MORGAN & CLAYPOOL PUBLISHERS

Learning to Rank for Information Retrieval and Natural Language Processing

Second Edition

Hang Li

Synthesis Lectures on Human Language Technologies

Graeme Hirst, *Series Editor*

Learning to Rank for Information Retrieval and Natural Language Processing

Second Edition

Synthesis Lectures on Human Language Technologies

Semi-Supervised Learning and Domain Adaptation in Natural Language Processing
Anders Søgaard
2013

Semantic Relations Between Nominals
Vivi Nastase, Preslav Nakov, Diarmuid Ó Séaghdha, and Stan Szpakowicz
2013

Computational Modeling of Narrative
Inderjeet Mani
2012

Natural Language Processing for Historical Texts
Michael Piotrowski
2012

Sentiment Analysis and Opinion Mining
Bing Liu
2012

Discourse Processing
Manfred Stede
2011

Bitext Alignment
Jörg Tiedemann
2011

Linguistic Structure Prediction
Noah A. Smith
2011

Learning to Rank for Information Retrieval and Natural Language Processing
Hang Li
2011

Computational Modeling of Human Language Acquisition
Afra Alishahi
2010

Introduction to Arabic Natural Language Processing
Nizar Y. Habash
2010

Cross-Language Information Retrieval
Jian-Yun Nie
2010

Automated Grammatical Error Detection for Language Learners
Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel Tetreault
2010

Data-Intensive Text Processing with MapReduce
Jimmy Lin and Chris Dyer
2010

Semantic Role Labeling
Martha Palmer, Daniel Gildea, and Nianwen Xue
2010

Spoken Dialogue Systems
Kristiina Jokinen and Michael McTear
2009

Introduction to Chinese Natural Language Processing
Kam-Fai Wong, Wenjie Li, Ruifeng Xu, and Zheng-sheng Zhang
2009

Introduction to Linguistic Annotation and Text Analytics
Graham Wilcock
2009

Dependency Parsing
Sandra Kübler, Ryan McDonald, and Joakim Nivre
2009

Statistical Language Models for Information Retrieval
ChengXiang Zhai
2008

Learning to Rank for Information Retrieval and Natural Language Processing, Second Edition

Hang Li

www.morganclaypool.com

ISBN: 9781627055840 paperback
ISBN: 9781627055857 ebook

DOI 10.2200/S00607ED2V01Y201410HLT026

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON HUMAN LANGUAGE TECHNOLOGIES

Lecture #26
Series Editor: Graeme Hirst, *University of Toronto*
Series ISSN
Synthesis Lectures on Human Language Technologies
Print 1947-4040 Electronic 1947-4059

Learning to Rank for Information Retrieval and Natural Language Processing

Second Edition

Hang Li
Huawei Technologies

SYNTHESIS LECTURES ON HUMAN LANGUAGE TECHNOLOGIES #26

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Learning to rank refers to machine learning techniques for training a model in a ranking task. Learning to rank is useful for many applications in information retrieval, natural language processing, and data mining. Intensive studies have been conducted on its problems recently, and significant progress has been made. This lecture gives an introduction to the area including the fundamental problems, major approaches, theories, applications, and future work.

The author begins by showing that various ranking problems in information retrieval and natural language processing can be formalized as two basic ranking tasks, namely ranking creation (or simply ranking) and ranking aggregation. In ranking creation, given a request, one wants to generate a ranking list of offerings based on the features derived from the request and the offerings. In ranking aggregation, given a request, as well as a number of ranking lists of offerings, one wants to generate a new ranking list of the offerings.

Ranking creation (or ranking) is the major problem in learning to rank. It is usually formalized as a supervised learning task. The author gives detailed explanations on learning for ranking creation and ranking aggregation, including training and testing, evaluation, feature creation, and major approaches. Many methods have been proposed for ranking creation. The methods can be categorized as the pointwise, pairwise, and listwise approaches according to the loss functions they employ. They can also be categorized according to the techniques they employ, such as the SVM based, Boosting based, and Neural Network based approaches.

The author also introduces some popular learning to rank methods in details. These include: PRank, OC SVM, McRank, Ranking SVM, IR SVM, GBRank, RankNet, ListNet & ListMLE, AdaRank, SVM MAP, SoftRank, LambdaRank, LambdaMART, Borda Count, Markov Chain, and CRanking.

The author explains several example applications of learning to rank including web search, collaborative filtering, definition search, keyphrase extraction, query dependent summarization, and re-ranking in machine translation.

A formulation of learning for ranking creation is given in the statistical learning framework. Ongoing and future research directions for learning to rank are also discussed.

KEYWORDS

learning to rank, ranking, ranking creation, ranking aggregation, information retrieval, natural language processing, supervised learning, web search, collaborative filtering, machine translation.

Contents

Preface

This book presents a survey on learning to rank and describes methods for learning to rank in detail. The major focus of the book is *supervised learning for ranking creation.*

The book targets researchers and practitioners in information retrieval, natural language processing, machine learning, data mining, and other related fields. It assumes that the readers of the book have basic knowledge of statistics and machine learning.

Chapter 1 gives a formal definition of learning to rank. Chapter 2 describes learning for ranking creation, and Chapter 3 describes learning for ranking aggregation. Chapter 4 explains in details about state-of-the-art learning to rank methods. Chapter 5 presents applications of learning to rank. Chapter 6 introduces theory of learning to rank. Chapter 7 introduces ongoing and future research on learning to rank.

I would like to express my sincere gratitude to my former colleagues, Tie-Yan Liu, Jun Xu, Tao Qin, Yunbo Cao, and Yunhua Hu. We worked together on learning to rank. Many thanks go to our former intern students, Zhe Cao, Ming-Feng Tsai, Xiubo Geng, Yanyan Lan, Fen Xia, Ming Li, Xin Jiang, and Wei Chen, who also participated in the research. I am very grateful to Wei-Ying Ma, Hsiao-Wuen Hon, and Harry Shum for their encouragement and guidance.

I also thank our collaborators in Microsoft, Chris Burges, Stephen Robertson, Michael Taylor, John Guiver, Dmitriy Meyerzon, Victor Poznanski, Rangan Majumder, and Steven Yao, and our collaborators in other organizations, Rong Jin, Zhi-Hua Zhou, Hamed Valizadegan, Cheng Xiang Zhai, and Thorsten Joachims.

I am deeply indebted to Jun Xu and Tao Qin who provided materials for writing this book.

Many thanks also go to Chen Wang, Wei Wu, Wei Chen, Jun Xu, Xin Jiang, Shuxin Wang, and two anonymous reviewers, who read the draft of this book and made many valuable comments.

I would also like to thank Graeme Hirst and Michael Morgan. Without their support, this book would not have been published.

Hang Li
October 4, 2014

CHAPTER 1

Learning to Rank

1.1 RANKING

There are many tasks in information retrieval (IR) and natural language processing (NLP), for which the central problem is ranking. These include document retrieval, entity search, question answering, meta-search, personalized search, online advertisement, collaborative filtering, document summarization, and machine translation.

In our view, there are basically two types of ranking problems: ranking creation[1] (or simply ranking) and ranking aggregation. Ranking creation is to create a ranking list of objects using the features of the objects, while ranking aggregation is to create a ranking list of objects using multiple ranking lists of the objects, as will be formally described later in this chapter.

Document retrieval, collaborative filtering, re-ranking in machine translation are examples of ranking creation, and meta-search is an example of ranking aggregation.

DOCUMENT RETRIEVAL

Document retrieval includes web search, enterprise search, desktop search, etc. Although having limitations, it is still the most practical way for people to access the enormous amount of information existing on the web and computers. For example, according to a report by IProspect [2], 56% of the internet users use web search every day and 88% of the internet users use web search every week.

Document retrieval can be described as the following task (cf., Fig. 1.1), in which ranking plays a key role. The retrieval system maintains a collection of documents. Given a query from the user, the system retrieves documents containing the query words from the collection, ranks the documents, and presents the top ranked list of documents (say, 1,000 documents) to the user. Ranking is performed mainly based on the relevance of the documents with respect to the query.

COLLABORATIVE FILTERING

Collaborative filtering is the most fundamental model for computer systems to make recommendations to the users in electronic commerce, online advertisement, etc. For example, if the users' preferences on some of the movies in a database are known, then we can employ collaborative filtering to recommend to the users movies which they might have not seen and might be interested in.

[1]Ranking creation is a term coined by the author of this book.
[2]http://www.iprospect.com/premiumPDFs/iProspectSurveyComplete.pdf

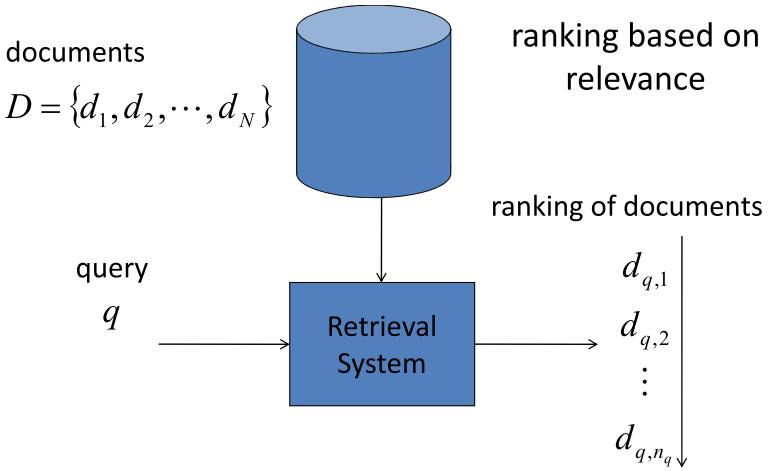

Figure 1.1: Document Retrieval. Downward arrow represents ranking of documents

The data in collaborative filtering is given in a matrix, in which rows correspond to users and columns correspond to items (cf., Fig. 1.2). Some elements of the matrix are known, while the others are not. The elements represent users' ratings on items where the ratings have several grades (levels). The question is how to determine the unknown elements of the matrix. One common assumption is that similar users may have similar ratings on similar items. When a user is specified, the system suggests a ranking list of items with the high grade items on the top.

	Item1	**Item2**	**Item3**	**...**	**ItemN**
User1	5	4			
User2	1		2		2
...		?	?	?	
UserM	4	3			

Figure 1.2: Collaborative Filtering

MACHINE TRANSLATION

Machine translation can help people to access information cross languages and thus is very important. Given a sentence in the source language, usually, there are a large number of possible translations (sentences) in the target language. The quality of translations can vary, however. How to select the best translation(s) is the key question.

A popular approach to machine translation consists of two phases: candidate generation and re-ranking (see Fig. 1.3). Given a sentence in the source language, the system first generates

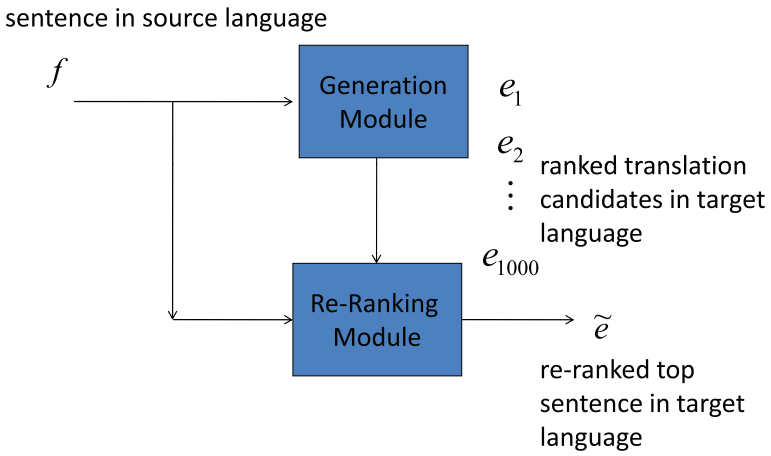

Figure 1.3: Machine Translation

and ranks all possible candidate translations in the target language using a generative model, then it conducts re-ranking on the top candidate translations (say, 1,000 candidates) using a discriminative model, and, finally, it chooses the top ranked candidate as output. The re-ranking process is performed based on the likelihood of candidates' being good translations, and it is critical to the performance of machine translation.

META-SEARCH

A meta-search system is a system that sends the user's request to several search systems and aggregates the results from those search systems. Meta-search is becoming more and more important when web continues to evolve, and more and more search systems (sometimes in different domains) become available.

More formally, in meta-search, the query is submitted to several search systems and ranking lists of documents are returned from the systems. The meta-search system then combines all the ranking lists and generates a new ranking list (meta ranking list), which is better than all the individual ranking lists. In practice, the sets of documents returned from different systems can be

different. One can take the union of the sets of documents as the final set of documents. Figure 1.4 illustrates the process of meta-search.

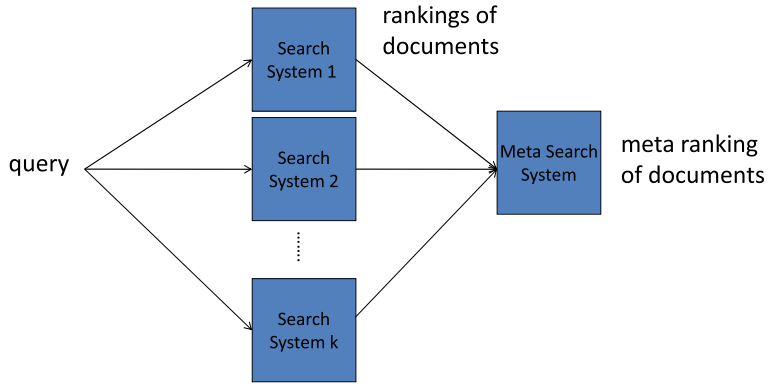

Figure 1.4: Meta-Search

1.2 LEARNING TO RANK

Recently, a new area called learning to rank has emerged in the intersection of machine learning, information retrieval, and natural language processing. Learning to rank is about performing ranking using machine learning techniques. It is based on previous work on ranking in machine learning and statistics, and it also has its own characteristics.

There may be two definitions on learning to rank. In a broad sense, learning to rank refers to any machine learning techniques for ranking. In a narrow sense, learning to rank refers to machine learning techniques for building ranking models in ranking creation and ranking aggregation described above. This book takes the latter definition (narrow sense). Figure 1.5 gives a taxonomy of problems in learning to rank.

Recent years have seen significant efforts on research and development of learning to rank technologies. Many powerful methods have been developed and some of them have been successfully applied to real applications such as web search. Over one hundred papers on the topic have been published. Benchmark data sets have been released (e.g., [74]), and a competition on the task [3] has also been carried out. Workshops (e.g., [60, 67, 68]) and journal special issues have been organized (e.g., [73]). A book devoted to the topic has also been published [72].

