, the problem is approached on the basis of pairs of objects. For example, RankBoost or RankNet use this approach. The general form of the loss function is following:

$$L(f; D, Y) = \sum_{s=1}^{m-1} \sum_{i=1, y(d_i) < y(d_s)}^{m} \phi(f(d_s) - f(d_i)), \tag{3.14}$$

where ϕ represent one of the following functions which are used by different algorithms.

- **Hinge function** $\phi(z) = (1 - z)_+$ (used by *RankingSVM*)
- **Exponential function** $\phi(z) = \exp^{-z}$ (used by *RankBoost*)
- **Logistic function** $\phi(z) = \log(1 + \exp^{-z})$ (used by *RankNet*)

3.4.3.3 Loss functions for list-wise methods

In the list-wise approach, the loss function is defined on all the samples of the document list. The problem can be formalized as a classification problem on permutations (used by *ListMLE* in [43]). There are then three different functions on the basis of permutations:

- **Likelihood loss**

  $$L(f; D, Y) = \sum_{s=1}^{n-1} \left(-f(d_{\pi(s)}) + \ln \left(\sum_{i=s}^{n} \exp(f(d_{\pi(i)})) \right) \right),$$

  where $d_{\pi(i)})$ gives a document ranked at i-th position in the ranked list π.

- **Cosine loss**

  $$L(f; D, Y) = \frac{1}{2} \left(1 - \frac{g(\mathbf{d})^T \cdot f(\mathbf{d})}{\|g(\mathbf{d})^T\| \cdot \|f(\mathbf{d})\|} \right),$$

  where $g(\mathbf{d})$ is a score vector of the ground truth and $f(\mathbf{d})$ denotes a score vector evaluated by the ranking function f on the vector of documents $\mathbf{d}$.

- **Cross entropy loss** As the definition is more complicated than the previous definitions, for the definition of cross entropy loss function on the permutations in LTR problem, we refer to [43] and [8]. Cross entropy loss is used, for example, in ListNet.

A list of algorithms and their description will be provided in the following sections.

3.4.4 Point-wise approach

Methods applying the point-wise approach handle ranking as a regression or classification of single objects. The advantage of this approach is that regression and classification belongs to well-known and well-explored methods and there is a lot of background theories available. On the other hand, it does not model the problem in a natural straightforward way which is also confirmed by the authors of [43]. Another issue is that the queries do not contribute to the loss function with the same part. The weight of a query depends on the number of documents related to the query. This can be problem especially when the counts of documents for different queries vary a lot. Moreover, such loss function cannot reflect the position of the document in the list.

3.4.4.1 PRank

PRank algorithm proposed by [13] approaches the problem similarly to RankNet (see Section 3.4.5.1) by using a neural net. They construct a ranker based on a perceptron, which maps each sample (feature vector) to a real number. The output is then $f(d_i) = \vec{w}^T \cdot \vec{x_i}$, where $\vec{w}$ is the vector of weights, i.e. the representation of the model, $\vec{x_i}$ denotes a feature vector of a document d_i and $f(\cdot)$ is a ranking function. It is also necessary to train the thresholds of the rank level intervals, as the final prediction of a relevance label is based on the intervals of ranking scores. It is a problem of ordinal regression. There is an advantage in efficiency of PRank, as it is trained on single samples and not on pairs, it uses only $O(m)$ iterations.

3.4.4.2 McRank

The authors of McRank [28] treat the problem as multiple ordinal classification. The approach is inspired by the fact that perfectly classified documents result in perfect DCG scores and the DCG errors are bounded by classification errors. The class probabilities are learned using a gradient boosting tree algorithm. The authors claim that McRank can improve LambdaRank in terms of NDCG. The authors also present instructions how to implement an efficient boosting tree algorithm.

3.4.4.3 Random Forest

Random Forest [2] is a kind of ensemble learning algorithm which combines predictions from an ensemble of random trees. Bagging is used to reduce the correlation between each pair of random trees in the ensemble. As the Random Forest method is an approach using regression, it belongs to the point-wise methods. Each of the trees in the ensemble votes for the output value. The final output is then determined by all the trees in the ensemble. As the regression is used, it is possible to use Random Forest even for multi-graded relevance ranking.

3.4.4.4 MART

MART [21] (which stands for Multiple Additive Regression Trees, or also known as Gradient boosted regression tree) is an approach utilizing a boosted tree model in which the output of the model is a linear combination of the outputs of a set of regression trees. According to [5], MART is considered a class of boosting algorithms that may be viewed as performing gradient descent in function space, using regression trees. MART can be trained to minimize any general cost (classification, regression, ranking), however, underlying model upon which MART is build is the least squares regression tree.

Since MART belongs to the family of boosting algorithms, it runs a several rounds of boosting and in each step there is a regression tree added and it's weight is determined. The final scoring (ranking) function is defined as follows (in (3.15)).

$$F_N(\vec{x_j}) = \sum_{i=1}^{N} \alpha_i f_i(\vec{x_j}), \tag{3.15}$$

where $f_i(\cdot)$ is a ranking function of a single regression tree, α_i is a weight coefficient (or *learning rate*) and $\vec{x_j}$ is obviously the feature vector of document d_j.

3.4.4.5 RC-Rank

RC-Rank is an algorithm provided by Seznam.cz. As not all the details are publicly available, we can provide only a brief basic description of RC-Rank. RC-Rank belongs to the category of methods applying point-wise approach. To the best of our knowledge, the algorithm works on the similar basis as *MART* algorithm, i.e. it is a boosting algorithm that is using the idea of multiple additive trees. However, the major difference is in the type of decision trees that are used by the algorithm. While *MART* uses *regression trees*, RC-Rank makes use of *oblivious decision trees.*

Oblivious Decision Trees *Oblivious decision tree* is a type of a decision tree similar to a *regression tree.* However, there is a restriction - all nodes at the same level of a particular tree use the same feature and the same split. Oblivious trees are usually built by a greedy algorithm. More details on oblivious trees can be found in [38].

3.4.5 Pair-wise approach

LTR methods using the pair-wise approach take document pairs as learning instances and formalize the problem as a classification task on pairs, i.e. for each pair of documents, it returns a label determining relative relevance of the pair, whether the first document should be ranked above the second one or vice versa. [8] There are advantages, such as it is possible to utilize and directly use existing methodologies on classification. It's much easier to obtain relative pair-wise preferences than whole ranked list. A disadvantage is that the number of document pairs can be different for different queries and thus the result can be biased in favor of queries with more document pairs. This is a similar issue to the one of the point-wise approach. Moreover, pair-wise approach is usually worse in efficiency than point-wise methods because it requires $O(m^2)$ iterations to consider all the pairs.

RankNet algorithm will be described more in detail as it is one of important milestones in the research of LTR.

3.4.5.1 RankNet

RankNet, proposed by [4], employs a simple probabilistic cost function (relative entropy), as a loss function and gradient descent as an algorithm to train a neural network model. They use the idea of [25] (RankSVM) to train the model on pairs. However, the trained ranking function maps to reals, since it would be computationally slow to rank items on the pair basis. It means, that document pairs are used as learning instances but then only single documents are evaluated during the ranking process. The approach can be used with many underlying algorithms. [4] used neural networks because of its flexibility ([32] claims that two layer neural network can approximate any bounded continuous function) and efficiency in a test phase (compared to kernel methods).

Let (A, B) be a pair of samples, $\bar{P}_{AB}$ a target probability of sample A being ranked higher than sample B, $o_i = f(x_i)$ and $o_{ij} = f(x_i) - f(x_j)$, the cross entropy cost (loss) function is then defined in (3.16).

$$C_{ij} = C(o_{ij}) = -\bar{P}_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}) \tag{3.16}$$

Mapping from the outputs to probabilities is provided by a logistic function (3.17).

$$P_{ij} = \frac{e^{o_{ij}}}{1 + e^{o_{ij}}} \tag{3.17}$$

And thus resulting function C_{ij} becomes

$$C_{ij} = -\bar{P}_{ij} o_{ij} + \log(1 + e^{o_{ij}}) \tag{3.18}$$

This cost function (slightly modified for the neural net purposes) is then optimized by the means of neural networks, i.e. back-propagation and forward-propagation. The ranking model is then represented by a vector of weights (parameters of a neural net) which have been learned.

[25] approaches the problem as ordinal regression, i.e. learning the mapping of an input vector to a member of an ordered set of numerical ranks (intervals on real numbers). The loss function used in their method depends on pairs of examples and their target ranks. It is complicated to find the interval thresholds, though.

3.4.5.2 RankBoost

RankBoost [20] is a boosting algorithm based on AdaBoost idea proposed by [19]. RankBoost employs pair-wise approach. It trains weak learners using distribution D_t defined over $\mathcal{X} \times \mathcal{X}$ at time stage t. The algorithm works over pairs of documents. There is a weight defined for each of the document pairs which changes after each of the iterations and the algorithm learns weak rankers that minimize the number of incorrect pairwise orderings.

As the weak learners, either feature values or thresholds are used. Using the feature values is straightforward. Using thresholds, let x be a document, $h(x)$ is 1 when a value of feature $f_i(x)$ is greater than threshold Ω and $h(x)$ is 0 otherwise. Many theoretical properties of RankBoost can be inherited from AdaBoost.

3.4.5.3 RankSVM

RankSVM (sometimes called RankingSVM) is an algorithm proposed by [25]. It is another algorithm applying pair-wise method, classifying pairs of documents and determining their relative relevance. RankSVM approaches the ranking as ordinal regression and therefore the thresholds of the classes have to be trained as well. RankSVM employs minimization of hinge loss function. It also allows direct use of kernels for non-linearity. RankSVM was one of the first algorithms with pair-wise approach to the problem.

3.4.6 List-wise approach

List-wise methods approaches the whole lists as the instances in learning. The advantage is that the approach is natural and straightforward and it utilizes all information about the documents including their position in a particular list. The disadvantage is that it is challenging and complicated to optimize a function defined on the whole list. As already mentioned before, there are two main branches. The algorithms can directly optimize one of IR performance measure (or a function correlating with the measure) or it can minimize a surrogate loss function, e.g. a probability loss function defined on permutations.

As the algorithms in this section often employ novel ideas and approaches which are different from classification or regression tasks, a few algorithms will be described in details. One of the first list-wise algorithms is *AdaRank*. Since, we find this algorithm very educative and it is a good example of the main ideas of list-wise approach in LTR, such as the bounding of performance measures, *AdaRank* will be thoroughly described even including the pseudo-algorithm.

3.4.6.1 AdaRank

AdaRank [44] is a method which employs the list-wise approach. It repeatedly constructs weak rankers which are then linearly combined to get the final ranking model. AdaRank utilizes, inspired by AdaBoost [19], boosting which means a method that is iteratively constructing weak rankers and re-weighting training samples in order to improve performance of the ensemble of the weak rankers, which if combined, can provide one strong ranker: $f(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$, where $h_t(\vec{x})$ is a weak ranker, α_t is the weight and T is the number of iterations.

It minimizes an exponential loss function that is upper bounding the basic loss function (see definition of basic loss function (3.27)). It means that it directly optimizes the given evaluation measure. Since there is also a lower bound of the performance on the training data specified, the performance of the method can be continuously improved during the training process.

As the AdaRank algorithm is easy to implement and contains many interesting parts, it will be described in detail (see Algorithm 1).

At the beginning the weights $P_1(i)$ are equal. However, during the iteration the weights are updated in order to make the queries, which were not ranked well, more important. A weak ranker $h_t(\vec{x})$ is constructed each iteration based on its weighted performance that is given by Eq. 3.19. Although, the weak rankers can be constructed in many ways, [19] uses single features as weak rankers.

$$E(h_t(\vec{x})) = \sum_{i=1}^{m} P_t(i) E(\pi(q_i, \mathbf{d_i}, h_t), \mathbf{y_i}), \tag{3.19}$$

where $h_t(\vec{x})$ is a weak ranker at iteration t, m is number of queries, $\pi(q_i, \mathbf{d_i}, h_t)$ denotes a list (permutation) π of documents $\mathbf{d_i}$ related to query q_i ranked by a weak ranker h_t. $E(\pi_i, \mathbf{y_i})$ is a performance evaluated on the basis of the ranked list π and related vector of relevance labels $\mathbf{y_i}$.