[3]Yahoo Learning to Rank Challenge. http://learningtorankchallenge.yahoo.com/

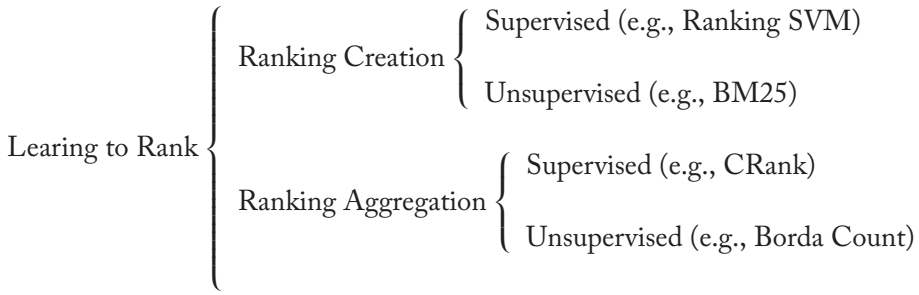

Figure 1.5: Taxonomy of Problems in Learning to Rank

1.3 RANKING CREATION

We can generalize the ranking creation problems already described as a more general task. Suppose that there are two sets. For simplicity, we refer to them as a set of requests $\mathcal{Q} = \{q_1, q_2, \cdots, q_i, \cdots, q_M\}$ and a set of offerings (or objects) $\mathcal{O} = \{o_1, o_2, \cdots, o_j, \cdots, o_N\}$, respectively[4]. $\mathcal{Q}$ can be a set of queries, a set of users, and a set of source sentences in document retrieval, collaborative filtering, and machine translation, respectively. $\mathcal{O}$ can be a set of documents, a set of items, and a set of target sentences, respectively. Note that $\mathcal{Q}$ and $\mathcal{O}$ can be infinite sets. Given an element q of $\mathcal{Q}$ and a subset O of $\mathcal{O}$ ($O \in 2^{\mathcal{O}}$), we are to rank the elements in O based on the information from q and O.

Ranking (ranking creation) is performed with ranking (scoring) function $F(q, O) : \mathcal{Q} \times \mathcal{O}^n \rightarrow \Re^n$

$$S_O = F(q, O)$$

$$\pi = \text{sort}_{S_O}(O),$$

where $n = |O|$, q denotes an element of $\mathcal{Q}$, O denotes a subset of $\mathcal{O}$, S_O denotes a set of scores of elements in O, and π denotes a ranking list (permutation) on elements in O sorted by S_O. Note that even for the same O, F can give two different ranking lists with two different q's. That is to say, we are concerned with ranking on O, with respect to a specific q.

Instead of using $F(q, O)$, we usually use ranking (or scoring) function $f(q, o)$ for ease of manipulation, where q is an element of $\mathcal{Q}$, o is an element of O, and s_o is a score of o. The ranking function $f(q, o)$ assigns a score to each o in O and the elements in O are then sorted by using the scores. That means ranking is actually performed by sorting with $f(q, o) : \mathcal{Q} \times \mathcal{O} \rightarrow \Re$

$$s_o = f(q, o)$$

$$\pi = \text{sort}_{s_o, o \in O}(O).$$

[4]The naming of request and offering is proposed by Paul Kantor.

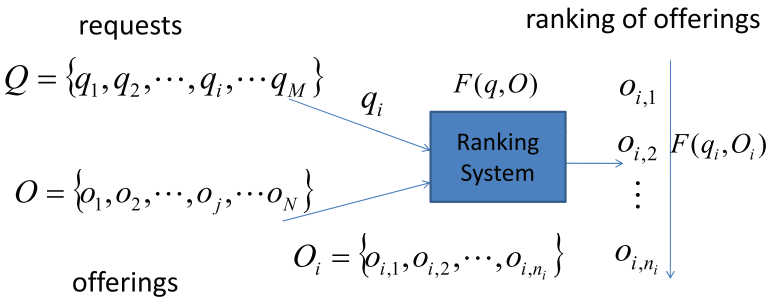

Figure 1.6: Ranking Creation (with Global Ranking Function). Downward arrow represents ranking of objects

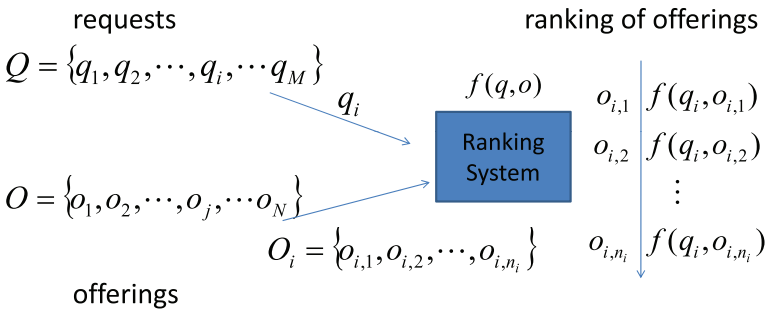

Figure 1.7: Ranking Creation (with Local Ranking Function). Downward arrow represents ranking of objects

We refer to $F(q, O)$ as global ranking function, $f(q, o)$ as local ranking function because the former works on a subset of objects while the latter works on a single object (cf., Figs. 1.6 and 1.7).

1.4 RANKING AGGREGATION

We can also define the general ranking aggregation task. Again, suppose that $\mathcal{Q} = \{q_1, q_2, \cdots, q_i, \cdots, q_M\}$ and $\mathcal{O} = \{o_1, o_2, \cdots, o_j, \cdots, o_N\}$ are a set of requests and a set of offerings, respectively. For an element q of $\mathcal{Q}$ and a subset O of $\mathcal{O}$, there are k ranking lists on O: $\Sigma = \{\pi_i | \pi \in \Pi, i = 1, \cdots, k\}$, where Π is the set of all ranking lists on O. Ranking aggregation takes request q and ranking lists of offerings Σ as input and generates a new ranking list of offerings π as output with ranking function $F(q, \Sigma) : \mathcal{Q} \times \Pi^k \rightarrow \Re^n$

$$S_O = F(q, \Sigma)$$
$$\pi = \text{sort}_{S_O}(O).$$

We usually simply define

$$F(q, \Sigma) = F(\Sigma).$$

That is to say, we assume that the ranking function does not depend on the request.

Ranking aggregation is actually a process of combining multiple ranking lists into a single ranking list, which is better than any of the original ranking lists, as shown in Figure 1.8. The ranking model is a *global* ranking model.

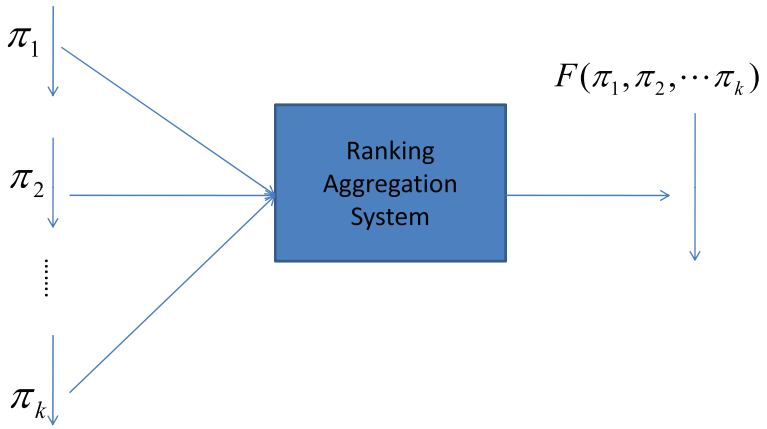

Figure 1.8: Ranking Aggregation. Downward arrows represent rankings of objects

Ranking creation generates ranking based on *features* of request and offerings, while ranking aggregation generates ranking based on *rankings* of offerings. Note that the output of ranking creation can be used as the input of ranking aggregation.

1.5 LEARNING FOR RANKING CREATION

When learning to rank is mentioned, it usually means ranking creation using supervised learning. This is also the main focus of this book. The learning task can be described in the following way. There are two systems: a learning system and a ranking system.

The learning system takes training data as input. The training data consists of requests and their associated ranking lists of offerings. For each request $q_i \in \{q_1, q_2, \cdots, q_m\}$, there is an associated set of offerings $O_i \in \{O_1, O_2, \cdots, O_m\}$ ($O_i = \{o_{i,1}, o_{i,2}, \cdots, o_{i,n_i}\}, i = 1, \cdots, m$), and there is a 'true' ranking list on the offerings $\pi_i \in \{\pi_1, \pi_2, \cdots, \pi_m\}$. The learning system constructs a ranking model (usually, a local ranking model $f(q, o)$) on the basis of the training data.

The ranking system then makes use of the learned ranking model for ranking prediction. Given a new request q_{m+1}, the ranking system receives a subset of offerings O_{m+1}, assigns scores to the offerings using the ranking model, and sorts the offerings in descending order of the scores, obtaining a ranking list π_{m+1}. See Fig. 1.9.

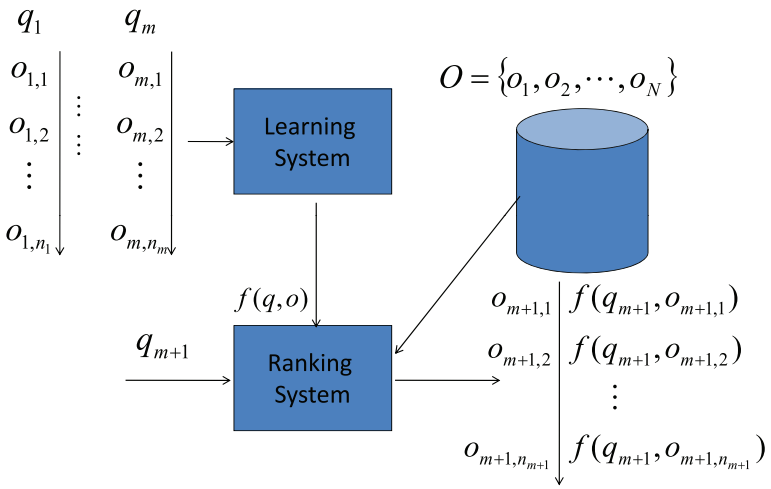

Figure 1.9: Learning for Ranking Creation. Downward arrows represent rankings

Here are the major characteristics of learning for ranking creation.

- Ranking creation: generating a ranking list of offerings based on the request and the offerings
- Feature-based: using features defined on the request and the offerings
- Local ranking model: a local ranking model $f(q, o)$ is utilized
- Supervised learning: the ranking model is usually created by supervised learning

1.6 LEARNING FOR RANKING AGGREGATION

Ranking aggregation can be supervised or unsupervised. In the supervised learning setting, the learning system takes training data as input. The training data consists of requests and their associated ranking lists of offerings. For each request $q_i \in \{q_1, q_2, \cdots, q_m\}$, there is an associated set of offerings $O_i \in \{O_1, O_2, \cdots, O_m\}$ where $O_i = \{o_{i,1}, o_{i,2}, \cdots, o_{i,n_i}\}, i = 1, \cdots, m$. Furthermore, for each O_i, there are k ranking lists on the set: $\Sigma_i = \{\pi_{i,1}, \pi_{i,2}, \cdots, \pi_{i,k}\}$, as well as a 'true' ranking list on the set: π_i. The learning system constructs a ranking model $F(q, \Sigma)$ using the training data.

The ranking system then makes use of the learned ranking model for ranking prediction. Given a new request q_{m+1}, the ranking system receives k ranking lists on the associated set of offerings O_{m+1}: $\Sigma_{m+1} = \{\pi_{m+1,1}, \pi_{m+1,2}, \cdots, \pi_{m+1,k}\}$, assigns scores to the offerings with the ranking model, and sorts the offerings in descending order of the scores, obtaining a ranking list π_{m+1}.

Here are the major characteristics of learning for ranking aggregation.

- Ranking aggregation: generate a ranking list of offerings from multiple ranking lists of the offerings
- Ranking-based: using multiple ranking lists of the offerings
- Global ranking model: a global ranking model $F(q, \Sigma)$ is utilized
- Supervised or unsupervised learning: the ranking model is created by either supervised or unsupervised learning

CHAPTER 2

Learning for Ranking Creation

This chapter gives a general introduction to learning for ranking creation. Ranking creation is aimed at creating a ranking list of offerings based on the features of the offerings and the request, so that 'good' offerings to the request are ranked at the top. Learning for ranking creation is concerned with automatic construction of the ranking model using machine learning techniques.

Recently intensive studies have been conducted on learning for ranking creation due to its importance in practice. Many methods have been proposed and some of the technologies have been successfully applied to applications such as web search.

Hereafter, we take document retrieval (or search) as an example to make the explanation. Without loss of generality, the technologies described here can be applied to other applications.

2.1 DOCUMENT RETRIEVAL AS EXAMPLE

Learning for ranking creation (in general learning to rank) plays a very important role in document retrieval. Traditionally, the ranking model in document retrieval $f(q, d)$ is constructed without training where q stands for a query and d stands for a document. In BM25 [96], the ranking model $f(q, d)$ is represented as a conditional probability distribution $P(r|q, d)$ where r takes on 1 or 0 as value and denotes being relevant or irreverent, q and d denote a query and a document, respectively. In Language Model for IR (LMIR) [86, 124], the ranking model is defined as a conditional probability distribution $P(q|d)$ where q denotes a query and d denotes a document. Both BM25 and LMIR are calculated with the given query and document, and thus no training is needed (only tuning of a few parameters is necessary).

Recently a new trend arises in IR, that is, to employ machine learning techniques to automatically construct the ranking model $f(q, d)$ for document retrieval (cf., [41]). This is motivated by a number of facts. In document retrieval, particularly in web search, there are many signals which can represent relevance. Incorporating such information into the ranking model and automatically constructing the ranking model becomes a natural choice. At web search engines, a large amount of search log data, such as click through data, is accumulated. This also brings a new opportunity of automatically creating the ranking model with low cost by deriving training data from search logs. All these facts have stimulated the research on learning to rank. Actually, learning to rank has become one of the key technologies for modern web search.

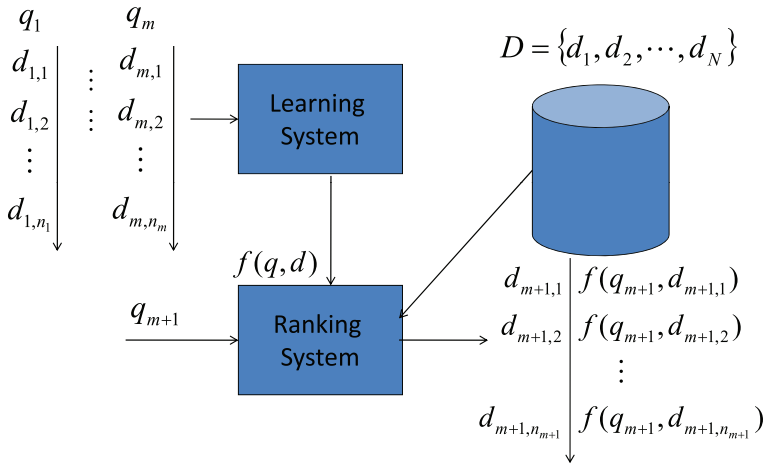

Figure 2.1: Learning to Rank for Document Retrieval

2.2 LEARNING TASK

We describe a number of issues in learning for ranking creation, with document retrieval as an example. These include training and testing processes, training data creation, feature construction, and evaluation. We also discuss the relations between ranking and other tasks such as ordinal classification.

2.2.1 TRAINING AND TESTING

Learning for ranking creation is comprised of training and testing, as a supervised learning task.

The training data contains queries and documents. Each query is associated with a number of documents. The relevance of the documents with respect to the query is also given. As will be explained later, the relevance information can be given in several ways. Here, we take the most widely used approach, and we assume that the relevance of a document with respect to a query is represented by a label. The labels are at several grades (levels). The higher grade a document has, the more relevant the document is.

Suppose that $\mathcal{Q}$ is the query set and $\mathcal{D}$ is the document set. Suppose that $\mathcal{Y} = \{1, 2, \cdots, l\}$ is the label set, where the labels represent grades. There exists a total order between the grades $l \succ l-1 \succ \cdots \succ 1$, where $\succ$ denotes the order relation. Further suppose that $\{q_1, q_2, \cdots, q_m\}$ is the set of queries for training and q_i is the i-th query. $D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$ is the set of documents associated with query q_i and $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,n_i}\}$ is the set of labels associated with query q_i, where n_i denotes the sizes of D_i and $\mathbf{y}_i$; $d_{i,j}$ denotes the j-th document in D_i; and $y_{i,j} \in \mathcal{Y}$ denotes the j-th grade label in $\mathbf{y}_i$, representing the relevance degree of $d_{i,j}$ with respect to q_i. The original training set is denoted as $S = \{(q_i, D_i), \mathbf{y}_i\}_{i=1}^m$.

Table 2.1: Summary of Notations

Notations	Explanations
$\mathcal{Q}$	query set
$\mathcal{D}$	document set
$\mathcal{Y} = \{1, 2, \cdots, l\}$	label set (grade set) with order $\succ$
$S = \{(q_i, D_i), \mathbf{y}_i\}_{i=1}^m$	original training data set
$q_i \in \mathcal{Q}$	i-th query in training data
$D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$	set of documents associated with q_i in training data
$d_{i,j} \in \mathcal{D}$	j-th document in D_i
$\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,n_i}\}$	set of labels on D_i with respect to q_i
$y_{i,j} \in \mathcal{Y}$	label of $d_{i,j}$ with respect to q_i
$x_i = \phi(q_i, d_{i,j})$	feature vector from $(q_i, d_{i,j})$
$\mathbf{x}_i = \Phi(q_i, D_i)$	feature vectors from (q_i, D_i)
Π_i	set of possible rankings on D_i with respect to q_i
$\pi_i \in \Pi_i$	permutation on D_i with respect to q_i
$\pi_i(j)$	rank (position) of j-th document in π_i
$S' = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$	transformed training data set
$f(q, d) = f(x)$	local ranking model
$F(q, D) = F(\mathbf{x})$	global ranking model
$T = \{(q_{m+1}, D_{m+1})\}$	original test data set
$T' = \{\mathbf{x}_{m+1}\}$	transformed test data set

A feature vector $x_{i,j} = \phi(q_i, d_{i,j})$ is created from each query-document pair $(q_i, d_{i,j}), i = 1, 2, \cdots, m; j = 1, 2, \cdots, n_i$, where ϕ denotes the feature functions. That is to say, features are defined as functions of query and document. Letting $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,n_i}\}$, we represent the training data set as $S' = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. We aim to train a local ranking model $f(q, d) = f(x)$ that can assign a score to a given query document pair q and d, or equivalently to a given feature vector x. More generally, we can also consider a global ranking model $F(q, D) = F(\mathbf{x})$. Note that the local ranking model outputs a single score, while the global ranking model outputs a set of scores.

Let the documents in D_i be identified by the integers $\{1, 2, \cdots, n_i\}$. We define permutation (ranking list) π_i on D_i as a bijection from $\{1, 2, \cdots, n_i\}$ to itself. We use Π_i to denote the set of all possible permutations on D_i, use $\pi_i(j)$ to denote the rank (or position) of the j-th document (i.e., $d_{i,j}$) in permutation π_i, and use $\pi^{-1}(j)$ to denote the document at the j-th rank in permutation π_i. Ranking is nothing but to select a permutation $\pi_i \in \Pi_i$ for the given query q_i and the associated set of documents D_i using the scores given by the ranking model $F(q_i, D_i)$ (or $f(q_i, d_i)$).

The test data consists of a new query q_{m+1} and associated set of documents D_{m+1}. $T = \{(q_{m+1}, D_{m+1})\}$. We create feature vector $\mathbf{x}_{m+1}$, use the trained ranking model to assign scores to the documents in D_{m+1}, sort them based on the scores, and give the ranking list of documents as output π_{m+1}.

Table 2.1 gives a summary of notations. Figures 2.2 and 2.3 illustrate the training and testing processes.

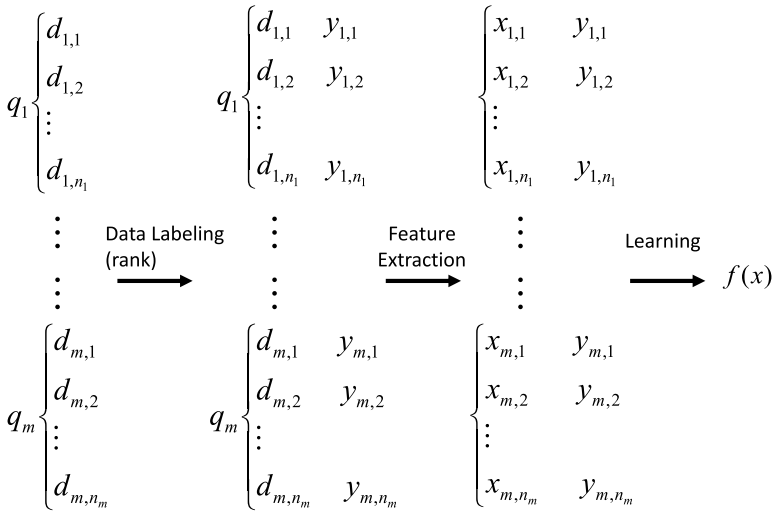

Figure 2.2: Training Process

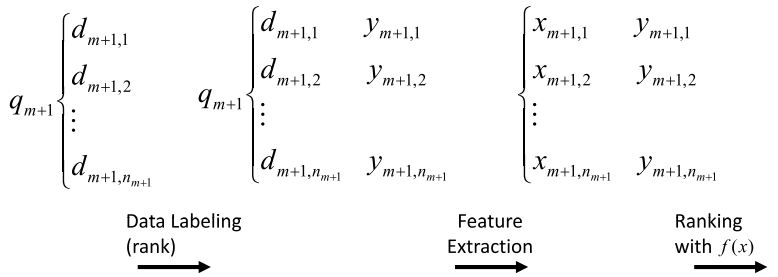

Figure 2.3: Testing Process

The training and testing data is similar to, but different from, the data in conventional supervised learning such as classification and regression. Query and its associated documents form a group. The groups are i.i.d. data, while the instances within a group are not i.i.d. data. A

(local) ranking model is a function of query and document, or equivalently, a function of feature vector derived from query and document.

2.2.2 TRAINING DATA CREATION

Learning for ranking creation is a *supervised* learning task and thus how to create high quality training data is a critical issue.

Ideally, the training data would consist of the perfect ranking lists of documents for each query. In practice, however, such kind of data could be difficult to obtain because the ranking lists must reflect users' average judgments on the relevance of the documents with respect to the queries.

Currently, there are two common ways to create training data. The first one is human labeling, which is widely used in the IR community. First, a set of queries is randomly selected from the query log of a search system. Suppose that there are multiple search systems. Then the queries are submitted to the search systems, and all the top ranked documents are collected. As a result, each query is associated with documents from multiple search systems (it is called the pooling strategy). Human judges are then asked to make relevance judgments on all the query document pairs. Relevance judgments are usually conducted at five levels, for example, perfect, excellent, good, fair, and bad. Human judges make relevance judgments from the viewpoint of average users. For example, if the query is 'Microsoft', and the web page is microsoft.com, then the label is 'perfect'. Furthermore, the Wikipedia page about Microsoft is 'excellent'. A page talking about Microsoft as its main topic will be labeled as 'good,' a page only mentioning Microsoft will be labeled as 'fair,' and a page not relevant to Microsoft will be labeled as 'bad'. Labels representing relevance are then assigned to the query document pairs. The labeling on query document pairs can be performed by multiple judges, and then majority voting can be conducted. Since human labeling is expensive, it is often the case that some query and document pairs are only judged by one single judge. Therefore, how to improve the quality of human relevance judgments becomes an important issue in learning to rank research.

The other way of generating training data is derivation from click through data. Click-through data at a web search engine records clicks on documents by users after they submit queries. Click-through data represents implicit feedbacks on relevance from users and thus is useful for relevance judgments. One method is to use the differences between numbers of clicks on documents to derive preferences (relative relevance) on document pairs [59]. Suppose that for a query three documents A, B, C are returned at the top 1, 2, 3 positions, and users' total numbers of clicks on the documents have been recorded. If there are more clicks on document B than document A, then we can determine that document B is more relevant than document A for this query because users seem to prefer document B to document A, even the latter is ranked lower than the former. Given a ranking list of documents, users tend to click documents on the top, even the documents may not be relevant. As a result, documents on the top tend to have more clicks. This is a phenomenon referred to as 'click bias'. Using the approach above, we can effec-

Table 2.2: Public DataSets for Learning to Rank

Dataset	URL
LEOTR	http://research.microsoft.com/en-us/um/beijing/projects/letor
Microsoft Learning to Rank Dataset	http://research.microsoft.com/en-us/projects/mslr
Yahoo Learning to Rank Challenge	http://webscope.sandbox.yahoo.com

tively deal with click bias because it derives preference pairs of documents by looking at 'skips' of higher ranked documents. Therefore, this method can generate preference pairs of documents as training data for learning to rank. Within each document pair, one document is regarded more relevant than the other with respect to the query. See also [93, 94].

The two approaches above both have pros and cons. It is very hard to maintain the quality of data, when it is created by humans. Human judges are prone to errors, and their understanding on relevance also has limitations because they are not query owners. Furthermore, manual data labeling is also costly. In contrast, derivation of training data from click-through data is of low cost and the data may also represent real users' judgments. The shortcoming of this approach is that click-through data is noisy and is only available for head queries (high frequency queries).

Table 2.2 gives a list of publically available datasets for learning to rank research. They are all datasets created by the first approach.

2.2.3 FEATURE CONSTRUCTION

The ranking model $f(q, d)$ is in fact defined as $f(x)$ where x is a feature vector based on q and d. That is to say, the ranking model is feature based. That is the reason that the ranking model has generalization ability. Specifically, it is trained from a small number of queries and their associated documents but is applicable to any other queries and their associated documents. As in other machine learning tasks, the performance of learning highly depends on the effectiveness of the features used. How to define useful features thus is a critically important problem.

In web search, BM25 and PageRank are widely used ranking features. In fact, both can be viewed as unsupervised ranking models. At the early stage of web search, the final ranking model was usually simply defined as a linear combination of BM25 and PageRank, or something similar. Later, more and more features have been developed. That is also the reason that a more general and principled learning approach is needed in ranking model construction. We give the definitions of BM25 and PageRank.

BM 25 is a probabilistic model representing the relevance of document d to query q [96]. It actually looks at the matching degree between the query terms and document terms and utilizes

the numbers of occurrence of query terms in the document to represent relevance. Specifically, BM25 of query q and document d is calculated as

$$\text{BM25}(q,d) = \sum_{w \in q \cap d} idf(w) \frac{(k+1)tf(w)}{tf(w) + k((1-b) + b\frac{dl}{avgdl})},$$

where w denotes a word in d and q, $tf(w)$ denotes the frequency of w in d, $idf(w)$ denotes the inverse document frequency of w, dl denotes the length of d, $avgdl$ denotes the average document length, and k and b are parameters.

Page Rank represents the importance of web page [84]. It views the web as a directed graph in which pages are vertices and hyperlinks are directed edges, defines a Markov process on the web graph, and views the stationary distribution (Page Rank) of the Markov process as scores of page importance. Page Rank of web page d is defined as $P(d)$

$$P(d) = \alpha \sum_{d_i \in M(d)} \frac{P(d_i)}{L(d_i)} + (1-\alpha)\frac{1}{N},$$

where $P(d)$ is the probability of visiting page d, $P(d_i)$ is the probability of visiting page d_i, $M(d)$ is the set of pages linked to d, $L(d_i)$ is the number of outlinks from d_i, N is the total number of nodes on the graph, and α is a weight.

There are other features utilized in web search. Table 2.3 gives some examples, which have been verified to be effective in web search. They include both query-document matching features and document features, representing relevance of document to query and importance of document, respectively.

Web pages usually contain a number of fields (metadata streams) such as title, anchor texts, URL, extracted title [52, 53], and associated queries in click-through data [3]. One can define query-document matching features, for example, BM25, for each field of the web page, and thus exploit a number of features in the same type.

2.2.4 EVALUATION

Evaluation on the performance of a ranking model is carried out by comparison between the ranking lists output by the model and the ranking lists given as ground truth. Several evaluation measures are widely used in IR and other fields. These include NDCG (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain) [55], MAP (Mean Average Precision) [110], WTA (Winners Take All), MRR (Mean Reciprocal Rank), and Kendall's Tau.

Given query q_i and associated documents D_i, suppose that π_i is the ranking list (permutation) on D_i and $\mathbf{y}_i$ is the set of labels (grades) of D_i. DCG measures the goodness of the ranking list with the labels. Specifically, DCG at position k for q_i is defined:

$$DCG(k) = \sum_{j:\pi_i(j) \le k} G(j)D(\pi_i(j)),$$

Table 2.3: Example Features of Learning to Rank for Web Search

Feature	Type	Explanation	Reference
Number of occurrences	Matching	number of times query exactly occurs in title, anchor, URL, extracted title, associated query, and body	
BM25	Matching	BM25 scores on title, anchor, URL, extracted title, associated query, and body	[96]
N-gram BM25	Matching	BM25 scores of n-grams on title, anchor, URL, extracted title, associated query, and body	[120]
Edit Distance	Matching	edit distance scores between query and title, anchor, URL, extracted title, associated query, and span in body (minimum length of text segment including all query words [101])	Our unpublished work
Number of in-links	Document	number of in-links to the page	
PageRank	Document	importance score of page calculated on web link graph	[84]
Number of clicks	Document	number of clicks on the page in search log	
BrowseRank	Document	importance score of page calculated on user browsing graph	[76]
Spam score	Document	likelihood of spam page	[47]
Page quality score	Document	likelihood of low quality page	[10]

where $G(\cdot)$ is a gain function and $D(\cdot)$ is a position discount function. Note that $\pi_i(j)$ denotes the position of $d_{i,j}$ in π_i. Therefore, the summation is taken over the top k positions in ranking list π_i[1]. DCG represents the cumulative gain of accessing the information from position one to position k with discounts on the positions. NDCG is normalized DCG, and NDCG at position k for q_i is defined:

$$NDCG(k) = DCG_{max}^{-1}(k) \sum_{j:\pi_i(j)\leq k} G(j)D(\pi_i(j)),$$

[1]Here, the definition of NDCG (or DCG) are formulated based on the indices of documents. It is also possible to define NDCG (or DCG) based on the indices of positions.

where $DCG_{max}(k)$ is the normalizing factor and is chosen such that a perfect ranking π_i^*'s NDCG score at position k is 1. In a perfect ranking, the documents with higher grades are ranked higher. Note that there can be multiple perfect rankings for a query and associated documents.

The gain function is normally defined as an exponential function of grade. That is to say, satisfaction of accessing information exponentially increases when grade of relevance increases. For example,

$$G(j) = 2^{y_{i,j}} - 1,$$

where $y_{i,j}$ is the label (grade) of $d_{i,j}$ in ranking list π_i. The discount function is normally defined as a logarithmic function of position. That is to say, satisfaction of accessing information logarithmically decreases when position of information access increases.

$$D(\pi_i(j)) = \frac{1}{\log_2(1 + \pi_i(j))},$$

where $\pi_i(j)$ is the position of $d_{i,j}$ in ranking list π_i.