Algorithm 1 AdaRank algorithm

```
Inputs
1: Input $S = (q_i, \mathbf{d_i}, \mathbf{y_i})$ - dataset
2: Input $E$ - evaluation measure
3: Input $T$ - number of iterations
Initialization $P_1(i) = 1/m$
Iteration:
1: for $t = 1, \ldots, T$ do
2:     Create weak ranker $h_t$, with weighted distribution $\mathbf{P_t}$ on training data $S$.
3:     Choose $\alpha_t$.
           $$\alpha_t = \frac{1}{2} \cdot \ln \frac{\sum_{i=1}^{m} P_t(i)\{1 + E(\pi(q_i, \mathbf{d_i}, h_t), \mathbf{y_i})\}}{\sum_{i=1}^{m} P_t(i)\{1 - E(\pi(q_i, \mathbf{d_i}, h_t), \mathbf{y_i})\}} \quad (3.20)$$
4:     Create $f_t$.
           $$f_t(\vec{x}) = \sum_{k=1}^{t} \alpha_k h_k(\vec{x}) \quad (3.21)$$
5:     Update $\mathbf{P_{t+1}}$
           $$P_{t+1}(i) = \frac{\exp\{-E(\pi(q_i, \mathbf{d_i}, f_t), \mathbf{y_i}\}}{\sum_{j=1}^{m} \exp\{-E(\pi(q_j, \mathbf{d_j}, f_t), \mathbf{y_j}\}} \quad (3.22)$$
6: end for
```

The objective of the algorithm is to maximize the ranking performance measure on the training data (see Eq. 3.23), where E is a general performance measure (MAP, NDCG, WTA).

$$\max_{f \in \mathcal{F}} \sum_{i=1}^{m} E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i}) \tag{3.23}$$

It can be said that it is equivalent to minimization of the loss function defined as you can see in Eq. 3.24.

$$\begin{aligned} \min_{f \in \mathcal{F}} \sum_{i=1}^{m} (E(\pi^*(q_i, \mathbf{d_i}, f), \mathbf{y_i})) - E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})) = \\ = \min_{f \in \mathcal{F}} \sum_{i=1}^{m} (1 - E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})), \end{aligned} \tag{3.24}$$

where $\pi^*(q_i, \mathbf{d_i}, f)$ is a perfectly ranked permutation. However, Eq. 3.24 is not a continuous smooth function. Thus, it is necessary to find a surrogate continuous function which is approximating Eq. 3.24 well. An upper bounding function is used instead. As it holds that $e^{-x} \geq 1 - x$ we can use following upper bound.

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{m} \exp\left(-E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})\right) \tag{3.25}$$

The objective of the algorithm is then to minimize loss $L(h_t, \alpha_t)$:

$$\min_{h_t,\alpha_t} L(h_t, \alpha_t) = \sum_{i=1}^{m} \exp\left(-E(\pi(q_i, \mathbf{d_i}, f_{t-1} + \alpha_t h_t), \mathbf{y_i})\right), \tag{3.26}$$

where α_t is a positive weight of the weak ranker h_t and f_{t-1} is the ensemble of the weak rankers constructed so far. Values of α_t and $\vec{P_{t+1}}$ are chosen in order to minimize the denominator of (3.22).

According to the experiments in [44], AdaRank provides significantly better results than Ranking SVM, RankBoost and BM25. However, according to our experiments in Chapter 5, it generally performs worse than the algorithms using trees, such as MART, LambdaMART or RC-Rank.

3.4.6.2 Coordinate Ascent

Coordinate Ascent is an algorithm proposed by [31]. The algorithm applies list-wise approach and it is based on the idea that the ranking scores are calculated as weighted combinations of the feature values. The algorithm then descends through the space of weights' values and optimizes a chosen performance measure. The algorithm restarts several times at different positions of the space of weights and then is descending for a given number of iterations. Although, this algorithm returns decent results, its efficiency is rather low and therefore (according to our experiments) and therefore we consider other list-wise algorithms being better choice - as the other algorithms (i.e. LambdaMART) provides consistently better results and it is also faster to learn the model.

3.4.6.3 PermuRank

PermuRank [45] minimizes the hinge loss function which is upper bounding the basic loss function. The basic loss is defined as follows:

$$\begin{aligned} L(f) = \sum_{i=1}^{m}(E(\pi^*(q_i, \mathbf{d_i}, f), \mathbf{y_i})) - E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})) = \\ = \sum_{i=1}^{m}(1 - E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})), \end{aligned} \tag{3.27}$$

where π_i is the permutation selected for query q_i by ranking model f, π_i^* is the optimal permutation. The definition expects $E(\pi_i^*, \mathbf{y}_i)$ being equal to 1. The authors employ the SVM technique to minimize the regularized hinge loss function, i.e. to optimize the upper bounding of the basic loss in a greedy way. It is not possible to optimize the basic loss because it is non-smooth and non-differentiable and therefore it is necessary to use the upper bounding.

3.4.6.4 *SVM^{map}*

SVM^{map} [47] is an SVM based method that directly optimizes MAP measure and employs list-wise approach. It globally optimizes a hinge-loss relaxation defined on MAP information retrieval performance measure. The hinge-loss function upper bounds the basic loss function based on Average Precision. The authors showed in the experiments that SVM^{map} can outperform other conventional SVM methods. It has been proven that SVM^{map} works in polynomial time.

3.4.6.5 ListMLE

ListMLE [43] uses list-wise approach to LTR, using the whole list of documents as the learning instance. ListMLE employs likelihood loss function while RankCosine and ListNet (see Section 3.4.6.6) use cosine loss and cross entropy loss functions. Let X be a space of sets of objects to be ranked and Y be a space of resulting permutations. Then the probability distribution of $P(X, Y)$ is unknown. Therefore it is necessary to construct an empirical expected loss function (3.28) based on training samples. The final ranking function is given by a sum of weighted weak learners. Weak rankers can be chosen as

$$R_S(\mathbf{h}) = \frac{1}{m}\sum_{i=1}^{m} l(\mathbf{h}(\mathbf{x^i}), \mathbf{y^i}), \tag{3.28}$$

where $l(\mathbf{h}(\mathbf{x^i}), \mathbf{y^i})$ is a $0-1$ loss function. It equals 0 when the predicted permutation is the same as the ground truth ranking and it equals 1 otherwise. And also $\mathbf{h}(\mathbf{x^i})$ can be defined as $\mathbf{h}(\mathbf{x^i}) = sort(g(x_1^i), \ldots, g(x_{n_i}^i))$, where $g(\cdot)$ is a scoring function and m is the number of queries. However, as $sort(\cdot)$ function and $0-1$ loss function are used, the expected loss function is non-differentiable. As in other cases, the algorithm uses surrogate loss function ϕ. The objective is then to minimize new expected loss function (3.29).

$$R_S^{\phi}(\mathbf{h}) = \frac{1}{m}\sum_{i=1}^{m} \phi(\mathbf{g}(\mathbf{x^i}), \mathbf{y^i}), \tag{3.29}$$

As mentioned before, ListMLE proposes using likelihood loss function as a surrogate loss function. The likelihood loss function is defined as:

$$\begin{gathered}\phi(\mathbf{g}(\mathbf{x}), \mathbf{y}) = -\log P(\mathbf{y}|\mathbf{x}; \mathbf{g}) \\ P(\mathbf{y}|\mathbf{x}; \mathbf{g}) = \prod_{i=1}^{n} \frac{\exp\left(g(x_{y(i)})\right)}{\sum_{k=i}^{n} \exp\left(g(x_{y(k)})\right)},\end{gathered} \tag{3.30}$$

Likelihood loss is proofed by [43] to be consistent, sound, continuous, differentiable, convex and it is also possible to compute the loss efficiently in linear $O(n)$ time, where n is a number of objects. Stochastic Gradient Descent (SGD) is used for minimization. Neural Network, parameterized by ω is then used as a ranking model which is maximizing sum over all the queries:

$$\sum_{i=1}^{m} \log P(\mathbf{y^i}|\mathbf{x^i};\mathbf{g}) \tag{3.31}$$

The authors of ListMLE claim that ListMLE leads to better experimental results in comparison to RankCosine and ListNet algorithms and that it also provides better properties.

3.4.6.6 ListNet

ListNet [8] is a method applying the list-wise approach in which lists of objects are used as training instances. The paper uses two probabilistic models for listwise loss function definition - permutation probability and top k probability. [8] proposes transforming both the lists of relevance marks assessed by humans and the scores given by ranking model into probability distributions. It is possible to use almost any metric between two probability distributions. However, permutation probability and top k probability are used in the paper. The paper makes use of gradient descent algorithm and neural network model, similarly to [4]. The permutation probability is proposed in the following form:

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\Phi(s_{\pi(j)})}{\sum_{k=j}^{n} \Phi(s_{\pi(k)})}, \tag{3.32}$$

where $s_{\pi(j)}$ denotes the score of object at position j of permutation π and $\Phi(s_{\pi(j)})$ is increasing and strictly positive function. Let Ω_n be a set of all possible permutations, it holds that $P_s(\pi) > 0$ and $\sum_{\pi \in \Omega_n} P_s(\pi) = 1$. Given two permutations we can calculate the probability distributions and then use any metric between the distributions as a list-wise loss function. As there are so many possible permutations, it is intractable to calculate all the distributions and thus the second approach, top k probability, is applied.

Let $\mathcal{G}(j_1, j_2, \ldots, j_k)$ be a subgroup containing all the permutations in which the top k objects are exactly $(j_1, j_2, \ldots, j_k)$, in this order. The number of possible subgroups decreased to $\frac{n!}{(n-k)!}$. The probability of the subgroup is just a sum of the probabilities of its members. However, it is possible to further improve the efficiency of the calculations. In fact, the probability function can be defined as in (3.33).

$$P_s(\mathcal{G}(j_1, j_2, \ldots, j_k)) = \prod_{t=1}^{k} \frac{\Phi(s_{j_t})}{\sum_{l=t}^{n} \Phi(s_{j_l})} \tag{3.33}$$

ListNet then uses Cross Entropy as a metric between two probability distributions and $\Phi(s_{\pi(j)})$ is substituted by an exponential function. Gradient is then calculated based on the scores and loss function definition with respect to ω parameter. Linear Neural Network was then used for modeling. Similarly to RankNet.

According to authors, ListNet can outperforms other methods, namely Ranking SVM, RankBoost and RankNet.

3.4.6.7 RankCosine

RankCosine [36] is a coordinate descent algorithm utilizing cosine similarity between the predicted ranking and the ground truth to create a generalized additive ranking model. The authors of [36] emphasize the importance of query-level loss functions. They claim that SVM, RankBoost or RankNet do not use a suitable loss functions, as the queries do not have the same importance in the loss functions. It can be a problem in the cases when the number of documents relevant to each query varies a lot. They named what the properties of a correct loss function should be. It is: a) Insensitivity to number of document pairs, b) Importance to rank top results correctly, c) Loss function is upper bounded, i.e. loss function should not be easily biased by difficult queries with high loss. The definition of cosine loss function is defined in Sec. 3.4.3.3.

The authors employed a stage-wise greedy search strategy and used a generalized additive model as the final ranking function. The final ranking function is then determined by weighted sum of weak learners.

The experiments performed by the authors showed that RankCosine can outperform RankNet, RankBoost, RankSVM and BM25 on OHSUMED and Gov (see Section 3.2.3.1 and Table 3.1) datasets.

3.4.6.8 LambdaRank

LambdaRank [6] is a method based on RankNet algorithm (see section 3.4.5.1). It also uses neural nets. While RankNet is optimizing for a smooth, convex approximation to the number of pairwise errors and therefore using gradients derived from the defined cost, LambdaRank is rather specifying the gradients directly. That is, specifying gradient based on the ranks of the documents and not based on the information retrieval measure. The performance measure is used only for the purpose of test evaluations and as a stopping criterion on a validation set.

Lambdas (λ's) can be understood as rules defining how to change the ranks of documents in a ranked list in order to optimize the performance. Other approaches just define how to change the ranks of documents based on the performance measure (which can be a problem using certain measures, e.g. WTA). The gradients (i.e. the rules giving how to change the ranks) are used for changing weights (parameters) in neural nets. The size of the step is then specified by a value of learning rate η which is usually a small number. Performing smaller steps then the steps which would maximally reduce the cost is a form or regularization for the model that can significantly improve test accuracy, as claimed in [5]. To explain the idea of *lambdas* once again in a more simplified way. They skip the issue and calculate the gradient on already ranked lists, defining rules that could be understood as 'How should we change the ranking scores[12] of particular documents in order to improve the performance of the model?', which is different from 'How should we change the weights or the parameters of the model to make the ranking scores change in order to improve the performance of the model?'.

[12] *Ranking score* is the number which is the value returned by the model for a particular document which is then used for sorting.

3.4.6.9 LambdaMART

LambdaMART[5, 42] is a method combining MART (section 3.4.4.4) and LambdaRank (section 3.4.6.8). It uses the idea of λ's according to LambdaRank and boosted regression trees according to MART. As the authors of [5] claim, 'it is a perfect marriage' (of ideas from MART and LambdaRank). Since MART models derivatives, and since LamdaRank works by specifying the derivatives at any point during training, the two algorithms works together well.

In comparison to LambdaRank, LambdaMART is capable of decreasing the utility of some queries in order to increase the overall utility (performance). It is caused by the way how the weights are updated.

As we consider this algorithm to be the state-of-the-art algorithm with big potential, it will be described more in detail. The description will be mainly inspired by equations and algorithms from [5].

The explanation of *lambdas* was also outlined in Sec. 3.4.6.8. Since IR performance measures are non-smooth and non-differentiable it is challenging to directly optimize them. As it it cannot be differentiated, it is not possible to easily compute gradients. It is based on the idea that it is not necessary to derive the gradients from a cost directly, when we can directly specify the desired gradients. By gradients we still mean gradients of cost (with respect to the model scores) but not directly computed from the cost and rather specified by some rules. This solution skips the issue introduced by *sort* function in most IR measures by calculating the gradients after the documents were ordered. The key underlying observation is, that in order to train a model, we need only gradients and not the exact cost.

Lambda λ are considered being gradients with contributions from all other documents (relevant to a given query) that have different relevance label. If document d_1 is more relevant than document d_2 then it will be pushed by the force $\lambda_{1,2}$ upwards, while if the document d_1 is less relevant than document d_2 then d_1 will be pushed by the force $\lambda_{1,2}$ downwards.

All necessary equations are available in [5], therefore only the most important equations will be provided.