Hence, DCG and NDCG at position k for q_i become

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))},$$

$$NDCG(k) = DCG_{max}^{-1}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))}.$$

DCG and NDCG of the whole ranking list for q_i become

$$DCG = \sum_{j:\pi_i(j)\leq n_i} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))},$$

$$NDCG = DCG_{max}^{-1} \sum_{j:\pi_i(j)\leq n_i} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))}.$$

DCG and NDCG values are further averaged over queries ($i = 1, \cdots, m$).

Table 2.4 gives examples of calculating NDCG values of two ranking lists. NDCG (DCG) has the effect of giving high scores to the ranking lists in which relevant documents are ranked high. See the examples in Table 2.4. For the perfect rankings, the NDCG value at each position is always one, while for imperfect rankings, the NDCG values are less than one.

MAP is another measure widely used in IR. In MAP, it is assumed that the grades of relevance are at two levels: 1 and 0. Given query q_i, associated documents D_i, ranking list π_i on D_i, and labels $\mathbf{y}_i$ of D_i, Average Precision for q_i is defined:

$$AP = \frac{\sum_{j=1}^{n_i} P(j) \cdot y_{i,j}}{\sum_{j=1}^{n_i} y_{i,j}},$$

Table 2.4: Example of NDCG

Perfect ranking	Formula	Explanation
(3, 3, 2, 2, 1, 1, 1)		grades: 3,2,1
(7, 7, 3, 3, 1, 1, 1)	$2^{y_{i,j}} - 1$	gains
(1, 0.63, 0.5, $\cdots$)	$1/\log(\pi_i(j)+1)$	position discounts
(7, 11.41, 12.91, $\cdots$)	$\sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}}-1}{\log(\pi_i(j)+1)}$	DCG scores
(1/7, 1/11.41, 1/12.91,$\cdots$)	$DCG^{-1}_{max}(k)$	normalizing factors
(1,1,1,$\cdots$)	$NDCG(k)$	NDCG scores
Imperfect ranking	**Formula**	**Explanation**
(2, 3, 2, 3, 1, 1, 1)		grades: 3,2,1
(3, 7, 3, 7, 1, 1, 1)	$2^{y_{i,j}} - 1$	gains
(1, 0.63, 0.5, $\cdots$)	$1/\log(\pi_i(j)+1)$	position discounts
(3, 7.41, 8.91, $\cdots$)	$\sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}}-1}{\log(\pi_i(j)+1)}$	DCG scores
(1/7, 1/11.41, 1/12.91,$\cdots$)	$DCG^{-1}_{max}(k)$	normalizing factors
(0.43, 0.65, 0.69, $\cdots$)	$NDCG(k)$	NDCG scores

where $y_{i,j}$ is the label (grade) of $d_{i,j}$ and takes on 1 or 0 as value, representing being relevant or irrelevant. $P(j)$ for query q_i is defined:

$$P(j) = \frac{\sum_{k:\pi_i(k)\leq\pi_i(j)} y_{i,k}}{\pi_i(j)},$$

where $\pi_i(j)$ is the position of $d_{i,j}$ in π_i. $P(j)$ represents the precision until the position of $d_{i,j}$ for q_i. Note that labels are either 1 or 0, and thus precision (i.e., ratio of label 1) can be defined. Average Precision represents averaged precision over all the positions of documents with label 1 for query q_i.

Average Precision values are further averaged over queries to become Mean Average Precision (MAP). Table 2.5 gives an example of calculating the AP value of one ranking list.

Kendall's Tau is a measure proposed in statistics. It is defined on two ranking lists: one is the ranking list by the ranking model, and the other is by the ground truth. Kendall's Tau of ranking list π_i with respect to ground truth π_i^* is defined:

$$T_i = \frac{2c_i}{\frac{1}{2}n_i(n_i-1)} - 1,$$

where c_i denotes the number of concordant pairs between the two lists, and n_i denotes the length of the two lists. For example, Kendall's Tau between two ranking lists: (A,B,C) and (C,A,B) is as

Table 2.5: Example of MAP

Perfect ranking	Formula	Explanation
(1, 0, 1, 1, 0, 0, 0)		Ranks:1,0
(1, -, 0.67, 0.75, –, –, –)	$P(j)$	Precision at position j with label 1
0.81	AP	Average Precision

follows.

$$T_i = \frac{2 \times 1}{3} - 1 = -\frac{1}{3}.$$

Kendall's Tau has values between -1 and $+1$. If the two ranking lists are exactly the same, then it is $+1$. If one ranking list is in reverse order of the other, then it is -1. It is easy to verify Kendall's Tau can also be written as

$$T_i = \frac{c_i - d_i}{\frac{1}{2}n_i(n_i - 1)},$$

where d_i denotes the number of discordant pairs between the two lists.

2.2.5 RELATIONS WITH OTHER LEARNING TASKS

There are some similarities between ranking and other machine learning tasks such as classification, regression, and ordinal classification (ordinal regression). Differences between them also exist, however. That is why learning to rank is also an interesting research problem in machine learning.

In classification, the input is a feature vector $x \in \Re^d$, and the output is a label $y \in \mathcal{Y}$, representing a class where $\mathcal{Y}$ is the set of class labels, and the goal of learning is to learn a classifier $f(x)$ which can determine the class label y of a given feature vector x.

In regression, the input is a feature vector $x \in \Re^d$, the output is a real number $y \in \Re$, and the goal of learning is to learn a function $f(x)$ which can determine the real number y of a given feature vector x.

Ordinal classification (or ordinal regression) [31, 71, 98] is close to ranking, but it is also different. The input is a feature vector $x \in \Re^d$, the output is a label $y \in \mathcal{Y}$, representing a grade where $\mathcal{Y}$ is a set of grade labels. The goal of learning is to learn a model $f(x)$ which can determine the grade label y of a given feature vector x. The model first calculates the score $f(x)$, and then it decides the grade label y using a number of thresholds. Specifically, the model segments the real number axis into a number of intervals and assigns to each interval a grade. It then takes the grade of the interval which $f(x)$ falls into as the grade of x.

In ranking, one cares more about accurate ordering of objects (offerings), while in ordinal classification, one cares more about accurate ordered-categorization of objects. A typical example of ordinal classification is product rating. For example, given the features of a movie, we are to

assign a number of stars (ratings) to the movie. In that case, correct assignment of number of stars is critical. A typical example of ranking is document retrieval. In document retrieval, given a query, the objective is to give a right ranking on the documents, although sometimes training data and testing data are labeled at multiple grades as in ordinal classification. The number of documents to be ranked can vary from query to query. There are queries for which more relevant documents are available in the collection, and there are also queries for which only weakly relevant documents are available.

As will be seen later, ranking can be approximated by classification, regression, and ordinal classification.

2.3 LEARNING APPROACHES

Learning to rank, particularly learning for ranking creation, has been intensively studied recently. The proposed methods can be categorized as the pointwise approach, pairwise approach, and listwise approach. There are also methods which may not belong to any of the approaches, for example, query dependent ranking [43] and multiple nested ranking [78].

The pointwise and pairwise approaches transform the ranking problem into classification, regression, and ordinal classification. The listwise approach takes ranking lists of objects as instances in learning and learns the ranking model based on ranking lists. The main differences among the approaches lie in the loss functions employed.

It is observed that the listwise approach and pairwise approach usually outperform the pointwise approach. In the recent Yahoo Learning to Rank Challenge, LambdaMART, which belongs to the listwise approach, achieved the best performance.

The methods can also be categorized based on the learning techniques which they employ. They include SVM techniques, Boosting techniques, Neural Net techniques, and others.

Table 2.6 gives a summary of the existing methods. Each of them will be described hereafter.

2.3.1 POINTWISE APPROACH

In the pointwise approach, the ranking problem (ranking creation) is transformed to classification, regression, or ordinal classification, and existing methods for classification, regression, or ordinal classification are applied. Therefore, the group structure of ranking is ignored in this approach.

More specifically, the pointwise approach takes training data in Figure 2.2 as input. It ignores the group structure and combines all the groups together $(x_{i,1}, y_{i,1}), \cdots, (x_{i,n_i}, y_{i,n_i}), i = 1, \cdots, m$. The training data becomes typical supervised learning data (representing mapping from x to y). When we take y as a class label, real number, and grade label, then the problem becomes classification, regression, and ordinal classification, respectively. We can then employ existing methods for classification, regression, or ordinal classification to perform the learning task.

Suppose that the learned model $f(x)$ outputs real numbers. Then, given a query, we can use the model to rank documents (sort documents according to the scores given by the model).

Table 2.6: Categorization of Learning to Rank Methods

	SVM	Boosting	Neural Net	Others
Pointwise	OC SVM [98]	McRank [71]		Prank [31] Subset Ranking [30]
Pairwise	Ranking-SVM [50] IR SVM [14]	RankBoost [39] GBRank [126]	RankNet [12] Frank [104]	
Listwise	SVM-MAP [122] PermuRank [121]	AdaRank [119] LambdaMART [113]	ListNet [15] ListMLE [115] LambdaRank [13]	SoftRank [102] AppRank [87]

The loss function in learning is pointwise in the sense that it is defined on a single object (feature vector). Table 2.7 summarizes the main characteristics of the pointwise approach. The pointwise approach includes Prank, OC SVM, McRank, and Subset Ranking. Chapter 4 explains Prank, OC SVM, and McRank in details. (See also [23–25].)

Crammer & Singer [31] have studied ordinal classification, which is to assign a grade to a given object. The grades can be used for ranking, and thus their method can also be viewed as a method for ranking. Crammer & Singer propose a simple and efficient online algorithm, called Prank. Given training data, Prank iteratively learns a number of parallel Perceptron models, and each model separates two neighboring grades. The authors have also conducted analysis on Prank in terms of mistake bound.

Shashua & Levin [98] propose a large margin approach to ordinal classification, referred to as OC SVM (Ordinal Classification SVM) in this book. In their method, they also try to learn parallel hyperplanes to separate neighboring grades, but their machinery is the Large Margin Principle. They consider two ways of defining the margin. The first assumes that the margins between the neighboring grades are the same and the margin is maximized in learning. The second allows different margins for different neighboring grades, and the sum of margins is maximized.

Li et al. [71] cast the ranking problem as multi-class classification and propose the McRank algorithm. The authors are motivated by the fact that the errors in ranking based on DCG is bounded by the errors in multi-class classification. They learn and exploit a classification model

Table 2.7: Characteristics of Pointwise Approach

Pointwise Approach (Classification)		
	Learning	**Ranking**
Input	feature vector	feature vectors
	x	$\mathbf{x} = \{x_i\}_{i=1}^n$
Output	category	ranking list
	$y = \text{classifier}(f(x))$	$\text{sort}(\{f(x_i)\}_{i=1}^n)$
Model	$\text{classifier}(f(x))$	ranking model $f(x)$
Loss	classification loss	ranking loss
Pointwise Approach (Regression)		
	Learning	**Ranking**
Input	feature vector	feature vectors
	x	$\mathbf{x} = \{x_i\}_{i=1}^n$
Output	real number	ranking list
	$y = f(x)$	$\text{sort}(\{f(x_i)\}_{i=1}^n)$
Model	regression model $f(x)$	ranking model $f(x)$
Loss	regression loss	ranking loss
Pointwise Approach (Ordinal Classification)		
	Learning	**Ranking**
Input	feature vector	feature vectors
	x	$\mathbf{x} = \{x_i\}_{i=1}^n$
Output	ordered category	ranking list
	$y = \text{threshold}(f(x))$	$\text{sort}(\{f(x_i)\}_{i=1}^n)$
Model	$\text{threshold}(f(x))$	ranking model $f(x)$
Loss	ordinal classification loss	ranking loss

that can assign to an object probabilities of being members of grades. They then calculate the expected grades of objects and use the expected grades to rank objects. The classification model is learned by using the Gradient Tree Boosting algorithm.

Cossock & Zhang [30] have developed the Subset Ranking algorithm. They first consider using DCG as evaluation measure. Since minimization of the loss function based on DCG is a non-convex optimization problem, they instead manage to minimize a surrogate loss function which is an upper bound of the original loss function. The surrogate loss function is defined based on regression errors. Therefore, the original ranking problem can be solved as a regression problem. Cossock and Zhang then derive a learning method for the task on the basis of regression. They have also investigated the generalization ability and the consistency of the learning method.

Table 2.8: Characteristics of Pairwise Approach

Pairwise Approach (Classification)		
	Learning	**Ranking**
Input	feature vectors $x^{(1)}, x^{(2)}$	feature vectors $\mathbf{x} = \{x_i\}_{i=1}^n$
Output	pairwise classification classifier($f(x^{(1)}) - f(x^{(2)})$)	ranking list sort($\{f(x_i)\}_{i=1}^n$)
Model	classifier($f(x)$)	ranking model $f(x)$
Loss	pairwise classification loss	ranking loss
Pairwise Approach (Regression)		
	Learning	**Ranking**
Input	feature vectors $x^{(1)}, x^{(2)}$	feature vectors $\mathbf{x} = \{x_i\}_{i=1}^n$
Output	pairwise regression $f(x^{(1)}) - f(x^{(2)})$	ranking list sort($\{f(x_i)\}_{i=1}^n$)
Model	regression model $f(x)$	ranking model $f(x)$
Loss	pairwise regression loss	ranking loss

2.3.2 PAIRWISE APPROACH

In the pairwise approach, ranking is transformed into pairwise classification or pairwise regression. For example, a classifier for classifying the ranking orders of document pairs can be created and employed in ranking of documents. In the pairwise approach, the group structure of ranking is also ignored.

Specifically, the pairwise approach takes training data in Figure 2.2 as input. From the labeled data of query q_i, $(x_{i,1}, y_{i,1}), \cdots, (x_{i,n_i}, y_{i,n_i}), i = 1, \cdots, m$, it creates preference pairs of feature vectors (documents). For example, if $x_{i,j}$ has a higher grade than $x_{i,k}$ $(y_{i,j} > y_{i,k})$, then $x_{i,j} \succ x_{i,k}$ becomes a preference pair, which means that $x_{i,j}$ is ahead of $x_{i,k}$. The preference pairs can be viewed as instances and labels in a new classification problem. For example, $x_{i,j} \succ x_{i,k}$ is a positive instance. We can employ exiting classification methods to train a classifier to conduct the classification. The classifier $f(x)$ can then be used in ranking. More precisely, documents are assigned scores by $f(x)$ and sorted by the scores. Training of a good model for ranking is realized by training of a good model for pairwise classification. The loss function in learning is pairwise because it is defined on a pair of feature vectors (documents). Table 2.8 summarizes the main characteristics of the pairwise approach.

Note that a perfect ranking implies perfect classification on all preference pairs and an incorrect ranking implies existence of incorrect classification on preference pairs. Therefore, learning of a ranking model is equivalent to learning of a pairwise classification model.

The pairwise approach includes Ranking SVM, RankBoost, RankNet, IR SVM, GBRank, and Frank. Chapter 4 introduces Ranking SVM, IR SVM, GBRank, and RankNet in details. (See also [37, 82, 92, 105, 127]).

Ranking SVM is one of the first learning to rank methods, proposed by Herbrich et al. [50]. The basic idea is to formalize the ranking problem as pairwise classification and employ the SVM technique to perform the learning task. The objects in different grades are used to generate preference pairs (which object is ahead of which) and viewed as data representing mappings from object pairs to orders. Herbrich et al. show that when the classifier is a linear model (or a linear model after applying the kernel trick), it can be directly employed in ranking.

The RankBoost Algorithm has been developed by Freund et al. [39]. In their work, they present a formal framework for ranking, namely the problem of learning to rank objects by combining a number of ranking features. They then propose the RankBoost algorithm based on the Boosting technique. They show theoretical results describing the algorithm's behavior both on the training data and the test data. An efficient implementation of the algorithm is also given.

Burges et al. [12] propose the RankNet algorithm, which is also based on pairwise classification like Ranking SVM and RankBoost. The major difference lies in that it employs Neural Network as ranking model and uses Cross Entropy as loss function. Burges et al. also show the good properties of Cross Entropy as loss function in ranking. Their method employs Gradient Descent to learn the optimal Neural Network model.

The advantage of the above methods is that existing learning techniques on classification and regression can be directly applied. The disadvantage is that the goal of learning may not be consistent with that of prediction. In fact, evaluation of ranking is usually conducted based on measures such as NDCG, which are defined on list of objects. In contrast, training in the pairwise approach is driven by enhancement of accuracy of pairwise classification or pairwise regression, which is defined on pairs of objects.

Cao et al. [14] propose employing cost sensitive Ranking SVM to overcome the shortcoming. The authors point out that there are two factors one must consider in ranking for document retrieval. First, correctly ranking documents on the top of list is crucial. Second, the number of relevant documents can vary from query to query. They modify the hinge loss function in Ranking SVM to deal with the problems, and then they formalize the learning task as cost sensitive Ranking SVM. The method is often referred to as IR SVM. Gradient Descent and Quadratic Programming are respectively employed to solve the optimization problem in learning.

Zheng et al. [126] suggest employing pairwise regression for ranking. The GB (Gradient Tree Boosting) Rank algorithm is proposed. They first introduce a regression framework for ranking, which employs a pairwise regression loss function. They then propose a novel optimization method based on the Gradient Tree Boosting algorithm to iteratively minimize the loss function.

They use two types of relevance judgments to derive training data: absolute relevance judgments by human judges and relative relevance judgments extracted from click through data.

The Frank method proposed by Tsai et al. [104] is based on a similar motivation as IR SVM. The authors propose employing a novel loss function in RankNet. The loss function named 'Fidelity Loss' not only retains the good properties of the Cross Entropy loss, but also possesses some desirable new properties. Particularly, Fidelity Loss is bounded between 0 and 1, which makes the learning method more robust against noises. In Frank, a Neural Network model is trained as ranking model using Gradient Descent.

2.3.3 LISTWISE APPROACH

The listwise approach addresses the ranking problem in a more natural way. Specifically, it takes ranking lists as instances in both learning and prediction. The group structure of ranking is maintained and ranking evaluation measures can be more directly incorporated into the loss functions.

More specifically, the listwise approach takes training data in Figure 2.2 as input. It views the labeled data $(x_{i,1}, y_{i,1}), \cdots, (x_{i,n_i}, y_{i,n_i})$ associated with query q_i as one instance. The approach learns a ranking model $f(x)$ from the training data that can assign scores to feature vectors (documents) and rank the feature vectors using the scores, such that feature vectors with higher grades are ranked higher. This is a new problem for machine learning and conventional techniques in machine learning cannot be directly applied. Recently, advanced learning to rank techniques have been developed, and many of them belong to the listwise approach. Table 2.9 summarizes the main characteristics of the listwise approach.

The listwise approach includes ListNet, ListMLE, AdaRank, SVM MAP, Soft Rank, AppRank, LambdaRank, and LambdaMART. Chapter 4 describes ListNet, ListMLE, AdaRank, SVM MAP, Soft Rank, LambdaRank, and LambdaMART in details. (See also [19, 91, 106, 107]).

Table 2.9: Characteristics of Listwise Approach

Listwise Approach		
	Learning	**Ranking**
Input	feature vectors $\mathbf{x} = \{x_i\}_{i=1}^n$	feature vectors $\mathbf{x} = \{x_i\}_{i=1}^n$
Output	ranking list $\text{sort}(\{f(x_i)\}_{i=1}^n)$	ranking list $\text{sort}(\{f(x_i)\}_{i=1}^n)$
Model	ranking model $f(x)$	ranking model $f(x)$
Loss	listwise loss function	ranking loss

Cao et al. [15] point out the importance of employing the listwise approach to ranking, in which lists of objects are treated as 'instances'. They propose using the Luce-Plackett model to calculate the permutation probability or top k probability of list of objects. The ListNet algorithm based on the idea is then developed. In the method, a Neural Network model is employed as model, and KL divergence is employed as loss function. The permutation probability or top k probability of a list of objects is calculated by the Luce-Plackett model. KL divergence measures the difference between the learned ranking list and the true ranking list using their permutation probability distributions or top k probability distributions. Gradient Descent is utilized as optimization algorithm. Xia et al. [115] extend ListNet to ListMLE, in which log likelihood is defined as loss function. The learning of ListMLE is equivalent to Maximum Likelihood Estimation on the parameterized Luce-Plackett model.

The evaluation measures in applications such as those in IR are defined on list of objects. Ideally, a learning algorithm would train a ranking model that could directly optimize the evaluation measures. Another group of listwise methods try to directly optimize the evaluation measures in learning. These include AdaRank developed by Xu & Li [119]. AdaRank minimizes a loss function directly defined on an evaluation measure by using the Boosting technique. It repeatedly constructs 'weak rankers' on the basis of reweighted training data and finally linearly combines the weak rankers for ranking prediction. AdaRank is a very simple and efficient learning to rank algorithm.

The algorithm of SVM MAP proposed by Yue et al. [122] also considers direct optimization of evaluation measures, particularly, MAP used in IR. SVM MAP is an SVM learning algorithm that can efficiently find a globally optimal solution to minimization of an upper bound of the loss function based on MAP. Xu et al. [121] show that one can extend the idea to derive a group of algorithms. The key idea of these algorithms is to introduce loss functions based on IR evaluation measures (defined on list of objects), consider different upper bounds of the loss functions, and employ the SVM technique to optimize the upper bounds. Different upper bounds and different optimization techniques can lead to different algorithms including one called PermuRank.

Another listwise approach tries to approximate the evaluation measures. SoftRank is a representative algorithm of such an approach. The major challenge for direct optimization of evaluation measures is that they are not directly optimizable due to the nature of the loss functions. Taylor et al. [102] present a soft (approximate) way of calculating the distribution of ranks of objects. With an approximated rank distribution, it is possible to approximately calculate evaluation measures such as NDCG. They show that SoftRank is a very effective way of optimizing evaluation measures.

Qin et al. [87] propose a general framework for direct optimization of IR measures in learning to rank. In the framework, the IR measures such as NDCG and MAP are approximated as surrogate functions, and the surrogate functions are then optimized. The key idea of the approach is as follows. The difficulty in directly optimizing IR measures lies in that the measures are rank based and thus are non-continuous and non-differentiable with respect to the score output by the

ranking function. The proposed approach approximates the ranks of documents by a continuous and differentiable function. An algorithm based on the framework is developed, which is called AppRank.

LambdaRank developed by Burges et al. [13] addresses the challenge by using an implicit listwise loss function[2]. Specifically, LambdaRank considers learning the optimal ranking model by optimizing the loss function with Gradient Descent. In fact, it only explicitly defines the gradient function of the loss function, referred to as Lambda Function. As example, LambdaRank employs a Neural Network model. Burges et al. also give necessary and sufficient conditions for the implicit cost function to be convex.

Wu et al. [11, 113] propose a new method called LambdaMART, using Boosting and the Lambda function in LambdaRank. It employs the MART (Multiple Additive Regression Trees) algorithm (also known as Gradient Tree Boosting) [40] to learn a boosted regression tree as ranking model. MART is an algorithm conducting Gradient Descent in the functional space. LambdaMART specifically employs the Lambda function as the gradients in MART. Wu et al. have verified that the efficiency of LambdaMART is significantly better than LambdaRank, and the accuracy of it is also higher.

2.3.4 EVALUATION RESULTS

According to the previous studies, the listwise method and the pairwise approach usually work better than the pointwise approach. As in other machine learning tasks, there is no single method that can always outperforms the other methods. This is a general trend on the learning to rank methods.

Tables 2.10-2.16 give the experimental results on a number of methods in terms of NDCG on the LETOR datasets [88]. The LETOR data sets are benchmark data for learning to rank, derived from TREC data by researchers in Microsoft Research [3]. One can get a rough sense about the performances achieved by the methods.

In the Yahoo Learning to Rank Challenge [4], the pairwise approach of LambdaMART achieved the best performance. In fact, the accuracies by the top performers were very close to each other.

[2]The general formulation of LambdaRank is listwise, but its implementation in practice is usually pairwise.
[3]http://research.microsoft.com/en-us/um/beijing/projects/letor/
[4]http://learningtorankchallenge.yahoo.com/

Table 2.10: NDCG on TD2003 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.32	0.31	0.30	0.33
Ranking SVM	0.32	**0.34**	**0.36**	**0.35**
RankBoost	0.28	0.32	0.31	0.31
FRank	0.30	0.27	0.25	0.27
ListNet	0.40	0.34	0.34	**0.35**
AdaRank-MAP	0.26	0.31	0.30	0.31
AdaRank-NDCG	**0.36**	0.29	0.29	0.30
SVMMAP	0.32	0.32	0.33	0.33

Table 2.11: NDCG on TD2004 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.36	0.34	0.33	0.30
Ranking SVM	0.41	0.35	0.32	0.31
RankBoost	**0.51**	**0.43**	**0.39**	**0.35**
FRank	0.49	0.39	0.36	0.33
ListNet	0.36	0.36	0.33	0.32
AdaRank-MAP	0.41	0.38	0.36	0.33
AdaRank-NDCG	0.43	0.37	0.35	0.32
SVMMAP	0.29	0.30	0.30	0.29

Table 2.12: NDCG on NP2003 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.45	0.61	0.64	0.67
Ranking SVM	0.58	**0.77**	0.78	0.80
RankBoost	**0.60**	0.76	0.78	**0.81**
FRank	0.54	0.73	0.76	0.78
ListNet	0.57	0.76	0.78	0.80
AdaRank-MAP	0.58	0.73	0.75	0.76
AdaRank-NDCG	0.56	0.72	0.74	0.77
SVMMAP	0.56	**0.77**	**0.79**	0.80

Table 2.13: NDCG on NP2004 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.37	0.56	0.61	0.65
Ranking SVM	0.51	0.75	**0.80**	**0.81**
RankBoost	0.43	0.63	0.65	0.69
FRank	0.48	0.64	0.69	0.73
ListNet	**0.53**	**0.76**	**0.80**	**0.81**
AdaRank-MAP	0.48	0.70	0.73	0.75
AdaRank-NDCG	0.51	0.67	0.71	0.74
SVMMAP	0.52	0.75	0.79	**0.81**

Table 2.14: NDCG on HP2003 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.42	0.51	0.55	0.59
Ranking SVM	0.69	0.77	0.80	0.81
RankBoost	0.67	0.79	0.80	0.82
FRank	0.65	0.74	0.78	0.80
ListNet	0.72	**0.81**	**0.83**	**0.84**
AdaRank-MAP	**0.73**	**0.81**	**0.83**	**0.84**
AdaRank-NDCG	0.71	0.79	0.80	0.81
SVMMAP	0.71	0.78	0.79	0.80

Table 2.15: NDCG on HP2004 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.39	0.58	0.61	0.65
Ranking SVM	0.57	0.71	0.75	0.77
RankBoost	0.51	0.70	0.72	0.74
FRank	0.60	0.73	0.75	0.76
ListNet	0.60	0.72	0.77	0.78
AdaRank-MAP	0.61	**0.82**	**0.83**	**0.83**
AdaRank-NDCG	0.59	0.75	0.79	0.81
SVMMAP	**0.63**	0.75	0.80	0.81

Table 2.16: NDCG on OHSUMED Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Regression	0.45	0.44	0.43	0.41
Ranking SVM	0.50	0.42	0.42	0.41
RankBoost	0.46	0.46	0.45	0.43
FRank	0.53	**0.48**	0.46	0.44
ListNet	0.53	0.47	0.44	0.44
AdaRank-MAP	**0.54**	0.47	0.46	0.44
AdaRank-NDCG	0.53	**0.48**	**0.47**	**0.45**
SVMMAP	0.52	0.47	0.45	0.43

CHAPTER 3

Learning for Ranking Aggregation

This chapter gives a general introduction to learning for ranking aggregation. Ranking aggregation is aimed at combining multiple rankings into a single ranking, which is better than any of the original rankings in terms of an evaluation measure. Learning for ranking aggregation is about building a ranking model for ranking aggregation using machine learning techniques.

Hereafter, we take meta-search as an example to make the explanation. Without loss of generality, the technologies described can be applied to other applications.

3.1 LEARNING TASK

In meta-search, the query from the user is sent to multiple search systems, and the ranking lists from the search systems are then combined and presented to the user in a single ranking list. Since the ranking lists from individual search systems may not be accurate enough, meta-search actually takes a majority voting over search ranking lists. The question is then how to effectively perform the majority voting. Here we call the rankings from individual search systems basic rankings, and the ranking in meta search final ranking.

Learning for ranking aggregation can be performed either as unsupervised learning or supervised learning. In traditional IR, ranking aggregation is usually based on unsupervised learning. Recently, supervised methods for ranking aggregation have also been proposed.

In supervised learning for ranking aggregation, the training data contains queries, their associated documents and basic rankings on the documents, as well as the corresponding final rankings. The testing data includes query, associated documents, and basic rankings on the documents.

Suppose that $\mathcal{Q}$ is the query set, and $\mathcal{D}$ is the document set. Further suppose that $\{q_1, q_2, \cdots, q_m\}$ is the set of queries in training data. $D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$ is the set of documents associated with query q_i, $\Sigma_i = \{\sigma_{i,1}, \sigma_{i,2}, \cdots, \sigma_{i,k}\}$ is the set of basic rankings on the documents in D_i with respect to query q_i, and π_i is the final ranking on the documents in D_i with respect to query q_i. Here, $d_{i,j}$ denotes the j^{th} document in D_i, $\sigma_{i,j}$ denotes the j^{th} basic ranking in Σ_i, and k denotes the number of basic rankings. The training set is represented as $S = \{(q_i, \Sigma_i), \pi_i\}_{i=1}^{m}$.