Lambda 'force' between document d_i and d_j is defined as follows:

$$\lambda_{ij} = \frac{|\Delta Z_{ij}|}{1 + e^{(s_i - s_j)}}, \tag{3.34}$$

where s_i is a ranking (model) score given to the document d_i by a ranking model, $|\Delta Z_{ij}|$ denotes the difference in a performance measure generated by swapping the rank positions of d_i and d_j, and Z is a chosen performance measure that is being optimized. The summation of all the forces pushing document d_i is then:

$$\lambda_i = \sum_{j:\{i,j\} \in I} \lambda_{ij} - \sum_{j:\{j,i\} \in I} \lambda_{ij} = \sum_{\{i,j\} \rightleftharpoons I} \lambda_{ij}, \tag{3.35}$$

where λ_{ij} is defined in (3.34) and I is a set of all pairs (x, y) where document d_x is more relevant than d_y and then we introduce simplified form of the sum equation. The equation sums up both, downward forces and upward forces.

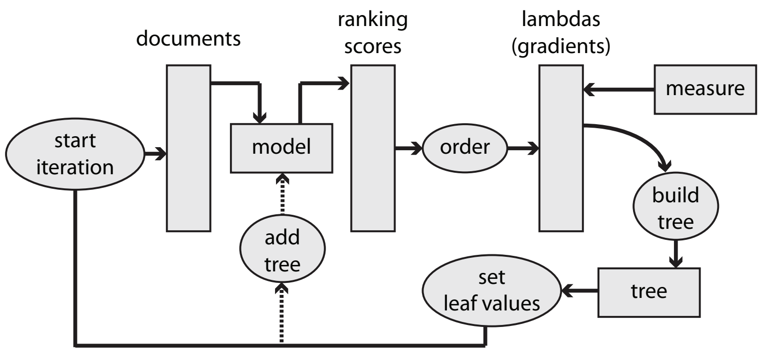


Figure 3.4: Scheme of the iteration of LambdaMART algoritmh.

Note, that

$$\frac{\partial C}{\partial s_i} = \sum_{\{i,j\} \rightleftharpoons I} \lambda_{ij}, \tag{3.36}$$

and then for Newton step calculation we have:

$$\frac{\partial^2 C}{\partial s_i^2} = \sum_{\{i,j\} \rightleftharpoons I} |\Delta Z_{ij}| \cdot \frac{-\lambda_{ij}}{|\Delta Z_{ij}|} \cdot \left(1 - \frac{-\lambda_{ij}}{|\Delta Z_{ij}|}\right). \tag{3.37}$$

Having both, (3.36) and (3.37) defined, it is then easy to determine the Newton step size for further calculations.

In very simplified way, the algorithm can be described as follows. When the ranking scores of documents are obtained from the current model, the documents are ranked according to the scores. Then based on the scores and the measure, the lambda gradients are calculated determining, how the scores of the documents should change in order to improve the overall performance. The most important part of the idea is then that the tree is built in order to predict the lambdas (i.e. changes of the scores, not the actual scores or relevance labels). The trees are gathered into an ensemble of trees. For a particular sample, the trees represent a collection of changes (gradients) to be applied to the score (which is initially zero). This description aimed to describe the algorithm in an understandable way and to point out the important parts, for detailed description with all deductions and the pseudo-algorithm, see [5].

3.4.7 Combining rankers

Let say, we have two available datasets S_1 and S_2 and two rankers F_1 and F_2. It can happen that ranker F_1 outperforms F_2 on the dataset S_1, while F_2 outperforms F_1 on the dataset S_2. Then it is hard to choose which ranker is the better one because there is a possibility that the results of the selected ranker will be worse that the results we reached during the training and testing.

This problem can be approached by combining rankers that performed well during the training and testing. The diversification caused by the combination of rankers minimizes the chance of achieving poor results because of one single ranker. Therefore, it can be suggested to linearly combine the results of selected ranking functions. The combined ranking function of L rankers is then (3.38).

$$F_L(\mathbf{x}) = \sum_{i=1}^{L} \alpha_i F_i(\mathbf{x}), \tag{3.38}$$

where $F_i(x)$ is a single ranker, x is a feature vector, and α_i is a weight coefficient of ranker $F_i(x)$.

3.5 Public Contests

First of all, please note that this section is not concerning neither our own models nor our algorithms. It is mainly a suggestion of a useful source of both, information and comparisons.

As the research on LTR is hot and rapidly evolving topic, it can be highly challenging to compare the methods based only on papers or publicly available datasets. However, there have been a few competitions involving LTR that can help comparing the approaches and methods. The competitions have asked for solutions of real-world problems and providing real-world datasets. Since there are usually tens or hundreds of competing teams, it is highly probable that the winning solutions propose high-performance and state of the art algorithms and models.

Nowadays, a lot of competitions concerning LTR are held by Kaggle[13]. Apart from Kaggle challenges, there were also competitions self-organized by Yahoo or Yandex[14] in recent years. In the following section, an interesting challenge will be briefly described and the basic idea of the winning solutions will be provided.

3.5.1 Expedia Hotel Searches

Expedia is the world's largest online travel agency (OTA). It provides a search engine for the customers to find a perfectly matching hotel. Providing the high-quality ranking is a way how to succeed in this very competitive market. Expedia provided a 3.6Gb dataset containing shopping and purchase data and also information on price competitiveness. The data are in the form of search result impressions, clicks and hotel room purchases. For each hotel, it's characteristics and location attractiveness are provided as well as user's aggregate purchase history and competitive OTA information. NDCG@38 was used as the evaluation metric. Hotels for each user query are labeled with relevance marks: 0 - the hotel was neither purchased nor clicked, 1 - the user clicked the hotel, 5 - a room at the hotel was purchased. The final rankings of the hotels should recommend hotels in order from the hotels with the highest chance of a room purchase to the rooms with the lowest probability of a room purchase. The interesting part of the training dataset is that the part of the impressions comes from search results that were randomly ordered. The randomly ordered search results were used in order to remove the bias of the impression statistics, as the results

[13]Kaggle (www.kaggle.com) is a platform for predictive modeling and analytics competitions employing crowdsourcing approach.

[14]Both, Yahoo and Yandex are search engines. Yahoo originates in USA, while Yandex is based in Russia.

Position	NDCG@38	Position	NDCG@38
1st	0.53984	6th	0.53033
2nd	0.53839	7th	0.52989
3rd	0.53366	8th	0.52925
4th	0.53102	9th	0.52924
5th	0.53069	10th	0.52787

Table 3.6: Final leaderboard of Expedia Hotel Searches challenge

at the top positions are more probable to be viewed/clicked than the results at the lower positions. There were different categories of features in the dataset: search criteria, static hotel characteristics, dynamic hotel characteristics, visitor information, competitive OTA information (availability and prices in other OTAs) and other.

The contest lasted 2 months and 337 teams participated, therefore it can be expected that the solutions of the top three teams (i.e. winners) should be of a high-quality and proofed in a real-world challenge. As noted before, NDCG@38 was used as the evaluation measure and the performance evaluations of the first ten teams is provided in Tab. 3.6.

1st place The first place was reached by Owen Zhang (NY, USA). First, he added new features as averages of numerical features, composite features[15], EXP features[16] and estimated position. Zhang used an ensemble of 26 gradient boosting machines (GBM) as his model. He was using NDCG evaluation measure and he utilized R[17] GBM implementation[18].

2nd place J. Wang and A. Kalousis coming from Switzerland were placed as the second. This team focused on models that can easily handle nonlinearity because of categorical (discrete) features. They utilized *LambdaMART* model since it can handle nonlinearity and it is computationally efficient. Similarly to the previous solution, they reasonably used *NDCG* evaluation measure. Their feature engineering was concentrated on features with monotonic utility with respect to the target variable and features that are discriminative. Another important element of their success was feature normalization in order to remove the scaling factors (e.g. prices depends on the location and the time). Their first experiments were based on linear models as linear regression or *RankSVM*. However, *LamdaMART* was proven to outperform the linear models. This solution does not use any ensembles of models.

4th place The solution of the third team was not published, thus the solution of 4th place will be described. First, the data preprocessing. They used the first quartile value to represent the missing data, they used only 10% of data, they balanced the number of positive and negative data. Second, the feature engineering - the authors found out that different features fit different models and added new composite features. Then the authors

[15]The features as difference between hotel price and recent price and order of the price within the same search are meant.

[16]Categorical features converted into numerical features.

[17]The R Project for Statistical Computing - http://www.r-project.org/

[18]http://cran.r-project.org/web/packages/gbm/gbm.pdf

experimented with many different models (using all three approaches - pointwise, pairwise, listwise). The best performance was reached by Gradient Boosting Machine, LambdaMART, *Random Forest* and *Logistic Regression*. They also experimented with Factorization Machine but it took a long time to train the model and a lot of feature engineering work was required. The authors also mentioned the use of Deep Learning. However, they found the approach not useful in this case. Their suggestions for the further investigation were interesting. They think about using Bayesian Database to deal with the missing data, try using Deep Neural Network in pairwise and listwise approach and try utilizing Representation learning.

3.5.2 Summary

Apart from Expedia Hotel contest at Kaggle, other contests were examined and it was observed that LambdaMART is the algorithm that is used by the winning teams most often and it is very often the main topic of the discussions (and boosting trees generally). It was also observed that the most of the solutions uses ensembles of models which usually performs better than a single model. To sum it up, the interesting source of state-of-the-art comparisons was introduced - high-quality contests, such as Kaggle. And according to the results of this contest and according to other contest that have been examined, it has been proven that algorithms using trees and boosting are state-of-the-art algorithms which deserves attention.

More information about LambdaMART was provided in Section 3.4.6.9 and for more details about ranker combining see Section 3.4.7.

Chapter 4

Implementation

When we were preparing the assignment of this thesis, RankLib[1] library with the implementation of many algorithms was not known to us. The aim of our own implementation was initially to provide us with a tool which could be used for the experiments and for the analysis of datasets and measures and besides, it was meant to help us to fully understand the main ideas of LTR and of a chosen algorithm.

However, during the work on the thesis, we fortunately found out that there is an existing implementation called RankLib, containing more implemented algorithms than we could have hoped. Moreover, it provides the implementation of the basic performance measures, such as ERR or NDCG. Above all, RankLib was obviously developed by an skillful programmer and experienced researcher, as the implementations of particular algorithms and measures include many different mathematical 'tricks' increasing efficiency of the computations that have been published, such as the efficient way how to calculate the matrix of changes in ERR performance measure for the purpose of LambdaMART algorithm. Moreover, the implementation itself provides very efficient code as the library pre-computes and re-uses as many values as possible. There are also methods in RankLib to handle the splitting of datasets and of validation sets (this functionality was not used almost at all) and for each of the algorithms, there were many parameters that could be modified.

Under the new circumstances, the importance of our own implementation decreased and at the same time, it enabled us to perform so many different experiments. In spite of this, we still found the implementation of our own algorithm being important. Own implementation of an algorithm provides better insight and also a useful tool for further visualizations, prototyping etc.

In the following paragraphs, our own implementation of LambdaMART will be described and then a comparison to RankLib implementation will be presented.

4.1 Own implementation of LambdaMART algorithm

LambdaMART is an algorithm with a lot of successes in contests like Kaggle.com (see Sec. 3.5). Besides, it is an algorithm that provides a way how to directly optimize arbitrary IR performance measure and it belongs to the category of the list-wise methods. We

[1]A brief description of RankLib library can be found in Section 5.1.1

consider LambdaMART algorithm being state-of-the-art algorithm that worth attention, as it offers a novel methods how to solve the problem and it employs more straightforward approach than most algorithms that try to reformulate the problem and solve a surrogate task.

Now, we can also confirm that the choice was right, as LambdaMART performed well in the experiments (see Chapter 5) and together with MART algorithm usually showed one of the best results. Our implementation was based mainly on [5], [7] and [11].

4.1.1 Python and NumPy

Our implementation of LambdaMART algorithm was programmed in Python. Python is a dynamic object-oriented scripting language. There are many advantages of Python. Among others, the readability of the code is really good (one of the main advantages of Python), it is also easy to program a prototype in Python and Python is built based on C language, and therefore most native functions are very fast and efficient. Python further provides, for example, list comprehensions and generators, which can improve the efficiency of many operations that would be much slower in other languages. In short, Python was chosen for it's readability, for the efficiency of implementation and it's speed.

Moreover, NumPy module was used for many operations. NumPy provides classes and methods for computing with multi-dimensional arrays and matrices. NumPy provides a user with many functions comparable to MATLAB. Because array operations in NumPy are optimized and well-coded, it was our intention to use the advantages of this module as much as possible.

4.1.2 RankPy

RankPy, how we called the package involving the implementation of LambdaMART algorithm, consists of many scripts and tools. It provides tools for evaluation of performances, it involves a module for data processing (loading, splitting and modifying of datasets), further, there is a script converting *.csv* result files to LaTeX(*.tex*) tables. Moreover, RankPy involves a few tools for plotting and analyzing the results of experiments and also a logging support that enables us to store many values and array from the process of learning or testing and analyze (or plot) them after the experiments are finished.

LambdaMART in RankPy worked as follows. As already mentioned, our purpose was to prepare an efficient implementation of LambdaMART and NumPy was used to compute many values and also there was a set of arrays that have been precomputed and in the further iterations only updated.

First, the algorithm generates a *Base Model* scores, determining what the scores what the documents with no model are. Then the lambda gradients are computed based on the current score. When the lambda gradients are known, a regression tree is built.

In order to further improve the efficiency of the implementation, we decided to use *scikit* module that provides methods for learning regression decision trees. *scikit* is a well-known module with high-quality and good efficiency for many machine learning tasks. However, in order to make *scikit* fully useful for our purpose, it was necessary to adjust the class of Regression tree because in the next step, it was necessary to manually recalculate and set new values of leaves of the tree. Once the tree is learned, the model scores are recalculated and the training measure is evaluated.

We did compare the performance of RankPy to RankLib implementation. The comparison was a proof that RankLib is programmed in a very efficient way and that it is hard to compete the speed and efficiency of RankLib implementation. Despite the fact, that we used high-quality and efficient implementation of decision trees from *scikit* and we also used a proper and efficient way how to recalculate the performance measures during the training, RankLib was still much more efficient. As we have already mentioned, RankLib is extremely optimized implementation of LambdaMART algorithm that precalculates and re-use all possible computations. This is confirmed by the fact, that during our comparison experiments, RankLib consumed almost three times more memory.

We can provide an example which is available on the attached CD. When the performance was compared on the given datasets (which are smaller versions of MSLR10k dataset) RankLib consumed up to 700MB of RAM, while RankPy needed only 230MB. On the other hand, RankPy needed much more time to perform the experiment. The experiment was performed 3 times and the times were averaged. The training time of the model in Rankpy was 4 hours and 22 seconds, while the training time of the model in RankLib was 2 minutes and 10 seconds. Even when compared to the training of RC-Rank models, it took longer to train RC-Rank models than it took to train similar models for LambdaMART in RankLib. Moreover, the testing time of a model was 3.4 seconds in RankLib, while in RankPy, it was 2minutes and 58 seconds. As RankLib proved to be surprisingly efficient and fast, and also providing other advantages mentioned above it was therefore used for the experiments in Chapter 5.

Although the performance of RankPy implementation is not as good as the implementation of RankLib, there are still many advantages of having own implementation. We have got a very valuable insight thanks to the implementation. Thanks to Python programming language, it is also easy to extend the code in order to add an extension of the algorithm or apply a new idea and it allows us to further improve the implementation and prototype potential ideas in the future work.

Besides, during the implementation, there were a few visualization tools created and therefore RankPy implementation can be used as a subject of research on smaller datasets where efficiency is not the issue and it can provide educative visualization to further understand the rules and the dependencies and relations.

A few examples of visualizations are provided. In Fig. 4.1, we can see how the documents with different relevance labels change their positions in the ranked list during the time. Note, that there are more queries (and documents lists) and therefore the position of irrelevant document in the top half of the list is perhaps caused by the influence of a different list with higher importance.

Next, in Fig. 4.2, there are curves visualizing how the values of lambdas for each document change during the time. We can see the changes in the lambda values as there were also changes in the relevance label lists. Note, that changes in lambdas can be also caused only by changes in ranking scores which does not have to be projected to the order of the documents. It is obvious that the top relevant document is still being pushed up the list while other documents are pushed down or ownly slightly up. As this figure is not easy to

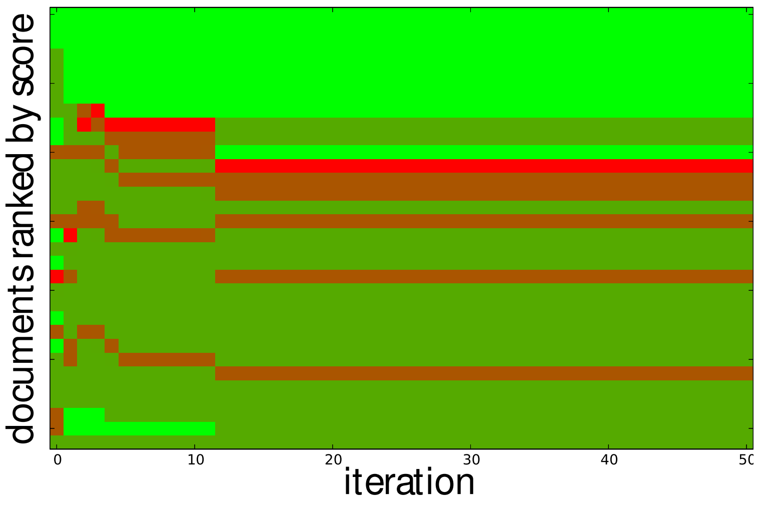


Figure 4.1: Visualization of relevance labels of a ranked list of documents evolving in time (50 iteration). The more red the color is, the less relevant the document is and vice versa.

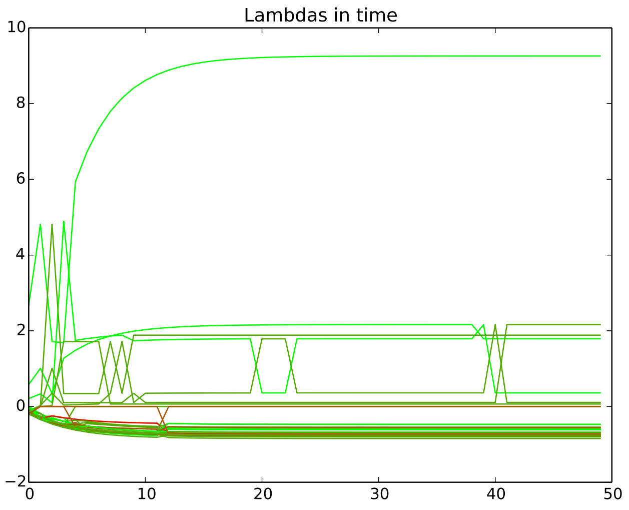


Figure 4.2: Visualization of lambdas of a list of documents evolving in time (50 iteration).

understand without a further insight and realization what is happening with the values, it is a good example of the purpose of those visualization and RankPy implementation.

The last Fig. 4.3 is very similar to the previous one. This time, it presents the values of ranking scores, which are the scores that are used as the keys to rank the documents afterwords. It can be observed that the difference among the documents is still increasing throughout the time.

It is obvious that although LambdaMART algorithm is not difficult to understand on the abstract level, it is challenging to understand all the underlying forces and rules that

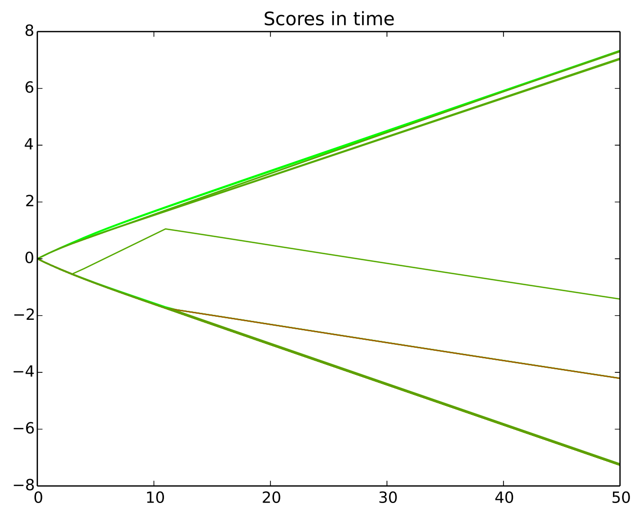


Figure 4.3: Visualization of ranking scores of a list of documents evolving in time (50 iteration).

can be applied on the values and the document lists. From this point of view, the own implementation is very valuable even though it's efficiency is worse in comparison to RankLib implementation.

Chapter 5

Experiments

This section presents several experiments mainly comparing algorithms from RankLib library to RC-Rank, analyzing the characteristics of datasets and algorithms and searching for the best parameter settings. Each of the section contains a description of an experiment with the setting and a specification which dataset, performance measure and algorithm were used. The results are commented and presented as figures or tables.

5.1 General setting

5.1.1 Implementation of LTR algorithms

In the most of the experiments, *RankLib* library was used. It supports several important LTR algorithms and the most popular performance measures. Apart from the learning algorithms, it also provides tools for testing learned models and for evaluating the models by the given performance measures. An existing LTR implementation was used because the correctness of the implementation has already been proven by the community and it would also be very complicated and in a few cases (e.g. poor performing algorithms) unproductive and inefficient to implement all the algorithms by ourselves.

Since RankLib is an open-source Java library, all the source codes are publicly available and it was possible to modify the code for our purpose. This was very helpful because it was necessary to implement, for example, the performance measure used by Seznam.cz and it was also possible to tweak the code for our purposes.

RC-Rank was provided as a binary file and its performance was evaluated by a Python script.

5.1.2 Available hardware and memory requirements

All the experiments were performed on a machine with following hardware and operating system: Ubuntu 64bit operating system, 8x Intel(R) Core(TM) i7-3770K CPU @ 3.5GHz and 32GB RAM. A few additional experiments were run on a similar machine: Ubuntu 64bit, 4x Intel(R) Core(TM) i7-3770K CPU @ 3.5GHz and 15GB RAM.

The maximal required memory during the experiments was 4GB. Most experiments used around 2GB RAM, which was average memory use in the experiments with bigger datasets. Java Virtual Machine (JVM) for RankLib library experiments used 4GB as the heap size.

5.2 Comparison of the algorithms

Since there are a lot of papers concerning LTR available, there have been a lot of different experiments carried out and documented. However, authors of the experiments use various datasets, most of which are not even public. The results are therefore often unverifiable and sometimes the results are not consistent. Even the novel algorithms are sometimes compared to old algorithms (e.g. RankBoost) and therefore there is no mutual comparison of state of the art algorithms. The comparison of the algorithms and the approaches is thus not very clear.

In order to get an overview and a proper comparison of algorithms, several experiments were carried out. The experiments aimed to compare chosen algorithms using available datasets and implementations. The results of the experiment provided a basic information about the performance of the algorithms and performances on different datasets were compared.

5.2.1 Experiment setting

Algorithms and parameters For our initial comparing and analyzing experiments Learning to Rank library called RankLib [14] was used. All the available algorithms provided by RankLib were used, i.e. LambdaMART, MART, Coordinate Ascent, Random Forest, RankNet, ListNet, AdaRank and RankBoost. There were three experiments for each of the algorithms performed (with different parameters) and the result of the best one is presented.

Besides the algorithms provided by RankLib library, we utilized also RC-Rank algorithm (see Sec. 3.4.4.5) which is an algorithm used and provided by Seznam.cz[1] RC-Rank was run with two different sets of parameters. The first parameter setup, labeled `RC-Rank baseline`, is a default setting of the algorithm. The second parameter setup is marked as `RC-Rank enhanced`. This is a parameter setup recommended by experts from Seznam.cz. The *enhanced* setting of the algorithm manipulates with the relevance labels and pre-calculates a small forest of so-called *context trees* which is then used to improve the learning.

There are also two artificial rows added to each of the result tables, `Best Ranker` and `Worst Ranker`. The rows represent what would be the performance of the best (or the worst) possible ranker on a given dataset. The performance values of both 'rankers' were obtained in the following way. A perfectly ranked permutation of documents is created for each of the queries in the given dataset. The perfectly ordered lists (permutations) are then evaluated by the chosen performance measure. The `Worst Ranker` values are calculated in a similar way. The difference is in the opposite order of the lists. Therefore, the performance scores are based on the document lists ordered in the worst possible way.

[1]For more information about Seznam.cz, see Section 2.4.

Performance measures There were three performance measures chosen for the evaluation of the algorithms' performance - i.e. NDCG@20, ERR@20 and the evaluation measure provided by Seznam.cz (Seznam Rank or SR from now on). NDCG and ERR were chosen because those are the measures that are used throughout the community most frequently. Both measures, unlike MAP, can also be used for multi-level relevance grades. SeznamRank was then chosen to see whether it correlates with the performances measured by NDCG and ERR and also because it is a measure used by Seznam.cz. SeznamRank was used only in the experiments on Seznam data and MSLR dataset, where five-level relevance grading is used.

Result table Each of the result tables consists of 5 columns - the algorithm's name, performance scores and a special averaged value called AVG_{Norm} (see below). Although, there were three runs for each algorithm, only the best performing model is recorded. In order to compare the performance of the algorithms using all three evaluation measures (NDCG@20, ERR@20 and SeznamRank), a new value called AVG_{Norm} is introduced. The value of AVG_{Norm} is calculated on the following basis. The performance scores of the models evaluated by the particular measures are first normalized, i.e. each score given by the measure is mapped to the $\langle 0;1\rangle$ range (based on maximal and minimal values of the performance measure reached by any of the models). AVG_{Norm} is then the average value of all three normalized evaluation measures' scores.

$$AVG_{Norm} = \frac{n(E_{ERR@20}) + n(E_{NDCG@20}) + n(E_{SR})}{3}, \tag{5.1}$$

where $n(\cdot)$ is the normalization function, i.e. $n(x) = \frac{x-min}{max-min}$ and E is an evaluation measure of a given type. Clearly, in the experiments where only two performance measures were used, AVG_{Norm} is averaged only over $NDCG$ and ERR.

Data In order to compare the algorithms and their performance on different datasets, there were 4 different datasets used in the experiment. Namely, LETOR 4.0, MSLR10k, WCL2R and also the dataset provided by Seznam.cz search engine[2](see Sections 3.2.3.1, 3.2.3.2, 3.2.3.5 and 3.2.3.6 respectively). For more information on the characteristics of the datasets see Table 3.1. Only the datasets with no undefined values were used.