In learning, a model for ranking aggregation is constructed, which takes the form of $F(q, \Sigma) : \mathcal{Q} \times \Pi^k \rightarrow \Re^n$, where q is a query, D is a set of associated documents, Σ is a set

Table 3.1: Summary of Notations

Notation	Explanation
$\mathcal{Q}$	query set
$\mathcal{D}$	document set
$\mathcal{S} = \{(q_i, \Sigma_i), \pi_i\}_{i=1}^m$	training data set
$q_i \in \mathcal{Q}$	i-th query in training data
$D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$	set of documents associated with q_i
$d_{i,j} \in \mathcal{D}$	j-th document in D_i
$\Sigma_i = \{\sigma_{i,1}, \sigma_{i,2}, \cdots, \sigma_{i,k}\}$	set of basic rankings on D_i with respect to q_i
$\sigma_{i,j} \in \Pi_i, (j = 1, \cdots, k)$	j-th basic ranking on D_i with respect to q_i
$\pi_i \in \Pi_i$	final ranking on D_i for q_i
Π_i	set of possible rankings on D_i with respect to q_i
$F(q, \Sigma)$	model for ranking aggregation
$\mathcal{T} = \{(q_{m+1}, \Sigma_{m+1})\}$	test data set
$q_{m+1} \in \mathcal{Q}$	query in test data
$D_{m+1} = \{d_{m+1,1}, d_{m+1,2}, \cdots, d_{m+1,n_{m+1}}\}$	set of documents associated with q_{m+1}
$\Sigma_{m+1} = \{\sigma_{m+1,1}, \sigma_{m+1,2}, \cdots, \sigma_{m+1,k}\}$	set of rankings on D_{m+1} with respect to q_{m+1}

of basic rankings on the documents in D with respect to q, n denotes the number of documents, and k denotes the number of basic rankings. $F(q, \Sigma)$ can assign scores to the documents in D, sort the documents according to the scores, and generate a final ranking.

$$S_D = F(q, \Sigma)$$

$$\pi = \text{sort}_{S_D} D.$$

Note that F is a global ranking function in the sense that it is defined on a set of documents.

The test data consists of query q_{m+1}, associated documents D_{m+1}, and basic rankings on the documents Σ_{m+1}. We use the trained ranking model $F(q, \Sigma)$ to assign scores to the documents in D_{m+1}, sort them based on the scores, and give the final ranking list. The test data set is represented as $T = \{(q_{m+1}, \Sigma_{m+1})\}$.

Table 3.1 gives a summary of notations.

Ranking aggregation is generally defined as a query dependent task so far. In practice, it is usually specified as a query independent task. That is,

$$F(q, \Sigma) = F(\Sigma).$$

Note that we use integers to represent documents. Let the documents in D_i be identified by the integers $\{1, 2, \cdots, n_i\}$. We define permutation π_i on D_i as a bijection from $\{1, 2, \cdots, n_i\}$

to itself. We use Π_i to denote the set of all possible ranking lists (permutations) on D_i, and use $\pi_i(j)$ and $\pi_i^{-1}(j)$ to respectively denote the rank of the j-th document and the document at the j-th rank.

Note that for simplicity, it is assumed here that all the basic rankings are on the same set of documents D_i. In practice, different basic rankings can be provided on different subsets of documents. In such case, one can just take the union of the subsets and define the union as D_i.

Evaluation measures in ranking aggregation can be any measure in learning to rank, depending on how the ground truth is given. For example, if the final ranking in ground truth is given as a ranking list, then Kendal's Tau can be used. If the final ranking is based on grades, then MAP or NDCG can be employed.

3.2 LEARNING METHODS

Existing methods for ranking aggregation include unsupervised learning methods such as Borda Count and Markov Chain, and supervised learning methods such as Cranking and Multinomial Preference Model. We give detailed explanations on Borda Count, Markov Chain, and Cranking in Chapter 4. See also [27, 61, 75].

Borda Count is an election method in which the best candidate is selected based on voters' rankings of candidates. Aslam & Montague [8] have applied it to meta-search and have verified the effectiveness of the method in meta-search. Borda Count is a simple method as follows. First, each voter gives a ranking of the n candidates (objects). For each ranking, the top ranked candidate receives n points, the second ranked candidate receives n-1 points, and so on. Then the candidates are ranked in descending order of their total points, generating the final ranking list, and the candidate with the most points wins.

Markov Chain based ranking aggregation assumes that there exists a Markov Chain on the objects. The basic rankings of objects are utilized to construct the Markov Chain, in the way that the transition probabilities are estimated based on the order relations in the rankings. The stationary distribution of the Markov Chain is then utilized to rank the objects in the final ranking. Dwork et al. [36] propose four methods for constructing the transition probability matrix of the Markov Chain, which leads to four different Markov Chain methods for ranking aggregation.

Cranking proposed by Lebanon & Lafferty [66] employs a generalization of the Mallows model for ranking aggregation. The generalized Mallows model, which is in the form of exponential function, defines a conditional probability distribution of final ranking of objects given multiple basic rankings of objects. The model can be viewed as a general probabilistic model for ranking aggregation. Lebanon & Lafferty propose several learning methods for estimating the parameters of the generalized Mallows model under different settings, particularly when the rankings in the training data are only partially available.

Multinomial Preference Model (MPM) is a model for ranking aggregation developed by [108, 109], which can be used in both unsupervised learning and supervised learning settings. The probability of a basic ranking list is calculated based on a multinomial distribution of preference

relations over all object pairs, where the distribution is parameterized by the scores of objects. Furthermore, the probability of the final ranking list is calculated as product of the probabilities of basic ranking lists. The scores of objects are shared by all the basic ranking lists. Parameters representing uncertainties of scores and parameters representing weights of basic rankings are also incorporated into the model. The learning of the model is formalized as Maximum Likelihood Estimation and an optimal solution can be found by Gradient Decent.

CHAPTER 4

Methods of Learning to Rank

This chapter describes in details about thirteen methods for ranking creation, including PRank [31], OC SVM [98], McRank [71], Ranking SVM [49, 50], IR SVM [14], GBRank [125, 126], RankNet [12], ListNet & ListMLE [15, 115], AdaRank [119], SVM MAP [122], SoftRank [45, 102], LambdaRank [13, 34], and LambdaMART [11, 113], and three methods for ranking aggregation, including Borda Count [36], Markov Chain [36], and CRanking [66].

4.1 PRANK

PRank (Perceptron Ranking) is an online algorithm for ordinal classification proposed by Crammer & Singer [31]. One can employ it in ranking (ranking creation) as a pointwise method. The basic idea of PRank is to utilize and learn a number of parallel Perceptron models while each model makes classification between the neighboring grades.

4.1.1 MODEL

Suppose that $\mathcal{X} \subseteq \Re^d$ and $\mathcal{Y} = \{1, 2, \cdots, l\}$ where there is a total order on $\mathcal{Y}$. $x \in \mathcal{X}$ is an object (feature vector) and $y \in \mathcal{Y}$ is a label representing grade. Given object x, we aim to predict its label (grade) y. That is to say, this is an ordinal classification problem. We employ a number of linear models (Perceptrons) $\langle w, x\rangle - b_r, (r = 1, \cdots, l-1)$ to make the prediction, where $w \in \Re^d$ is a weight vector and $b_r \in \Re, (r = 1, \cdots, l)$ are biases satisfying $b_1 \leq \cdots \leq b_{l-1} \leq b_l = +\infty$. The models correspond to parallel hyperplanes $\langle w, x\rangle - b_r = 0$ separating grades r and $r+1$, $(r = 1, \cdots, l-1)$. Here $\langle\rangle$ denotes dot product. Figure 4.1 gives an example of PRank models. If x satisfies $\langle w, x\rangle - b_{r-1} \geq 0$ and $\langle w, x\rangle - b_r < 0$, then $y = r$, $(r = 1, \cdots, l)$. We can write it as $\min_{r\in\{1,\cdots,l\}}\{r|\langle w, x\rangle - b_r < 0\}$.

4.1.2 LEARNING ALGORITHM

PRank employs the Perceptron learning algorithm [97] to simultaneously learn the linear models online. The Perceptron learning algorithm is based on Stochastic Gradient Descent, and so is PRank.

PRank takes one input pair at each round. Suppose that for the current round, the input pair is (x, y), and we are to update the weights w and biases $b_r, (r = 1, \cdots, l-1)$. For simplicity, we omit the superscript representing the round here. Given a feature vector x, the current models can predict a grade $\hat{y}$ for it. Specifically, if x satisfies $\langle w, x\rangle - b_{r-1} \geq 0$ and

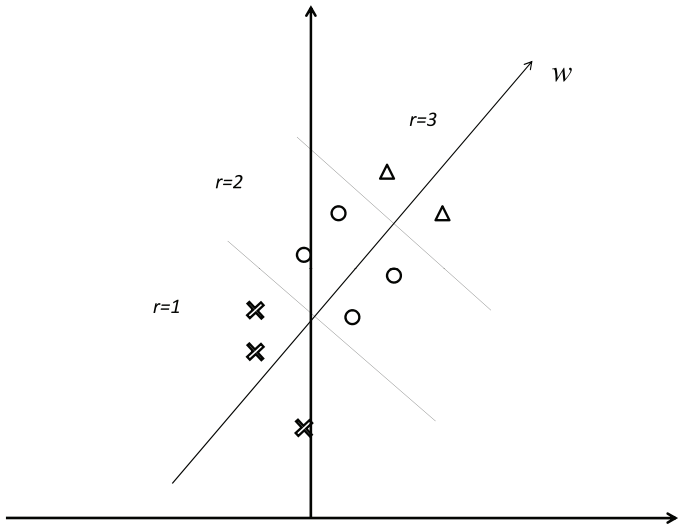

Figure 4.1: PRank Model

$\langle w, x\rangle - b_r < 0$, then the predicted grade should be $\hat{y} = r$, $(r = 1, \cdots, l)$. On the other hand, given the true grade label y, it is also possible to say which models should predict the feature vector as positive example and which models should predict it as negative example. We use variables $(z_1, \cdots, z_{l-1}) = (+1, \cdots, +1, -1, \cdots, -1)$ to represent the fact, where the first $y - 1$ variables correspond to the models which should make positive predictions $(+1)$ and the rest variables the models which should make negative predictions (-1). Thus, if the prediction of a model is correct, then $z_r(\langle w, x\rangle - b_r) > 0$ holds; if the prediction is incorrect, then $z_r(\langle w, x\rangle - b_r) \leq 0$ holds $(r = 1, \cdots, l - 1)$. When an error occurs, PRank adjusts the weights for all the models making the error. Specifically, PRank updates those models' biases b_r with $b_r - z_r$ and updates the weights w with $w + (\sum z_r)x$ where the sum is taken over the models making the error.

Figure 4.2 shows the PRank algorithm.

4.2 OC SVM

The method proposed by Shashua & Levin [98] also utilizes a number of parallel hyperplanes as ranking model. Their method learns the parallel hyperplanes by the Large Margin principle. In one implementation, the method tries to maximize a fixed margin for all the neighboring grades.

4.2.1 MODEL

Suppose that $\mathcal{X} \subseteq \Re^d$ and $\mathcal{Y} = \{1, 2, \cdots, l\}$ where there exists a total order on $\mathcal{Y}$. $x \in \mathcal{X}$ is feature vector and $y \in \mathcal{Y}$ is a label representing a grade. As in PRank, we employ a number of

Input: training data $\{(x_t, y_t)\}_{i=1}^{T}$
Initialize $w = 0, b_1, \cdots, b_{l-1} = 0, b_l = \infty$.
For $t = 1, \cdots, T$

- Get a feature vector $x_t \in \mathcal{R}^n$
- Predict $\hat{y}_t = \min_{r\in\{1,\cdots,l\}}\{r|\langle w, x_t\rangle - b_r < 0\}$
- Get a new label y_t
- If $\hat{y}_t \neq y_t$ then update w
  - For $r = 1, \cdots, l-1$: if $y_t \leq r$ then $z_r = -1$, else $z_r = +1$
  - For $r = 1, \cdots, l-1$: if $z_r(\langle w, x_t\rangle - b_r) \leq 0$ then $\zeta_r = z_r$, else $\zeta_r = 0$
  - Update $w = w + (\sum \zeta_r)x$
  - For $r = 1, \cdots, l-1$ update $b_r = b_r - \zeta_r$

End For
Output: the ranking model $\langle w, x\rangle - b_r = 0, r = 1, \cdots, l-1$

Figure 4.2: PRank Algorithm

parallel hyperplanes $\langle w, x\rangle - b_r = 0, (r = 1, \cdots, l-1)$ to predict the label y of a given feature vector x, where $w \in \Re^d$ is a weight vector and $b_r \in \Re, (r = 1, \cdots, l)$ are biases satisfying $b_1 \leq \cdots \leq b_{l-1} \leq b_l = +\infty$. If x satisfies $\langle w, x\rangle - b_{r-1} \geq 0$ and $\langle w, x\rangle - b_r < 0$, then $y = r$, $(r = 1, \cdots, l)$. That is to say, the prediction is based on $\min_{r\in\{1,\cdots,l\}}\{r|\langle w, x\rangle - b_r < 0\}$.

4.2.2 LEARNING ALGORITHM

OC SVM assumes that the parallel hyperplanes separate the instances in any two adjacent grades with the same large margin (Figure 4.3).

Suppose that the training data is given as follows. For each grade $r = 1, \cdots, l$, there are m_r instances: $x_{r,i}, i = 1, \cdots, m_r$. The learning task is formalized as the following Quadratic Programming (QP) problem.

$$\begin{gathered}\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{r=1}^{l-1}\sum_{i=1}^{m_r}(\xi_{r,i} + \xi^*_{r+1,i}) \\ \text{s. t. } \langle w, x_{r,i}\rangle - b_r \leq -1 + \xi_{r,i} \\ \langle w, x_{r+1,i}\rangle - b_r \geq 1 + \xi^*_{r+1,i} \\ \xi_{r,i} \geq 0, \quad \xi^*_{r+1,i} \geq 0 \\ i = 1, \cdots, m_r, \quad r = 1, \cdots, l-1 \\ m = m_1 + \cdots + m_l\end{gathered} \tag{4.1}$$

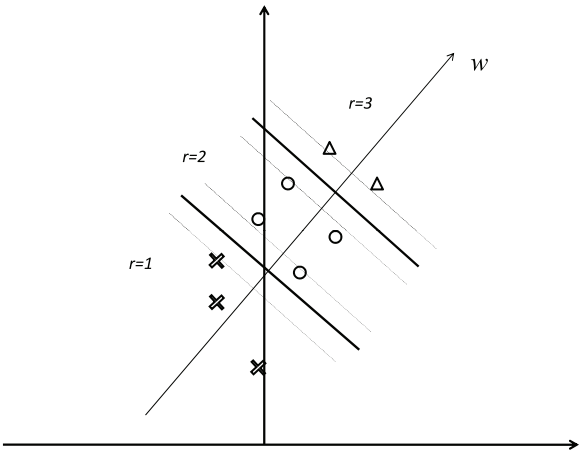

Figure 4.3: OC SVM Model

where $x_{r,i}$ denotes the i-th instance in the r-th grade, $\xi_{r+1,i}$ and $\xi^*_{r+1,i}$ denote the corresponding slack variables, $||\cdot||$ denotes L_2 norm, m is the number of training instances, and $C > 0$ is coefficient. The detailed way of solving the problem can be found in [98].

Figure 4.4 shows the learning algorithm of OC SVM.

Input: training data $\{(x_i, y_i)\}_{i=1}^m$
Solve the QP problem in (4.1)
Output: the ranking model $\langle w, x\rangle - b_r = 0, r = 1, \cdots, l-1$

Figure 4.4: Learning Algorithm of OC SVM

4.3 MCRANK

Li et al.[71] propose taking ranking as an ordinal classification (multi-class classification) problem and utilizing Gradient Boosting[1] to learn the ranking model. Their approach is motivated by the theoretical result that ranking errors in terms of DCG are bounded by classification errors. The algorithm McRank (Mc stands for multi-class) thus is a pointwise method.[2]

[1]Gradient Boosting has several names. Here we adopt the name used in Wikipedia. MART (Multiple Additive Regression Tree) is the name of one of its implementations.
[2]There are two variants of McRank, which work almost equally well. Here, we take the simpler one.