5.2.2 Experiments' details and results

5.2.2.1 Comparison using MSLR10k dataset

MSLR10k dataset(see Sec. 3.2.3.2) created by Microsoft was used during the experiment. The original split of the dataset was kept[3]. The proportions of the splits are 60%, 20% and 20% (for the trainset, the validation set and the testset respectively), i.e. 6000, 2000 and 2000 queries present in the splits. Please note that MSLR10k was the biggest dataset which was used during the experiments.

As MSLR10k is the biggest dataset from the sets that were used, a smaller variance error can be expected as proven by [1]. The algorithms loses the tendency to overfit. From

[2]http://search.seznam.cz/

[3]Named as *Fold1* in the downloaded MSLR10k dataset files.

Algorithm	NDCG@20	ERR@20	SR4	AVG Norm
Best Ranker	0.9640	0.5896	0.6403	1.0000
RC-Rank enhanced	0.4928	0.3684	0.3271	0.5470
RC-Rank baseline	0.4945	0.3578	0.3307	0.5435
LambdaMART	0.4606	0.3581	0.3385	0.5359
MART	0.4696	0.3484	0.3312	0.5297
Coordinate Ascent	0.4402	0.3346	0.3116	0.5014
Random Forests	0.4496	0.3274	0.3058	0.4976
ListNet	0.4258	0.3238	0.2969	0.4826
AdaRank	0.4026	0.3009	0.2718	0.4485
RankBoost	0.3786	0.2402	0.2074	0.3722
RankNet	0.3597	0.2205	0.1865	0.3435
Worst Ranker	0.0091	0.0012	0.0005	0.0000

Table 5.1: Performance of various algorithms (RankLib algorithms and RC-Rank) on MSLR10k dataset. NDCG@20, ERR@20 and SeznamRank4[4] were measured. Best Ranker and Worst Ranker 'dummy' rows contain the best and the worst possible performance score.

this reason, the experiment was performed only once and the results were not averaged over several runs.

It is obvious that RC-Rank, LambdaMART and MART prevail the others. RankBoost and RankNet are significantly worse than the rest of the algorithms in this case. Interesting relation can be observed between enhanced and baseline versions of RC-Rank. RC-Rank enhanced was better only when measured by ERR@20. When measured by NDCG and SR4, RC-Rank baseline was better than enhanced version, however the difference was not so obvious to overcome the difference in ERR measure. This result shows that although the measures more or less correlates, there can be minor differences that can influence the final choice of the algorithm or the choice of parameters. From this reason, it is necessary to choose a proper performance measure.

5.2.2.2 Comparison using Seznam.cz dataset

The dataset provided by Seznam.cz(see Section 3.2.3.6) was used for the purpose of this experiments. The dataset was randomly split into three parts. Each query and related documents list was considered as a single sample when being split. Therefore all documents related to a query are always present in only one of the splits. The first part used as a training set consists from 60% of samples, the second part and the third part, each consists from 20% of samples are used a validation set and a testing set, respectively.

The experiments were performed using the algorithms from the implementation of RankLib library and RC-Rank algorithm. See the results in Tab. 5.2. This dataset can also be considered a large dataset and therefore no repeated runs were averaged, either.

Similar results to the results from the experiments on MSLR dataset could be observed. The *tree algorithms* as RC-Rank, MART and LambdaMART prevailed the others. MART and LambdaMART show again the differences in the measures. While MART performs

Algorithm	NDCG@20	ERR@20	SR	Avg Norm
Best Ranker	1.0000	0.7296	0.8654	1.0000
RC-Rank enhanced	0.8874	0.6658	0.7445	0.7918
RC-Rank baseline	0.8861	0.6647	0.7422	0.7887
MART	0.8814	0.6611	0.7359	0.7789
LambdaMART	0.8780	0.6630	0.7361	0.7770
Coordinate Ascent	0.8682	0.6564	0.7246	0.7577
RankBoost	0.8692	0.6536	0.7223	0.7557
Random Forests	0.8688	0.6536	0.7225	0.7554
RankNet	0.8575	0.6430	0.7060	0.7297
ListNet	0.8541	0.6443	0.7049	0.7267
AdaRank	0.8436	0.6335	0.6917	0.7032
Worst Ranker	0.6543	0.2229	0.1660	0.0000

Table 5.2: Performance of various algorithms (RankLib algorithms and RC-Rank) on Seznam.cz's dataset. NDCG@20, ERR@20 and SeznamRank were measured. Best Ranker and Worst Ranker 'dummy' rows contain the best and the worst possible performance score.

better when measured by NDCG, LambdaMART beat MART when measured by ERR or SR.

RC-Rank enhanced overcame RC-Rank baseline in all three measures. This can be caused by the fact that the parameters of RC-Rank enhanced were tuned by the experts using this particular dataset.

5.2.2.3 Comparison using LETOR 4.0 dataset

The experiment used LETOR 4.0(see Section 3.2.3.1) dataset. This dataset is much smaller than the previous datasets. And because our initial experiments showed that the variance of the results would be too high and the results would not be representative, 5 different runs using 5 different fold splits were performed and then averaged.

Both subparts MQ2007 and MQ2008 were joined together to create a bigger dataset. Then the dataset was split into 5 parts. 4 different parts were joined and one was left out for the testing phase - this process was repeated 5 times and there was always a different part left out. This created 5 different data folds. Each fold then consisted of 80% training data and 20% testing data. During the learning phase, a quarter of training set was used as a validation set. Approximate sizes of the training, validation and testing sets were 60%, 20% and 20% respectively. The final result in the table is a testing performance averaged over all 5 runs.

As the maximal relevance grade of LETOR 4.0 dataset is 2, it was not possible to use SeznamRank (see Sec. 3.3.2.9) which is designed for 5-level relevance grades (or 6-level in the case when unlabeled documents are included). See the results in Tab. 5.3.

Note that the best possible NDCG@20 score is not 1.0 as we would expect (because NDCG is a normalized value). This is caused by the fact that LETOR 4.0 includes a few queries containing only documents annotated by zero relevance grades. Then there is no order among the documents and moreover there is no information about 'what is an actual

Algorithm	NDCG@20	ERR@20	Avg Norm
Best Ranker	0.8407	0.5238	1.0000
RC-Rank enhanced	0.5348	0.3400	0.6097
RC-Rank baseline	0.5302	0.3325	0.5990
Random Forest	0.5036	0.3148	0.5634
MART	0.5026	0.3145	0.5624
LambdaMART	0.5042	0.3123	0.5613
ListNet	0.5055	0.3102	0.5600
Coordinate Ascent	0.5011	0.3106	0.5575
RankBoost	0.5037	0.3086	0.5572
AdaRank	0.4986	0.3023	0.5475
RankNet	0.4950	0.3037	0.5464
Worst Ranker	0.0988	0.0249	0.0000

Table 5.3: Performance of various algorithms (RankLib algorithms and RC-Rank) on LETOR 4.0 dataset. NDCG@20 and ERR@20 were measured. Best Ranker and Worst Ranker ‘dummy’ rows contain the best and the worst possible performance score.

characteristic of a relevant document’ (as there is none). It is questionable how to deal with this issue, but we have adopted approach of the author of RankLib library. The author assigns $NDCG = 0.0$ to such queries where there are no documents with a label different from zero. Another possible approach would be to skip such a query and leave it out of the performance measuring or pre-process the data and delete the queries. However, the deletion is not a proper solution as documents carrying information would be deleted. For example, point-wise algorithms would utilize even this kind of queries and the algorithms would use such query to learn ‘what is the characteristic of an irrelevant document’.

It is obvious that *tree algorithms*, such as *RC-Rank*, *Random Forest*, *MART* and *LambdaMART* prevail. RC-Rank appeared to be the best algorithm in this experiment. As the size of the dataset is rather small, the models can tend to overfit, although the validation was used to choose the model which was simple enough to avoid the overfitting. As in the previous experiments, RC-Rank benefits from the use of Oblivious trees, as it can serve as another mean of regularization. This characteristic of RC-Rank can be very valuable, especially for smaller datasets.

5.2.2.4 Comparison using WCL2R dataset

WCL2R dataset(see Sec. 3.2.3.5) was used to compare the algorithms in this experiment. The same way of data splitting was used in the experiment The original data were split into 5 folds and then combined into 5 different folds. During the learning, a quarter of training set was used as a validation set. This dataset can be considered really small. Therefore, even when the experiments averaged over 5 runs, it is necessary to evaluate the experiment and the results keeping the data size in your mind.

RC-Rank performed well and was again the best algorithm in the experiment with a huge difference to other algorithms. This success can be again explained by the fact that RC-Rank uses Oblivious trees that offers a higher regularization. LambdaMART and MART

Algorithm	NDCG@20	ERR@20	Avg Norm
Best Ranker	1.0000	0.7543	1.0000
RC-Rank enhanced	0.6515	0.6548	0.7598
RC-Rank baseline	0.6501	0.6260	0.7400
RankNet	0.5908	0.5661	0.6707
AdaRank	0.5704	0.5686	0.6621
ListNet	0.5759	0.5586	0.6583
Random Forests	0.5679	0.5610	0.6557
LambdaMART	0.5159	0.5683	0.6347
Coordinate Ascent	0.5452	0.5441	0.6332
MART	0.5503	0.5384	0.6320
RankBoost	0.5434	0.5350	0.6263
Worst Ranker	0.0000	0.0000	0.0000

Table 5.4: Performance of various algorithms (RankLib algorithms and RC-Rank) on WCL2R dataset. NDCG@20 and ERR@20 were measured. Best Ranker and Worst Ranker 'dummy' rows contain the best and the worst possible performance score.

performed not as good as in the previous experiments. The setting of the tree depth of the regression trees could be one of the explanations. The trees built by both algorithms were probably too deep. Unfortunately, to avoid this, it would be necessary to perform this experiment with many different parameter settings and search for the best possible setting. The validation set served only to cut the tree in order to prevent overfitting. However, this aim of this experiment was to provide further comparison of the algorithms, not to find the best possible parameter settings for small datasets.

5.2.3 Summary

The experiments were summarized in the sections dedicated to the particular experiments. RC-Rank performed best in all performed experiments. Oblivious trees could be the reason for that, as the trees improve the regularization of the algorithm. Other tree algorithms, as MART and LambdaMART, proved that can also successfully learn the ranking model (except for the last experiment on WCL2R dataset). As MART algorithm is very similar to RC-Rank (as explained in Sec. 3.4.4.5) and both, MART and RC-Rank, apply the point-wise approach, LambdaMART could be the way to go and how to improve the results. LambdaMART uses multiple additive regression trees that have proved to be successful and it also applies list-wise approach to the ranking problem.

LambdaMART was considered being the algorithm with the best potential and worth further analysis. Therefore, the further experiments will be mainly focused on LambdaMART algorithm.

5.3 Time efficiency of the algorithms

When commercially used, the performance score of the algorithms is not the only decision factor. It also depends on the efficiency of the algorithms. To compare the time efficiency

Algorithm	MSLR	Seznam	AvgNorm
AdaRank	3412376	684319	0.000
RankBoost	4615198	1410637	0.033
ListNet	10697162	1721900	0.122
MART	10637105	2549840	0.140
LambdaMART	15058219	3771462	0.227
RankNet	19677988	2272570	0.255
Random Forest	17696045	4149453	0.271
Coordinate Ascent	40676951	22473891	1.000

Table 5.5: Execution time of training on Seznam and MSLR datasets. The time is in milliseconds.

of the algorithms, the results from the previous experiments were employed. This section presents the comparison of average times needed for training and testing phase.

5.3.1 Experiment setting

Algorithms For the purpose of this experiment, we used all the algorithms available in RankLib library.

Performance measures This time, the quality was not evaluated by a performance measure. The analysis was performed comparing the running time of the algorithms in milliseconds.

Data The execution times of the training and testing were obtained from the experiments that had been previously carried out on Seznam dataset and MSLR10k dataset.

5.3.2 Experiment results

The first, Tab. 5.5, provides the times of training on MSLR and Seznam datasets. The second, Tab. 5.6, provides the times of testing. The times are in milliseconds averaged over several runs in experiment in Sec. 5.2. $AvgNorm$ value is an average of normalized value on both datasets (similarly to (5.1)).

Even though the time values were averaged over several runs of the algorithms, we can see that the variance of the times is quite high. The times are dependent on many factors, such as the complexity of the model, the size of the dataset etc. We can observe two main findings. Tree models are generally slower in the testing phase than other models represented, for example, by a vector of feature weights or by weights of neural network, which is cause by the general complexity of tree models. Also the training phase of the algorithms using trees is slower than, for example, AdaRank. Note, that there are algorithms efficient in the testing phase thanks to the simplicity of the model, while the training phase is really slow. Lastly, also the particular implementation can influence the efficiency of the algorithm and its testing.

Alg	MSLR	Seznam	AvgNorm
AdaRank	5951	3404	0.000
Coordinate Ascent	6122	3496	0.005
RankNet	6151	3720	0.008
Random Forest	16934	9800	0.294
LambdaMART	6485	30923	0.327
ListNet	28728	3554	0.459
RankBoost	30836	3780	0.504
MART	6560	46845	0.512

Table 5.6: Execution time of testing on Seznam and MSLR datasets. The time is in milliseconds.

5.4 Overfitting analysis of LambdaMART algorithm

Overfitting analysis of *LambdaMART* algorithm was performed in order to find out, whether the algorithm tends to overfit. A typical overfitting behaves as shown in Fig. 5.1, i.e. in the case in the figure, the choice of proper complexity would be crucial because from the marked point, the testing error rises. In this section, an experiment will be performed to find out to which extent LambdaMART proves the same overfitting tendencies.

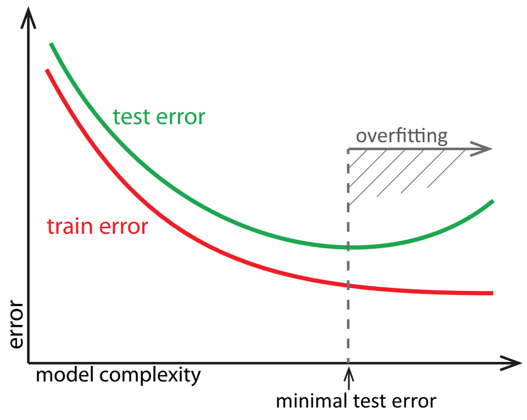


Figure 5.1: Overfitting curves - example

5.4.1 Experiment setting

Algorithms For the overfitting analysis experiment, only LambdaMART algorithm (implementation from *RankLib* library) was used, as it was evaluated as the the best algorithm and a potential competitor for RC-Rank (see Sec. 5.2 for the experiments related to this decision).

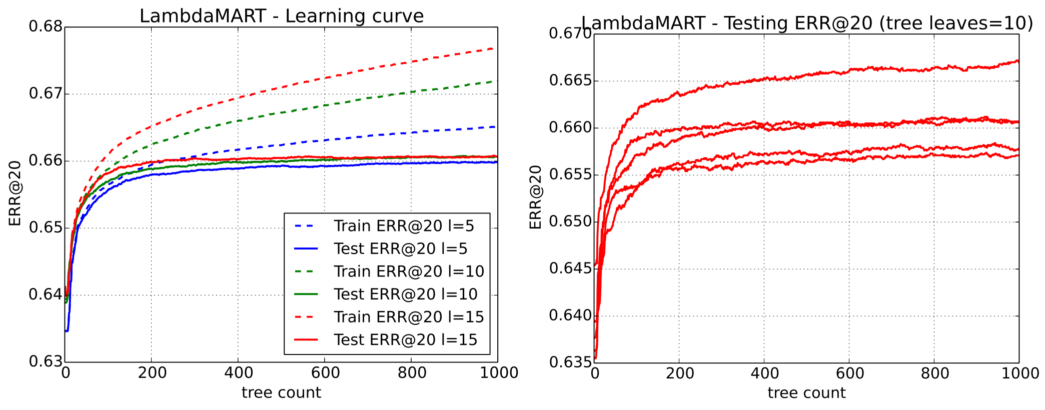
However, due to the similarity of LambdaMART, MART and RC-Rank algorithms, it can be expected that the results are applicable to all algorithms using *additive tree boosting*.

Three different parameter setups of LambdaMART were examined. The models used trees with 5, 10 and 15 leaves, respectively. As further explained, data were split into 5 folds and then each experiment ran 5 times and the results were averaged.

Performance measures As demonstrated in Section 3.3, the measures correlate and it is sufficient to evaluate the experiment using only one of the measures. Please note, that unlike in the example in Fig. 5.1, this experiment does not use error[5] or loss function, but it uses ERR@20 evaluation measure, i.e. the higher the value is, the better the algorithm performs.

Data In order to compare the algorithms and their performance on different datasets, there were 4 different datasets used in the experiment. Namely, LETOR 4.0, MSLR10k, WCL2R and also the dataset provided by Seznam.cz search engine[6](see Sections 3.2.3.1, 3.2.3.2, 3.2.3.5 and 3.2.3.6 respectively). For more information on the characteristics of the datasets see Table 3.1. Only the datasets with no undefined values were used.

5.4.2 Experiment results



(a) Training and testing ERR@20 for three different parameter setups. Averaged over 5 runs using 5 different folds.

(b) Learning curves of 5 runs using 5 different folds. Experiments used trees with 10 leaves.

Figure 5.2: Learning curves of LambdaMART for 3 different tree depths

For the results of this experiment see Fig. 5.2. Two plots are provided. The first plot provides results for all three parameter setups averaged over 5 runs, containing both, training and testing performance. The second plot presents testing performances of single runs when the parameter of number of leaves was set to 10. To make it clear, the averaging of the curves from the second plot served as a source for one of the curves in the first plot. According to the plots, there is no major tendency to overfit on larger datasets. A rapid increasing of both, training and testing performances, can be observed at the beginning of the learning, during the first 100 iterations (trees). Up to 200 trees, the performance is still clearly increasing. However, once 200 trees were reached, the growth of the performance decreases.

This experiment proves that any significant overfitting is not the issue here. Ensemble sizes above 1000 trees were not examined as for the real-world use of the models, it is necessary to keep the size of the models on a reasonable efficient level.

[5]Do not confuse ERR as a performance measure with ERR as an abbreviation for error. ERR performance measure is used in the experiment.

[6]http://search.seznam.cz/

Please note, it is not claimed that it is not necessary to keep the model as simple as possible and that there is no need for regularization. This experiment only proves that the importance of aforementioned is not so high.

5.5 Dataset size analysis

The experiments in this section aimed to analyze the relation between the size of the dataset and the performance of the model. As proved by the authors of [1], the bigger the dataset is, the small the variance error is. Throughout the experiment, the size of the training dataset was being continuously reduced, and the quality of the ranking models trained on the smaller datasets were evaluated.

This experiment consists of two different subparts. The first part of the experiment analyzed the relation between the dataset size and the performance on 4 different datasets. The second part of the experiment then used only one dataset, but the experiment was performed using different algorithms.

5.5.1 Analysis using different datasets

5.5.1.1 Experiment setting

Algorithms For this part of the analysis, only LambdaMART algorithm was evaluated. The algorithm was chosen as a representative of *multiple additive boosting trees* algorithms. Regression trees had 10 leaves and the learning coefficient was set to 0.1.

Performance measures Similarly to the previous experiment, ERR@20 evaluated the performance of models. The results using NDCG and SR would correlate with the results of ERR (see Section 3.3 for more information on performance measures and their correlation). Please, note again, that ERR is meant to be maximized (it is not a loss function).

Results The results are presented in a figure. The figure presents performance of LambdaMART on the datasets of different sizes. X-axis is plot in logarithmic scale.

Data This experiment was performed on 4 different datasets that have been used also in previous experiments. Namely, MSLR10k, MQ2007 (subpart of LETOR 4.0), Seznam and WCL2R datasets. For more information on datasets, see Sec. 3.2.3 and Tab. 3.1.

Each of the datasets went through the same procedure. It was first split into trainset (64% of dataset), validation set (16%) and testset (20% dataset). Then there were several smaller datasets generated from the trainset (the sizes ranged usually from 1% of trainset to 100% of trainset). Then there was a model trained on the prepared datasets and each of the models was then evaluated by the performance measures.

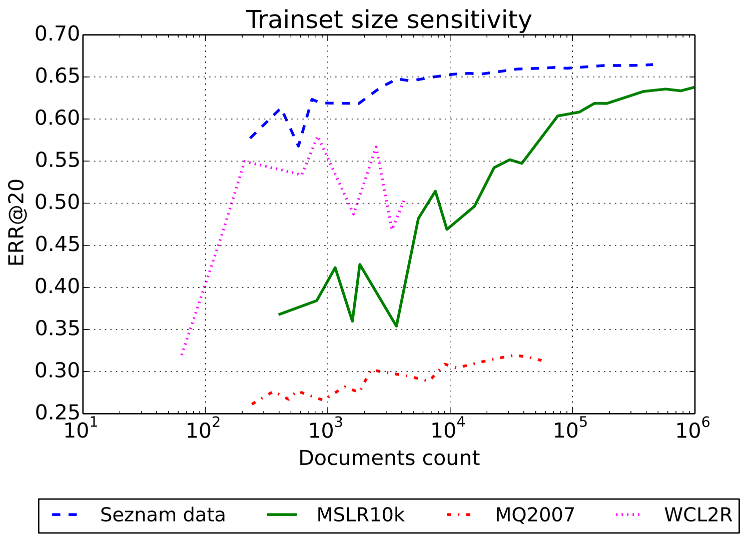


Figure 5.3: Performance of LambdaMART algorithm based on the size of the training set. Comparison of Seznam dataset, MSLR10k, WCL2R and MQ2007 subpart of LETOR 4.0.

5.5.1.2 Experiment results

For the results of the experiment, see Fig. 5.3. WCL2R dataset is very small and therefore the variance of the results of the experiment is high. Because of the small size of WCL2R dataset, even the size of the test set is small which further supports the variance in the results. This phenomenon could be partially eliminated by repeatedly running the experiment using different folds of the data as it was done in the experiment in Sec. 5.2.2.4. However, as WCL2R dataset is not important for us (because of its size and source), we decided not to perform the experiments repeatedly. MQ2007 is a similar case, however the variance of the results is not so high.

The interesting part of the experiment is the comparison of MSLR and Seznam datasets. While Seznam dataset seems to perform reasonably well with less than 10000 samples, MSLR dataset struggles even when 100000 documents are available. Around 600000 documents seems to be the minimal number for MSLR dataset. Of course, even the performance of models trained on Seznam dataset increases with increasing number of document, but from about 5000 documents, the growth is not significant.

There is a possible explanation for the difference between the curves. Both, MSLR and Seznam's datasets are datasets originating from commercial search engines but while Seznam dataset uses carefully chosen features that were really in use, MSLR dataset uses a set of features that were computed especially for the dataset release according to a set of features that are believed to be important from point of view of research community. Therefore, we can assume that the set of features provided in Seznam dataset is of higher quality and it is much easier to find dependencies among the feature values and the relevance labels.

It is likely that if MSLR dataset used the same set of features as Seznam dataset, the performance curves would be similar.

5.5.2 Analysis using different algorithms

5.5.2.1 Experiment setting

Algorithms Unlike the previous part of the experiment, this part uses also other algorithms than LambdaMART. It uses the algorithms provided by *RankLib* implementation. AdaRank, MART, RankBoost, LambdaMART and Random Forest were chosen for the experiment.

Performance measures Same as in the previous part of the experiment.

Results A figure demonstrating the relation between the dataset size and the performance evaluation of a model. Again, logarithmic scale was used for X-axis.

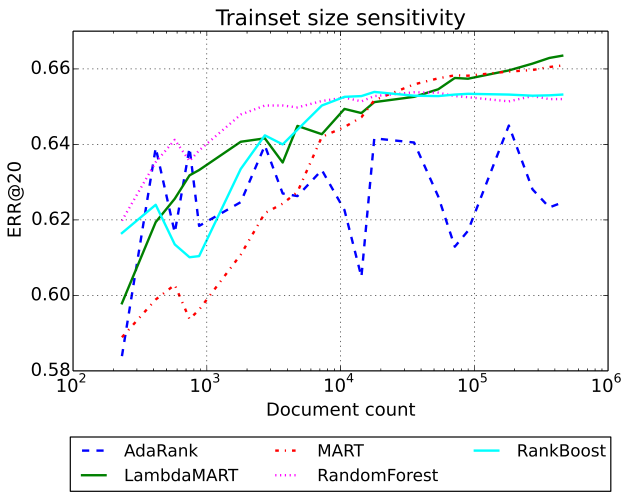


Figure 5.4: Performance of LambdaMART algorithm based on the size of the training set. Comparison of various algorithms.

Data We used the same procedure of data generation as in the previous experiment. However, only Seznam dataset was used for the purpose of this experiment.

5.5.2.2 Experiment results

For the results of the experiment, see Fig. 5.4. It can be observer, that the performance curves of different algorithms behave similarly. However, AdaRank proved a strange behavior, when the performance did not really depend on the data size and the variance of results was high. Unfortunately, we are not able to explain this phenomenon, however we can guess that it is caused by reaching some point of local optimum and that the algorithm struggled to get through. AdaRank's unexpected curve would be probably better if the experiment was repeated more times on different folds.

Concerning the other curves, It seems that Random Forest algorithm performs well even when only small amount of data is available, while MART algorithm seems to be one that definitely needs more data to construct a proper high-quality model. Once there is enough data, LambdaMART and MART outperform the other algorithms.

5.6 Relevance label noise sensitivity

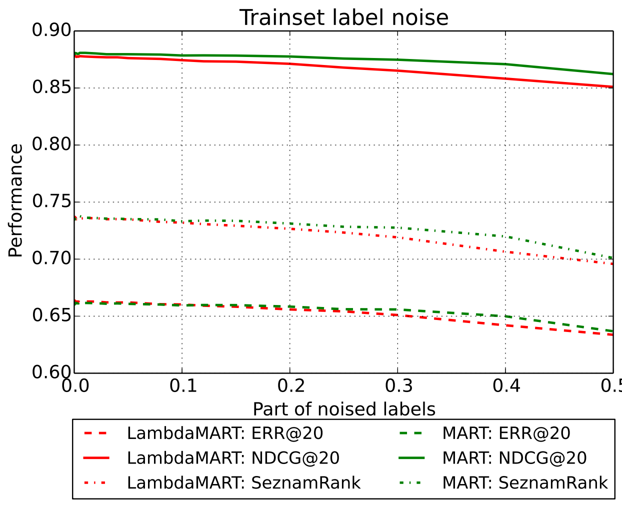
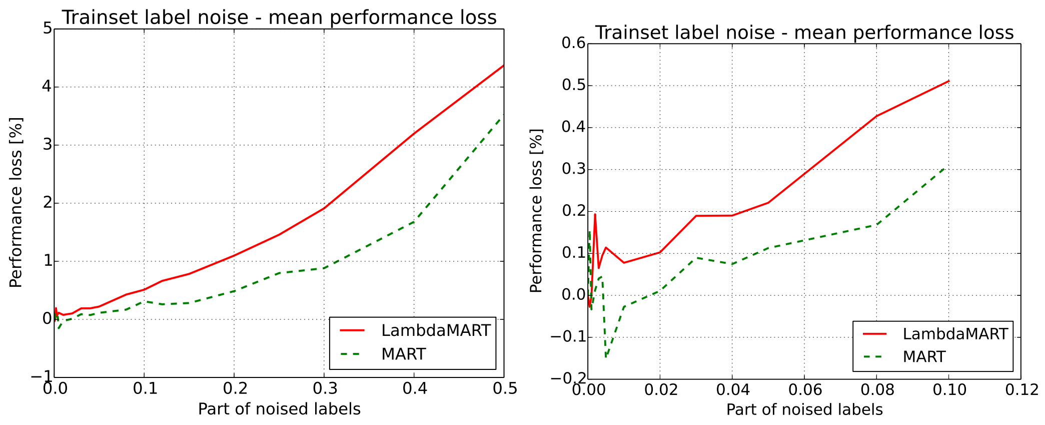


Figure 5.5: Performance of LambdaMART and MART algorithm based on the amount of the relevance labels with noise. Seznam dataset was used.