4.3.1 MODEL

Suppose that $\mathcal{X} \subseteq \Re^d$ and $\mathcal{Y} = \{1, 2, \cdots, K\}$ where there exists a total order on $\mathcal{Y}$. $x \in \mathcal{X}$ is a feature vector representing an object and $y \in \mathcal{Y}$ is a label representing a grade. Given feature vector x, we first calculate the probability of its being in grade $y = k$ using a probabilistic multi-class classifier, denoted as $P(y = k|x)$. We then calculate the expected grade of x as

$$f(x) = \sum_{k=1}^{K} P(y = k|x) \cdot k$$

Finally, we rank all the given objects using their expected grades. We can in general calculate the expected score of x and rank objects with their expected scores. The expected score of x is defined as

$$f(x) = \sum_{k=1}^{K} P(y = k|x) \cdot s(k)$$

where $s(k)$ is a monotonically increasing function, for example, $s(k) = k$ or $s(k) = 2^k$.

Note that McRank has K models for ranking, one for each class. This is in contrast to GBRank and LambdaMART, which have only one model for ranking.

4.3.2 LEARNING ALGORITHM

Suppose that training data is given as $\{(x_i, y_i)\}_{i=1}^{N}$. For the multi-class classification learning problem, we conduct the following optimization.

$$\arg\min \sum_{i=1}^{N} \sum_{k=1}^{K} -\log p(y = k|x_i)\mathbf{1}[y_i = k]$$

We specifically employ the Gradient Tree Boosting algorithm (Algorithm 6 in [40]) to learn the model, as shown Figure 4.5.

4.4 RANKING SVM

Ranking SVM is one of the first learning to rank methods, proposed by Herbrich et al. [49, 50]. The basic idea of Ranking SVM is to transform ranking into pairwise classification and employ the SVM technique [28] to perform the learning task.

4.4.1 LINEAR MODEL AS RANKING FUNCTION

Assume that $X \subseteq \Re^d$ is the feature space and $x \in X$ is an element in the space (feature vector). Further suppose that f is a scoring function $f : X \to \Re$. Then one can rank feature vectors (objects) in X with $f(x)$. That is to say, given any two feature vectors $x_i \in X$ and $x_j \in X$, if

Parameter: M (number of iterations), L (tree size, i.e., number of leaf nodes), and η (learning rate)
Input: $S = \{(x_i, y_i)\}_{i=1}^{N}$
$\tilde{y}_{i,k} = 1$, if $y_i = k$, and $\tilde{y}_{i,k} = 0$, otherwise
$F_{k,1}(x) = 0, k = 1, \cdots, K$
For $m = 1, \cdots, M$

- **For** $k = 1, \cdots, K$
  - $p_{i,k} = \frac{\exp(F_{k,m}(x_i))}{\sum_{l=1}^{K} \exp(F_{l,m}(x_i))}$, for $i = 1, \cdots, N$
  - $\{R_{k,l,m}\}_{l=1}^{L}$ = L-leaf node regression tree for $\{\tilde{y}_{i,k} - p_{i,k}, x_i\}_{i=1}^{N}$
  - $\beta_{k,l,m} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{k,l,m}} \tilde{y}_{i,k} - p_{i,k}}{\sum_{x_i \in R_{k,l,m}} (1-p_{i,k}) p_{i,k}}$
  - $F_{k,m}(x) = F_{k,m-1}(x) + \eta \cdot \sum_{l=1}^{L} \beta_{k,l,m} \mathbf{1}[x \in R_{k,l,m}]$

Output: classifier $F_{k,M}(x), k = 1, \cdots, K$

Figure 4.5: Learning Algorithm of McRank

$f(x_i) > f(y_j)$, then x_i should be ranked ahead of x_j, and vice versa.

$$x_i \succ x_j \Leftrightarrow f(x_i) > f(x_j).$$

In principle, function $f(x)$ can be any function. For simplicity, let us suppose that $f(x)$ is a linear function for the time being.

$$f(x) = \langle w, x \rangle,$$

where w denotes a weight vector and $\langle \cdot \rangle$ denotes inner product.

We can transform the ranking problem into a binary classification problem if the scoring function is a linear function. The reason is as follows.

First, the following relation holds for any two feature vectors x_i and x_j, when $f(x)$ is a linear function.

$$f(x_i) > f(x_j) \Leftrightarrow \langle w, x_i - x_j \rangle > 0.$$

Next, for any two feature vectors x_i and x_j, we can consider a binary classification problem on the difference of the feature vectors $x_i - x_j$. Specifically, we assign a label y to it.

$$y = \begin{cases} +1, & \text{if } x_i - x_j > 0 \\ -1, & \text{if } x_i - x_j < 0 \end{cases}$$

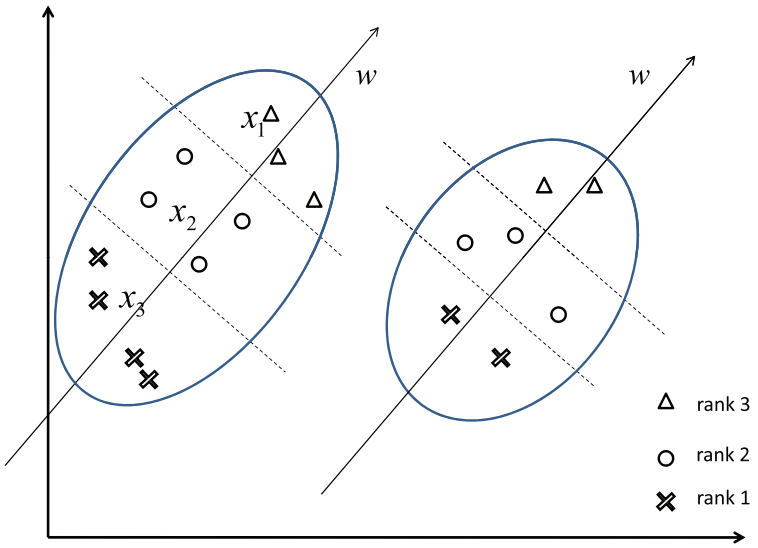

Figure 4.6: Ranking Problem

Hence,

$$\langle w, x_i - x_j \rangle > 0 \Leftrightarrow y = +1.$$

Therefore, the following relation holds. That is to say, if x_i is ranked ahead of x_j, then y is $+1$, otherwise, y is -1[3].

$$x_i \succ x_j \Leftrightarrow y = +1.$$

4.4.2 RANKING SVM MODEL

We can learn and utilize a linear classifier, such as Linear SVM for the ranking task. The classifier can be directly used as ranking model. One can also extend the linear model to a non-linear model by using the kernel trick. We call the above method Ranking SVM.

Figure 4.6 shows an example ranking problem. Suppose that there are two groups of objects (documents associated with two queries) in the feature space. Further suppose that there are three grades (levels). For example, objects x_1, x_2, and x_3 in the first group are at three different grades. The weight vector w corresponds to the linear function $f(x) = \langle w, x \rangle$, which can score and rank the objects. Ranking objects with the function is equivalent to projecting the objects into the vector and sorting the objects according to the projections on the vector. If the ranking function is 'good', then there should be an effect that objects at grade 3 are ranked ahead of objects at grade 2, etc. Note that objects belonging to different groups are incomparable.

[3]For ease of explanation, we do not consider the case in which there is a tie. One can make an extension to handle it.

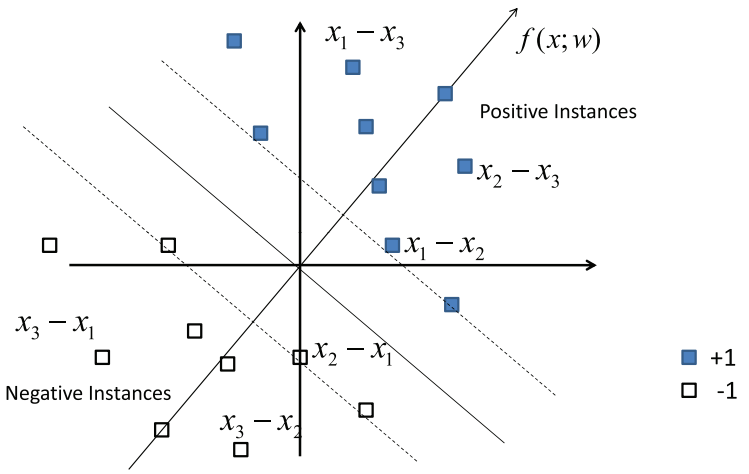

Figure 4.7: Transformation to Pairwise Classification

Figure 4.7 shows how the ranking problem in Figure 4.6 can be transformed to Linear SVM classification. The differences between two feature vectors at different grades in the same group are treated as new feature vectors, e.g., $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$. Furthermore, labels are also assigned to the new feature vectors. For example, $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are positive. Note that feature vectors at the same grade or feature vectors from different groups are not utilized to create new feature vectors. One can train a Linear SVM classifier, which separates the new feature vectors as shown in Figure 4.7. Note that the hyperplane of the SVM classifier passes the original, and the positive and negative instances are anti-symmetric. For example, $x_1 - x_2$ and $x_2 - x_1$ are positive and negative instances, respectively. In fact, we can discard the negative instances in learning because they are redundant.

4.4.3 LEARNING ALGORITHM

More formally, Ranking SVM is formalized as the following constrained optimization problem (Quadratic Programming). We first consider the linear case,

$$\begin{gathered}\min_{w,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s. t. } y_i\langle w, x_i^{(1)} - x_i^{(2)}\rangle \geq 1 - \xi_i \\ \xi_i \geq 0 \\ i = 1, \ldots, N\end{gathered}$$

where $x_i^{(1)}$ and $x_i^{(2)}$ denote the first and second feature vectors in a pair of feature vectors, $||\cdot||$ denotes L_2 norm, N is the number of training instances, and $C > 0$ is coefficient.

It is equivalent to the following non-constrained optimization problem, i.e., minimization of regularized hinge loss function.

$$\min_{w} \sum_{i=1}^{N} [1 - y_i \langle w, x_i^{(1)} - x_i^{(2)} \rangle]_+ + \lambda ||w||^2, \tag{4.2}$$

where $[x]_+$ denotes function $\max(x, 0)$ and $\lambda = \frac{1}{2C}$. The primal QP problem can be solved by solving the dual problem.

$$\begin{array}{c} \max_{\alpha} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i^{(1)} - x_i^{(2)}, x_j^{(1)} - x_j^{(2)} \rangle + \sum_{i=1}^{N} \alpha_i \\ \text{s. t. } \sum_{i=1}^{N} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C, \\ i = 1, \cdots, N. \end{array} \tag{4.3}$$

The optimal solution, used as ranking function, is given as

$$f(x) = \sum_{i=1}^{N} \alpha_i^* y_i \langle x, x_i^{(1)} - x_i^{(2)} \rangle. \tag{4.4}$$

The learning algorithm of Ranking SVM is summarized in Figure 4.8.

We can also generalize the above problem to the non-linear case by using the kernel trick.

$$\begin{array}{c} \max_{\alpha} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i^{(1)} - x_i^{(2)}, x_j^{(1)} - x_j^{(2)}) + \sum_{i=1}^{N} \alpha_i \\ \text{s. t. } \sum_{i=1}^{N} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C, \\ i = 1, \cdots, N. \end{array}$$

The optimal ranking function is in the form

$$f(x) = \sum_{i=1}^{N} \alpha_i^* y_i K(x, x_i^{(1)} - x_i^{(2)}).$$

Parameter: C
Input: training data $\{((x_i^{(1)}, x_i^{(2)}), y_i)\}, i = 1, \cdots, N$
Solve the dual problem in (4.3) to obtain the optimal parameters $\alpha_i^*, (i = 1, \cdots, N)$
Output: the ranking model in (4.4)

Figure 4.8: Learning Algorithm of Ranking SVM

4.5 IR SVM

IR SVM proposed by Cao et al. [14] is an extension of Ranking SVM for information retrieval, whose idea can be applied to other applications as well.

4.5.1 MODIFIED LOSS FUNCTION

Ranking SVM transforms ranking into pairwise classification, and thus it actually makes use of 0-1 loss in the learning process. There exists a gap between the loss function and the IR evaluation measures. IR SVM attempts to bridge the gap by modifying the 0-1 loss, that is, conducting cost sensitive learning of Ranking SVM.

Grade: 3, 2, 1
Documents are represented by their grades
Example 1:
    ranking for query-1: 3 2 1
Example 2:
    ranking-1 for query-2: 2 3 2 1 1 1 1
    ranking-2 for query-2: 3 2 1 2 1 1 1
Example 3:
    ranking for query-3: 3 2 2 1 1 1 1
    ranking for query-4: 3 3 2 2 2 1 1 1 1 1

Figure 4.9: Example Ranking Lists

We first look at the problems caused by straightforward application of Ranking SVM to document retrieval, using examples in Figure 4.9.

One problem with direct application of Ranking SVM is that it equally treats document pairs across different grades. In example 1, there are three pairs of documents. They are document pairs with label pairs (grade pairs) 3-2, 3-1, and 2-1, respectively. Ranking SVM uses the same 0-1 loss for the document pairs. This is in contrast to the fact that different document pairs should have different importance in ranking. Actually, making correct ordering on the pair of 3-1 (ranking 3 ahead of 1) is more critical than the other pairs. Example 2 indicates the problem from another perspective. There are two rankings for the same query. In ranking-1 the documents at positions 1 and 2 are swapped from the perfect ranking, while in ranking-2 the documents at positions 3 and 4 are swapped from the perfect ranking. There is only one error for each ranking in terms of 0-1 loss. Therefore, they have the same effect on training of Ranking SVM, which is not desirable. Actually, ranking-2 should be better than ranking-1, from the viewpoint of IR, because the result of its top is better. Again, to have a high accuracy on top-ranked documents is crucial for an IR system, which is reflected in the IR evaluation measures.

Another issue with Ranking SVM is that it equally treats document pairs from different queries. The numbers of documents usually vary from query to query. In example 3, there are two queries, and the numbers of documents associated with them are different. For query3 there are 2 document pairs between grades 3-2, 4 document pairs between grades 3-1, 8 document pairs between grades 2-1, and in total 14 document pairs. For query4, there are 31 document pairs. Ranking SVM takes 14 instances (document pairs) from query3 and 31 instances (document pairs) from query4 for training. Thus, the impact on the training process from query4 will be larger than the impact from query3. In other words, the model learned will be biased toward query4. This is in contrast to the fact that in IR evaluation queries are evenly important.

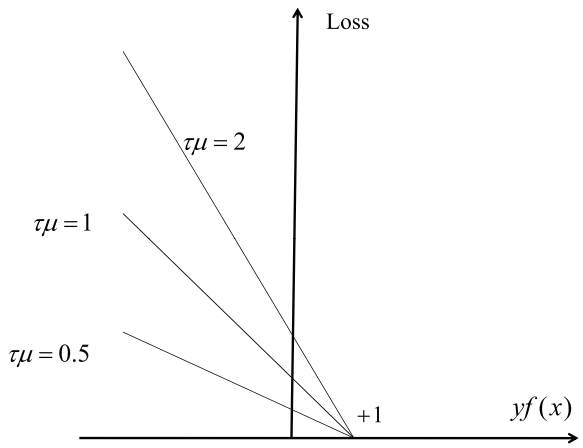

Figure 4.10: Modified Hinge Loss Functions

IR SVM addresses the above two problems by changing the 0-1 classification into a cost sensitive classification. It does so by modifying the hinge loss function of Ranking SVM. Specifically, it sets different losses for document pairs across different grades and from different queries. To emphasize the importance of correct ranking on the top, the loss function heavily penalizes errors on the top. To increase the influences of queries with less documents, the loss function heavily penalizes errors from such queries.