(a) Overview (0% - 50%)

(b) In detail (0% - 10%)

Figure 5.6: Performance loss based on the noise in the training set labels. How many percent of performance the model lost is presented by Y-axis. When 50% of relevance labels were noised, the performance dropped only by less than 4.5%.

As mentioned before, the relevance labels are assessed by trained annotators. The issues of this process are that a decision of a human being is influenced by his subjective point of view, humans tend to make mistakes etc. This experiment aimed to simulate the mistakes of the annotators and the influence of their subjective point of view by addition of a random noise to the labels of a training set. Several training datasets with the same data samples were generated. Then each of the datasets was noised by a different amount of noise. The performance of the models trained on the noised datasets were then compared.

The results obtained from the experiment were really interesting and surprising. The results will be presented in Section 5.6.1.1.

5.6.1 Experiment setting

Algorithms For this analysis, LambdaMART and MART algorithms were used and compared. Both coming from *RankLib* implementation.

Performance measures The models obtained from this experiment were evaluated by ERR@20, NDCG@20 and SR measures.

Data Seznam dataset was used in this experiment. The dataset was first split into 80% training set and 20% testing set. Then there were several training datasets generated each of them containing different amount of noise. There was no noise added to the testing set and the same set was used to evaluate all the ranking models. 20 different training sets with the noise amount from the range $\langle 0.05\%; 50.00\% \rangle$ were used to analyse the sensitivity to the noise. Note, that the noise was added only to relevance labels. There is no artificially added noise among feature values.

The noise was generated as follows. An adequate part of a dataset was chosen and than each of the labels from the chosen part was randomly changed to one of the valid labels. The probability of change was same for all the relevance labels (uniform distribution) and it was not possible to keep the same label, i.e. the relevance label had to be changed.

5.6.1.1 Experiment results

The results of this example were more than surprising. Let M_1 be a model learned on the original dataset and M_2 be a model learned on the dataset with 50% of randomly changed relevance labels. In Fig. 5.6, we can see that the performance of M_2 was worse only by less than 4.5% in comparison to M_1.

When observing the influence of the noise, it is necessary to note, that 50% of noise does not have to necessarily mean that 50% of information is wrong. For example, when we change relevance label of document d_i from $y_i = 3$ to $y_i = 5$, the relative order among documents with relevance labels lower than 3 and d_i is still the same. The generation of noise can be also interpreted in the way that in average 10% of dataset was added as noisy data to a collection of documents of a given relevance label and there were still enough data to find the dependencies among data and relevances.

This phenomenon would deserve further attention and research, which would be out of the scope of this work. However, this finding is pointing towards the theory that the amount of data and its general quality (e.g. good feature selection) can be more important than precise high quality labeling of the documents.

It was also observed that when there is a little amount[7] of noise added to the trainset, the performance of a model actually increases. This can be caused by the fact that addition of a small amount of noise can serve as a form of regularization during the learning phase.

5.7 LambdaMART - parameter search

There are several parameters that can influence LambdaMART algorithm during the learning. The parameters are: number of trees, number of leaves in a tree, learning rate. The number of trees does not have to be exactly set, as the appropriate ensemble size can be chosen with the help of validation set. The number of leaves influence the depth of trees that are in the ensemble. When the number is too low, the descriptive power of the trees can be insufficient and when the number of leaves is too high, the model can tend to overfit. The learning rate then influences a weight of each tree in the final ensemble of trees.

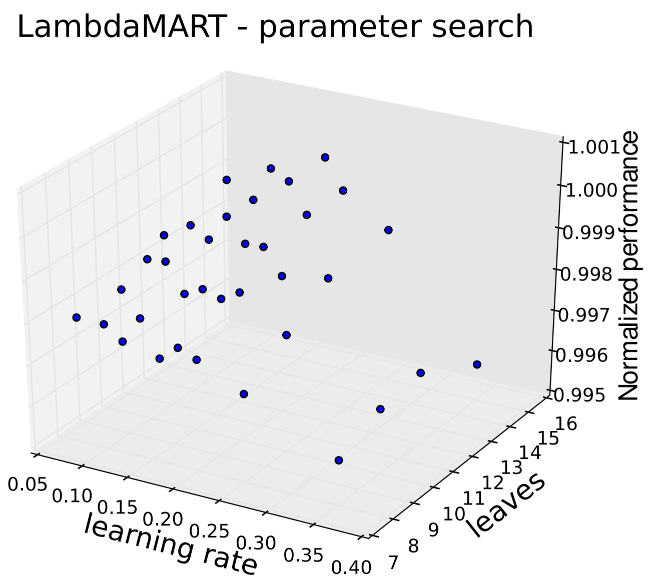


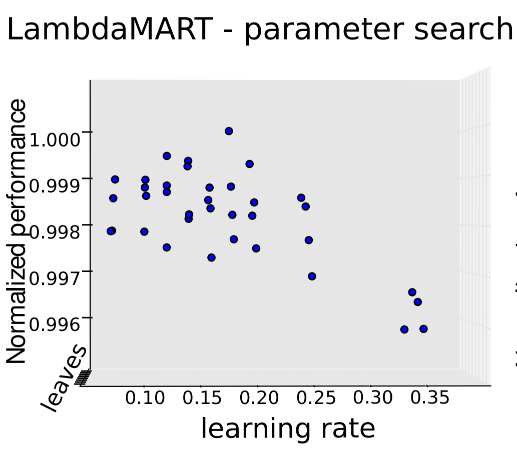
Figure 5.7: Performance of LambdaMART based on learning rate and number of leaves in a tree.

The appropriate parameters' setting can vary for each dataset. This experiment was performed in order to tune the setting of the algorithm on Seznam data and find out if the improvement based on parameter tuning could help LambdaMART to perform better than RC-Rank.

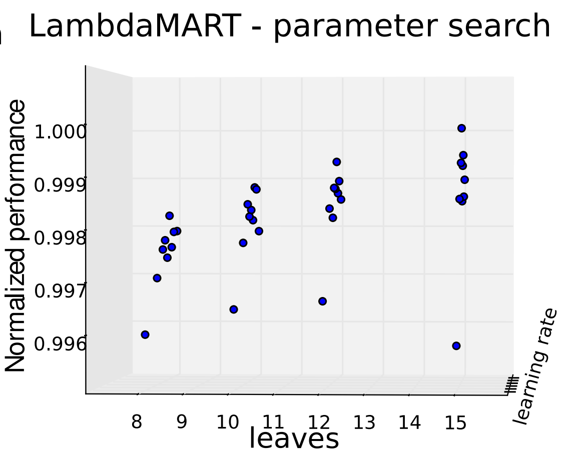
5.7.1 Experiment setting

Algorithms Since this experiment focuses only on the parameter tuning of LambdaMART, thus only LambdaMART algorithm was employed.

[7]It means about 0.5% of the relevance labels changed in the training dataset



(a) Learning rate detail



(b) Number of leaves detail

Figure 5.8: Performance of LambdaMART based on learning rate and number of leaves in a tree. Two plots focusing on single parameter. The optimal values seem to be with parameters set to $number\_of\_leaves = 15$ and $learning\_rate = 0.16$

Data The dataset provided by Seznam.cz was used in this experiment. There was following split of dataset used - 80% train set and 20% test set. However, two different splits were generated in order two create two folds and use them for repeated run.

Performance measures The performance of a model was evaluated by ERR@20, NDCG@20 and SeznamRank measures. LambdaMART used ERR@20 for optimization during the training. As there were two runs of the experiment using different data folds, the final result is the average of both runs. The final evaluation was performed based on a value defined as follows:

$$NormPerformance(m) = \frac{\sum_{z \in Z} z(m) / \max_{k \in M} z(k)}{|M|},$$

where M is a set of all models that have been trained, m is the current model, Z is a set of measures, i.e. NDCG@20, ERR@20 and SR and $z(\cdot)$ is the measure function.

5.7.1.1 Experiment results

As the resulting performance measure scores are related to both parameters, learning rate and number of leaves in a tree, it was necessary to plot a 3D scatterplot. First, Fig. 5.7 displays the performances based on both parameters in 3D overview, and then Fig. 5.8 introduces two scatterplots focusing only on one of the parameters.

It can be observed that the optimal performance was reached with the parameters set to the values around: $learning\_rate = 0.18$ and $number\_of\_leaves = 15$. However, it is necessary to note, that when using such a big dataset, with a big ensemble of trees, the differences in the performance are rather small.

To demonstrate the small differences in the performance, the difference between the worst performing and the best performing models are: $\Delta NDCG@20 = 0.0025$, $\Delta ERR@20 = 0.0028$ and $\Delta SR = 0.00415$. It can be concluded that the parameter tuning for LambdaMART serves rather to influence the process of learning (e.g. faster convergence) then the final performance of a model. However, it is important to keep in mind that it is important to keep the parameter values in reasonable ranges and that both, learning rate and number of leaves in a tree, can server as a mean for regularization.

5.8 LambdaMART - ERR@k cut-off position analysis

Constant k and cut-off position related to performance measures were already discussed in Sec. 3.3.2.10. It is clear what the purpose of the cut-off position is when talking about a performance evaluation on a testset. For example, it is reasonable to use ERR@10 when only top 10 documents are displayed. However, LambdaMART allows us to directly optimize a performance measure. A question rises - what is the influence of the cut-off position on the learning phase. Is it better to keep the same cut-off position for both, training and testing evaluation measure? Or is it better to use bigger k constant and try to find dependencies on further positions in the list? This experiment was performed in order to outline the answer for the questions.

5.8.1 Experiment setting

Algorithms As mentioned in the introduction, only LambdaMART algorithm will be used in the experiment. LambdaMART allows us to directly optimize a given performance measure.

Performance measures Although, LambdaMART will be optimized using ERR@k (with k within the range $\langle 2; 50 \rangle$), the following performance measures will be used to evaluate the model: ERR@20, NDCG@20 and SR.

Data For the purpose of this experiment, MSLR dataset was used. The dataset was split into 80% and 20% for trainset and testset, respectively. Two different folds were generated for two different runs. The results for both runs were then averaged.

5.8.1.1 Experiment results

The results in Tab. 5.7 demonstrates that there is no significant relation between constant k (specifying cut-off position of the training performance measure) and the performance. Which partially confirms the results from Sec. 5.6. The dataset is big enough and the model is robust enough to provide the model with the dependencies even when there is no importance put on the further positions of the list. Moreover, the importance of the lower positions is really low and therefore its influence and the information provided is really small in comparison to the positions at the top of a ranked list.

k	NDCG@20	ERR@20	SR	Norm Perf
14	0.57815	0.6321	0.32215	0.989224005
18	0.5824	0.63165	0.32445	0.99375603
44	0.58075	0.63295	0.3254	0.994465052
20	0.5828	0.6319	0.32485	0.994522863
40	0.58265	0.6322	0.3254	0.99515527
48	0.5808	0.6342	0.32575	0.99550532
30	0.58085	0.63455	0.32575	0.995717206
10	0.5831	0.6332	0.3256	0.996139661
26	0.5824	0.6346	0.3258	0.996678023
50	0.5847	0.632	0.32665	0.997493577
38	0.58165	0.6362	0.32655	0.997853407
34	0.5829	0.63575	0.32695	0.998738058

Table 5.7: Performance based on cut-off position

Chapter 6

Conclusions

Even though, LTR and IR are hot research topics, there is still a long way to go. It may seem that everything have been solved already but when comparing the results from Sec. 5.2.2.1, where the best performing algorithm reached only about 50% of potentially best score, it appears that there is still a space for improvements. Many state-of-the-algorithms still solve the issue buy reformulation of the problems without a straightforward approach and there is still a lot of theory missing and waiting to be developed.

Besides, concerning the search engines, there are many sub problems that emerge recent years. Very interesting problems concerning LTR in the search engines are, for example, the issue of click logs utilization and the creation of click models and automatic relevance label assessment. Further, there is a whole branch of issues related to Sponsored search which is also very important for the moder search engines. Moreover, the possibilities of diversification and personalization of the topics are also in the center of the attention.

As it was explained in Chapter 1, this thesis aimed to thoroughly analyze the current state of the art of Learning to Rank. In Chapter 2, the description of the background of the topic and this work was first proposed. Then, Chapter 3 covered the theoretical analysis and the listings of LTR in the current state. The description of the general framework of LTR task is first introduced, followed by the description of available datasets and their statistics. After several applicable performance measures were listed and their characteristics were analyzed, a thorough list of LTR algorithms and their categorization to point-wise, pair-wise and list-wise algorithms is presented. At the end of the chapter, we also suggest public LTR contests, as a very useful source of information and state-of-the-art algorithm comparisons.

In Chapter 4 describes the circumstances of the development of our own implementation. According to our findings from the previous chapters and also according to the initial experiments, LambdaMART was chosen as the potentially best state-of-the-art LTR algorithm. A description of our implementation and the comparison to an existing implementation is also presented. Finally, the chapter also suggests using our own implementation as a good tool for analysis and getting better insight, mainly thanks to good properties of Python programming language as a good prototyping language and also because of visualization tool we developed in order to examine LambdaMART algorithm and the forces that can be found inside the algorithm.

Chapter 5 starting on page 53 provides several experiments offering interesting and sometimes surprising conclusions about the datasets and algorithms. To sum up the experiments that were performed:

- A thorough analysis of performance of LTR algorithms on various datasets. This analysis proved that RC-Rank was the best performing algorithm that was available to us. However, MART and LambdaMART algorithm were not much worse.
- The algorithms were compared by the means of time efficiency during the training and testing.
- LambdaMART (as a representative of the algorithms using tree boosting) was examined how high is its tendency to overfit.
- Further, the analysis of the performance of a model based on a training dataset size was performed which showed that the necessary amount of data depends mainly on a particular dataset.
- Very interesting results were observed when the sensitivity of a model to a noise in relevance labels was examined. We discovered that even when 50% of labels are noised, the performance of the model decreases only by less than 5%.
- As LambdaMART was chosen as a potentially best state-of-the-art algorithm, a search for a best parameter setting was performed.
- And lastly, the role and the influence of k cut-off position of a training measure was examined on LambdaMART algorithm. Coming up with the finding that the influence of k constant is really small.

The results of particular experiments are introduced in self-standing sections in Chapter 5. However, it is necessary to mention that RC-Rank was consistently the best performing algorithm from all the tested algorithms. As MART and LambdaMART algorithms performed also well, it can be assumed that employing an algorithm using gradient boosting of decision trees is a key to the success. The main difference between MART and RC-Rank is that RC-Rank uses oblivious trees and therefore the use of oblivious decision trees could improve even the performance of LambdaMART which is a good idea for a further research and work in this topic.

As it was already mentioned in the Introduction, to the best of our knowledge, there is no thorough analysis of available algorithms comparing them to each other on the same datasets. The contributions of this thesis are mainly the exhaustive state-of-the-art analysis and description in Chapter 3 which could be used as a self-standing catalogue of LTR methods. The chapter also suggests several datasets with their statistics which can be very useful for anybody who search for a optimal dataset for his experiments. The thesis then provides a comprehensive set of experiments offering a lot of interesting dependencies among LTR subjects (datasets, algorithms and even performance measures) and lastly our own implementation of LambdaMART algorithm was suggested and compared to an existing implementation of LTR algorithms.

There is still a lot of open and unsolved questions (issues) in Learning to Rank and Information Retrieval and as the amount of data is growing as well as the demands of users, the importance of this branch can only rise. This thesis can then serve as a guide to any researcher interested in this topic, providing him with essential knowledge and experiments.

Bibliography

[1] BRAIN, D. – WEBB, G. I. On The Effect of Data Set Size on Bias And Variance in Classification Learning. In RICHARDS, D. et al. (Ed.) *Proceedings of the Fourth Australian Knowledge Acquisition Workshop (AKAW '99)*, s. 117–128, Sydney, 1999. The University of New South Wales.

[2] BREIMAN, L. Random Forests. *Machine Learning*. 2001, 45, 1, s. 5–32. ISSN 0885-6125.

[3] BRODER, A. A Taxonomy of Web Search. *SIGIR Forum*. September 2002, 36, 2, s. 3–10. ISSN 0163-5840.

[4] BURGES, C. et al. Learning to rank using gradient descent. In *In ICML*, s. 89–96, 2005.

[5] BURGES, C. J. C. From RankNet to LambdaRank to LambdaMART: An Overview. Technical report, Microsoft Research, 2010. Available online at: <`http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf`>.

[6] BURGES, C. J. C. – RAGNO, R. – LE, Q. V. Learning to Rank with Nonsmooth Cost Functions. In SCHÖLKOPF, B. et al. (Ed.) *NIPS*, s. 193–200. MIT Press, 2006. ISBN 0-262-19568-2.

[7] BURGES, C. J. et al. Learning to Rank Using an Ensemble of Lambda-Gradient Models. *Journal of Machine Learning Research-Proceedings Track*. 2011, 14, s. 25–35.

[8] CAO, Z. et al. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, s. 129–136, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

[9] CHANG, C.-C. – LIN, C.-J. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* May 2011, 2, 3, s. 27:1–27:27. ISSN 2157-6904.

[10] CHAPELLE, O. – ZHANG, Y. A Dynamic Bayesian Network Click Model for Web Search Ranking. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, s. 1–10, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4.

[11] CHAPELLE, O. et al. Expected Reciprocal Rank for Graded Relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, s. 621–630, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-512-3.

[12] CHEN, W. et al. Ranking Measures and Loss Functions in Learning to Rank. In *NIPS*, s. 315–323, 2009.

[13] CRAMMER, K. – SINGER, Y. Pranking with Ranking. In *Advances in Neural Information Processing Systems 14*, s. 641–647. MIT Press, 2001.

[14] DANG, V. RankLib. Online, 2011. Available online at: <`https://sourceforge.net/p/lemur/wiki/RankLib/`>. University of Massachusetts Amherst, Available at: https://sourceforge.net/p/lemur/wiki/RankLib/.

[15] DUH, K. – 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*, SIGIR '08, s. 251–258, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4.

[16] DUPRET, G. – LIAO, C. A Model to Estimate Intrinsic Document Relevance from the Clickthrough Logs of a Web Search Engine. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, s. 181–190, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6.

[17] EFFECTIX.COM. *Google vs. Seznam.cz* [online]. Feb 2013. [cit. May 2014]. Available online at: <`http://www.doba-webova.com/cs/google-vs-seznam`>.

[18] FRAKES, W. B. – BAEZA-YATES, R. (Ed.). *Information Retrieval: Data Structures and Algorithms.* Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1992. ISBN 0-13-463837-9.

[19] FREUND, Y. – SCHAPIRE, R. E. A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* August 1997, 55, 1, s. 119–139. ISSN 0022-0000.

[20] FREUND, Y. et al. An Efficient Boosting Algorithm for Combining Preferences. *J. Mach. Learn. Res.* December 2003, 4, s. 933–969. ISSN 1532-4435.

[21] FRIEDMAN, J. H. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics.* 2000, 29, s. 1189–1232.

[22] GANTZ, J. – REINSEL, D. *THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East* [online]. Dec 2012. [cit. May 2014]. Available online at: <`http://idcdocserv.com/1414`>.

[23] GENS, F. *IDC Predictions 2013: Competing on the 3rd Platform* [online]. Dec 2012. Available online at: <`http://www.idc.com/research/Predictions13/downloadable/238044.pdf`>.

[24] GULIN, À. – KARPOVICH, P. Greedy function optimization in learning to rank. In *Lection on the RuSSIR 2009 conference.*, 2009. Available online at: <`http://romip.ru/russir2009/slides/yandex/lecture.pdf`>.

[25] HERBRICH, R. – GRAEPEL, T. – OBERMAYER, K. Large Margin Rank Boundaries for Ordinal Regression. In SMOLA, A. et al. (Ed.) *Advances in Large Margin Classifiers*, s. 115–132, Cambridge, MA, 2000. MIT Press.

[26] JäRVELIN, K. – KEKäLäINEN, J. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* October 2002, 20, 4, s. 422–446. ISSN 1046-8188.

[27] JIN, R. – VALIZADEGAN, H. – LI, H. Ranking refinement and its application to information retrieval. In *In WWW '08: Proceeding of the 17th international conference on World Wide Web*, s. 397–406, 2008.

[28] LI, P. – BURGES, C. J. C. – WU, Q. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In PLATT, J. C. et al. (Ed.) *NIPS*. MIT Press, 2007.

[29] MANNING, C. D. – RAGHAVAN, P. – SCHüTZE, H. *Introduction to Information Retrieval*. New York, NY, USA : Cambridge University Press, 2008. ISBN 0521865719, 9780521865715.

[30] MCCOLLUM, J. *Standing Between Google and World Domination* [online]. Sep 2008. [cit. May 2014]. Available online at: <`http://www.marketingpilgrim.com/2008/09/standing-between-google-and-world-domination.html`>.

[31] METZLER, D. – BRUCE CROFT, W. Linear Feature-based Models for Information Retrieval. *Inf. Retr.* June 2007, 10, 3, s. 257–274. ISSN 1386-4564.

[32] MITCHELL, T. M. *Machine Learning*. New York, NY, USA : McGraw-Hill, Inc., 1 edition, 1997. ISBN 0070428077, 9780070428072.

[33] MOFFAT, A. – ZOBEL, J. Rank-biased Precision for Measurement of Retrieval Effectiveness. *ACM Trans. Inf. Syst.* December 2008, 27, 1, s. 2:1–2:27. ISSN 1046-8188.

[34] PAANANEN, A. Comparative Analysis of Yandex and Google Search Engines. Master's Thesis, Helsinki Metropolia University of Applied Sciences, May 2012.

[35] PLAKHOV, A. *Entity-oriented Search Result Diversification* [online]. Yandex, 2011. [cit. May 2014]. Available online at: <`http://romip.ru/russiras/doc/slides-2011/yandex.pdf`>.

[36] QIN, T. et al. Query-level loss functions for information retrieval. *INFORMATION PROCESSING AND MANAGEMENT*. 2008, 44, 2.

[37] RIDER, F. *The Scholar and the Future of the Research Library: A Problem and Its Solution*. New York, NY, USA : Hadham Press, 1 edition, 1944.

[38] ROKACH, L. *Data mining with decision trees: theory and applications*. 69. Singapore : World scientific, 2008.

[39] SANDERSON, M. – CROFT, W. The History of Information Retrieval Research. *Proceedings of the IEEE*. May 2012, 100, Special Centennial Issue, s. 1444–1451. ISSN 0018-9219.

[40] SMITH, D. *University Lecture: CS6200 Information Retrieval* [online]. Northeastern University, Sep 2013. [cit. May 2014]. Available online at: <`http://www.ccs.neu.edu/course/cs6200f13/cs6200-f13-2.pdf`>.

[41] TAYLOR, M. et al. SoftRank: Optimizing Non-smooth Rank Metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, s. 77–86, New York, NY, USA, 2008. ACM.

[42] WU, Q. et al. Adapting Boosting for Information Retrieval Measures. *Inf. Retr.* June 2010, 13, 3, s. 254–270. ISSN 1386-4564.

[43] XIA, F. et al. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, s. 1192–1199, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

[44] XU, J. – 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*, SIGIR '07, s. 391–398, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7.

[45] XU, J. et al. Directly optimizing evaluation measures in learning to rank. In *SIGIR 2008 - Proceedings of the 31th annual international ACM SIGIR conference.* ACM Press, 2008.

[46] XUE, G.-R. et al. Optimizing Web Search Using Web Click-through Data. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, s. 118–126, New York, NY, USA, 2004. ACM. ISBN 1-58113-874-1.

[47] YUE, Y. et al. A Support Vector Method for Optimizing Average Precision. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, s. 271–278, 2007.

[48] ZHU, Z. A. et al. A Novel Click Model and Its Applications to Online Advertising. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, s. 321–330, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6.

Appendix A

List of Abbreviations

CSV Comma Separated Values, a file format

ERR Expected Reciprocal Rank

GBM Gradient Boosting Machine

IDC International Data Corporation

IR Information Retrieval

LTR Learning to Rank

MART Multiple Additive Regression Trees

NDCG Normalized Discounted Cumulative Gain

RBP Rank Biased Performance

SERP Search Engine Results Page

SR SeznamRank - evaluation measure provided by Seznam.cz

WTA Winner Takes All

Appendix B

Contents of CD and Instructions

B.1 Setup of the environment

It is necessary to have Java installed and also Python on your computer. Preferably Unix operating system.

RankPy implementation further requires:

- NumPy (python-numpy package)
- MatplotLib (python-matplotlib package)
- SciPy (python-scipy package)
- SciPy (python-scipy package)
- Scikit (scikit_learn package)

B.2 Contents of CD

The following list characterize the folder hierarchy and the description.

- example/
- – data/ - contains three example datasets generated from MSLR10k
- – log/ - empty folder which is used for storing the logs from RankPy
- – model/ - empty folder which is used for storing the models from RankPy and RankLib
- – RankLib/ - files necessary to run RankLib experiments
- – RankPy/ - files necessary to run RankPy experiments (can be used as a self-standing Eclipse project)
- – example_ranklib.sh - a file containing script running a demonstrative example using RankLib

- – example_rankpy.sh - a file containing script running a demonstrative example using RankPy
- text/
- – Modry-thesis-2014.pdf - the file containing PDF file with the thesis text
- – src/ - folder containing TeX source files

B.3 Further instructions

It is necessary to first copy all the files from *example* folder to your local hard drive or any other space where you have the proper rights.