Figure 4.10 plots the shapes of different hinge loss functions with different penalty parameters. The x-axis represents $yf(x_i^{(1)} - x_i^{(2)})$ and the y-axis represents loss. When $yf(x_i^{(1)} - x_i^{(2)}) \geq 1$, the losses are zero. When $yf(x_i^{(1)} - x_i^{(2)}) < 1$, the losses are linearly decreasing functions with different slopes. If the slope equals -1, then the function is the normal hinge loss function. IR SVM modifies the hinge loss function, specifically modifies the slopes for different grade pairs and different queries. It assigns higher weights to document pairs belonging to important grade pairs and normalizes weights of document pairs according to queries.

4.5.2 LEARNING ALGORITHM

The learning of IR SVM is equivalent to the following optimization problem. Specifically, minimization of the modified regularized hinge loss function,

$$\min_{w} \sum_{i=1}^{N} \tau_{k(i)} \mu_{q(i)} [1 - y_i \langle w, x_i^{(1)} - x_i^{(2)} \rangle]_+ + \lambda ||w||^2,$$

where $[x]_+$ denotes function $\max(x, 0)$, $\lambda = \frac{1}{2C}$, and $\tau_{k(i)}$ and $\mu_{q(i)}$ are weights. See the loss function of Ranking SVM (4.2).

Here $\tau_{k(i)}$ represents the weight of instance i whose label pair belongs to the k-th grade pair. Xu et al. propose a heuristic method to determine the value of τ_k. The method takes, average decrease in NDCG@1 when randomly changing the positions of documents belonging to the grade pair, as the value of τ_k. Moreover, $\mu_{q(i)}$ represents the normalization weight of instance i from query q. The value of $\mu_{q(i)}$ is simply calculated as $\frac{1}{n_q}$, where n_q is the number of document pairs for query q.

The equivalent constrained optimization (Quadratic Programming) problem is as below.

$$\begin{gathered} \min_{w,\xi} \tfrac{1}{2}||w||^2 + C_i \textstyle\sum_{i=1}^{N} \xi_i \\ \text{s. t. } y_i \langle w, x_i^{(1)} - x_i^{(2)} \rangle \geq 1 - \xi_i, \\ C_i = \tfrac{\tau_{k(i)} \mu_{q(i)}}{2\lambda} \\ \xi_i \geq 0, \\ i = 1, \ldots, N. \end{gathered}$$

The primal problem can be solved by solving the dual problem.

$$\begin{gathered} \max_{\alpha} -\tfrac{1}{2} \textstyle\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i^{(1)} - x_i^{(2)}, x_j^{(1)} - x_j^{(2)} \rangle + \sum_{i=1}^{N} \alpha_i \\ \text{s. t. } \textstyle\sum_{i=1}^{N} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C_i, \\ C_i = \tfrac{\tau_{k(i)} \mu_{q(i)}}{2\lambda} \\ i = 1, \cdots, N. \end{gathered} \tag{4.5}$$

The optimal solution is in the following form.

$$f(x) = \sum_{i=1}^{N} \alpha_i^* y_i \langle (x_i^{(1)} - x_i^{(2)}), x \rangle. \tag{4.6}$$

The learning algorithm of IR SVM is summarized in Figure 4.11.

4.6 GBRANK

GBRank proposed by Zheng et al. [125, 126] is also a pairwise method, which is based on Gradient Tree Boosting.

Parameter: C_i estimated based on τ and μ
Input: training data $\{((x_i^{(1)}, x_i^{(2)}), y_i)\}_{i=1}^N$
Solve the dual problem in (4.5) to obtain $\alpha_i^*, (i = 1, \cdots, N)$
Output: the ranking model in (4.6)

Figure 4.11: Learning Algorithm of IR SVM

4.6.1 LOSS FUNCTION

GBRank takes preference pairs as training data

$$\{(x_i^{(1)}, x_i^{(2)}), x_i^{(1)} \succ x_i^{(2)}\}_{i=1}^N.$$

In GBRank, the following pairwise loss function is utilized.

$$L(f) = \frac{1}{2}\sum_{i=1}^{N}(\max\{0, \tau - (f(x_i^{(1)}) - f(x_i^{(2)})\})^2,$$

where $f(x)$ is the ranking function and $\tau (0 < \tau \leq 1)$ is parameter. Note that it is assumed that $x_i^{(1)} \succ x_i^{(2)}$ holds. The intuitive explanation to the loss function is that if $f(x_i^{(1)})$ is larger than $f(x_i^{(2)})$ with τ, then the loss is zero; otherwise, the loss is $\frac{1}{2}(f(x_i^{(2)}) - f(x_i^{(1)}) + \tau)^2$ (cf., the loss function in Ranking SVM (4.2)).

We can employ Functional Gradient Decent to optimize the loss function with respect to the training instances. First, we view

$$f(x_i^{(1)}), \quad f(x_i^{(2)}), \quad i = 1, \cdots, N.$$

as variables and compute the gradient of $L(h)$ with respect to the training instances

$$-\max\{0, f(x_i^{(2)}) - f(x_i^{(1)}) + \tau\}, \quad \max\{0, f(x_i^{(2)}) - f(x_i^{(1)}) + \tau\}, \quad i = 1, \cdots, N$$

If $f(x_i^{(1)}) - f(x_i^{(2)}) \geq \tau$, then the corresponding loss is zero, and there is no need to change the ranking function. If $f(x_i^{(1)}) - f(x_i^{(2)}) < \tau$, the corresponding loss is non-zero, and we change the ranking function using Gradient Descent

$$f_k(x) = f_{k-1}(x) - \eta \nabla L(f_k(x)),$$

where ∇ stands for gradient and η is learning rate. More specifically,

$$f_k(x_i^{(1)}) = f_{k-1}(x_i^{(1)}) + \eta(f_{k-1}(x_i^{(2)}) - f_{k-1}(x_i^{(1)}) + \tau)$$

$$f_k(x_i^{(2)}) = f_{k-1}(x_i^{(2)}) - \eta(f_{k-1}(x_i^{(2)}) - f_{k-1}(x_i^{(1)}) + \tau),$$

where $f_k(x)$ and $f_{k-1}(x)$, respectively, denote the values of $f(x)$ in the k-th and $(k-1)$-th iterations, and η is learning rate. If η equals one, then we only need to update the function in the following way (in the k-th iteration).

$$f_k(x_i^{(1)}) = f_{k-1}(x_i^{(2)}) + \tau$$

$$f_k(x_i^{(2)}) = f_{k-1}(x_i^{(1)}) - \tau.$$

4.6.2 LEARNING ALGORITHM

GBRank collects all the pairs with non-zero losses (in the k-th iteration)

$$\{(x_i^{(1)}, f_{k-1}(x_i^{(2)}) + \tau), (x_i^{(2)}, f_{k-1}(x_i^{(1)}) - \tau)\}$$

views it as regression data and employs Gradient Tree Boosting [40] to learn a model $g_k(x)$ that can make prediction on the regression data. The learned model $g_k(x)$ is then linearly combined with the existing model $f_{(k-1)}(x)$ to create a new model $f_k(x)$ (in the k-th iteration)

$$f_k(x) = \frac{k f_{k-1}(x) + \beta g_k(x)}{k+1},$$

where β is shrinkage factor.

Figure 4.12 shows the GBRank algorithm.

4.7 RANKNET

RankNet developed by Burges et al. [12] is also a pairwise method.

4.7.1 LOSS FUNCTION

RankNet adopts Cross Entropy as loss function in learning.

First, it is assumed that in the training data a probability is associated with each pair of objects. For object pair (document pair) $x_i^{(1)}$ and $x_i^{(2)}$, probability $\bar{P}_i$ is given, which represents the probability that $x_i^{(1)}$ is ahead of $x_i^{(2)}$ (e.g., $x_i^{(1)}$ has a higher grade than $x_i^{(2)}$). For example, $\bar{P}_i = 1$ means that $x_i^{(1)}$ should definitely be ahead of $x_i^{(2)}$. $\bar{P}_i = 0.5$ means that it is not certain which is ahead of which (e.g., they belong to the same grade).

Second, it is assumed that a probability is calculated for each pair of objects using the ranking function. For object pair (document pair) $x_i^{(1)}$ and $x_i^{(2)}$, probability P_i is calculated. Suppose that ranking function $f : \Re^d \to \Re$ assigns scores to objects. Let $s_i^{(1)} = f(x_i^{(1)})$, $s_i^{(2)} = f(x_i^{(2)})$, $s_i = s_i^{(1)} - s_i^{(2)}$. Then we define

$$P_i \equiv \frac{\exp(s_i)}{1 + \exp(s_i)}. \tag{4.7}$$

Parameter: β (shrinkage factor), τ (threshold), and K (number of iterations)
Input: $S = \{((x_i^{(1)}, x_i^{(2)}), x_i^{(1)} \succ x_i^{(2)})\}_{i=1}^{N}$
Initialize ranking function $f_0(x)$
For $k = 1, \cdots, K$

- Use the previous function $f_{k-1}(x)$
- Separate S into two subsets

$$S^+ = \{(x_i^{(1)}, x_i^{(2)}) | f_{k-1}(x_i^{(1)}) - f_{k-1}(x_i^{(2)}) \geq \tau\}$$

$$S^- = \{(x_i^{(1)}, x_i^{(2)}) | f_{k-1}(x_i^{(1)}) - f_{k-1}(x_i^{(2)}) < \tau\}$$

- Create regression data

$$\{(x_i^{(1)}, f_{k-1}(x_i^{(2)}) + \tau), (x_i^{(2)}, f_{k-1}(x_i^{(1)}) - \tau) | (x_i^{(1)}, x_i^{(2)}) \in S^-\}$$

- Employing Gradient Tree Boosting to learn regression model $g_k(x)$ using the regression data
- Construct the current function

$$f_k(x) = \frac{k f_{k-1}(x) + \beta g_k(x)}{k+1}$$

Output: the ranking function $f_K(x)$

Figure 4.12: Learning Algorithm of GB Rank

If $f(x_i^{(1)}) > f(x_i^{(2)})$, then $x_i^{(1)}$ is ranked ahead of $x_i^{(2)}$ with probability P_i.

Cross Entropy, which measures 'distance' between two probability distributions is defined as

$$L_i = -\bar{P}_i \log P_i - (1 - \bar{P}_i) \log(1 - P_i). \tag{4.8}$$

We make use of Cross Entropy as loss function for prediction on the order of a pair of objects. Plugging (4.7) into (4.8) yields

$$L_i = -\bar{P}_i s_i + \log(1 + \exp(s_i)).$$

When $\bar{P}_i = 1$, Cross Entropy loss becomes logistic loss

$$L_i = \log(1 + \exp(-s_i)).$$

4.7.2 MODEL

RankNet employs Neural Network as model (Figure 4.13). That is why the method is called RankNet. The Neural Network is supposed to be a three layer network with a single output node, represented as

$$s = f(x;\theta) = f\left(\sum_j w_j \cdot f_j\left(\sum_k w_{jk}x_{(k)} + b_j\right) + b\right), \tag{4.9}$$

where $x_{(k)}$ denotes the k-th element of input x, w_{jk}, and b_{jk}, and f_j denote the weight, offset, and activation function of the first layer, respectively, w_j, b, and f denote the weight, offset, and activation function of the second layer, respectively, and s denotes the final output. θ denotes the parameter vector. The activation functions are sigmoid functions (non-linear functions).

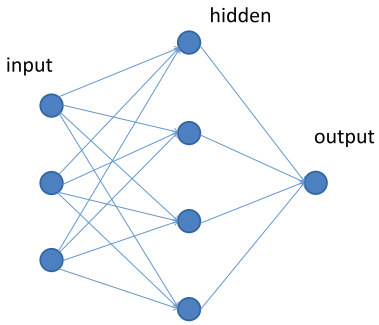

Figure 4.13: RankNet Model

4.7.3 LEARNING ALGORITHM

RankNet employs the Back Propagation algorithm (equivalently Stochastic Gradient Descent) to learn the parameters of network. Given training data $\{(x_i^{(1)}, x_i^{(2)}), P_i\}, i = 1, \cdots, N$. The algorithm iteratively updates parameter θ with each training instance (preference pair) using

$$\theta - \eta\frac{\partial L}{\partial \theta}, \tag{4.10}$$

where η is learning rate. For simplicity, we omit the superscript representing the index of iteration.

The loss on a preference pair is calculated as $L = L(s^{(1)} - s^{(2)})$. The gradient $\frac{\partial L}{\partial \theta}$ of loss L with respect to parameter θ is calculated as

$$\frac{\partial L}{\partial \theta} = L'\left(\frac{\partial s^{(1)}}{\partial \theta} - \frac{\partial s^{(2)}}{\partial \theta}\right),$$

where $L' = \frac{dL(s)}{ds}$

More precisely, the gradients with respect to specific parameters are calculated as follows.

$$\frac{\partial L}{\partial b} = L' f'^{(1)} - L' f'^{(2)} \equiv \Delta^{(1)} - \Delta^{(2)} \tag{4.11}$$

$$\frac{\partial L}{\partial w_j} = \Delta^{(1)} f_j^{(1)} - \Delta^{(2)} f_j^{(2)} \tag{4.12}$$

$$\frac{\partial L}{\partial b_j} = \Delta^{(1)} w_j f'^{(1)}_j - \Delta^{(2)} w_j f'^{(2)}_j \equiv \Delta_j^{(1)} - \Delta_j^{(2)} \tag{4.13}$$

$$\frac{\partial L}{\partial w_{jk}} = \Delta_j^{(1)} x_{(k)}^{(1)} - \Delta_j^{(2)} x_{(k)}^{(2)}. \tag{4.14}$$

Learning is actually performed by forward and backward propagation on the Neural Network. Foreword propagation (fprop) corresponds to re-calculation of the final score based on the new parameters, and backward propagation (backprop) corresponds to re-calculation of the parameters.

RankNet uses a validation data set to make selection of parameters, i.e., it employs Cross Validation in parameter selection. In this way, any IR measure can be utilized as evaluation measure. Burges et al. take the uses of Neural Network, Cross Entropy loss, back-propagation, and Cross Validation as the major characteristics of the RankNet method.

Parameter: η (learning rate) and T (number of iterations)
Input: training data $\{((x_i^{(1)}, x_i^{(2)}), P_i)\}_{i=1}^N$
For $t = 1, \cdots, T$

- **For** $i = 1, \cdots, N$
  - Take the instance and preference probability in training data $(x_i^{(1)}, x_i^{(2)}), P_i$
  - Fprop: use the current network to calculate scores $s_i^{(1)} = f(x_i^{(1)})$ and $s_i^{(2)} = f(x_i^{(2)})$
  - Calculate loss with P_i, $s_i^{(1)}$ and $s_i^{(2)}$ using (4.8)
  - Bprop: update the parameters of network using (4.10)-(4.14)
- **End For**

End For
Output: the ranking model in (4.9)

Figure 4.14: Learning Algorithm of RankNet

4.7.4 SPEED UP OF TRAINING

Burges et al. have also proposed an efficient algorithm for speeding up the training process of RankNet [13].

Two ingredients are used to make the speed up. First, instead of conducting Stochastic Gradient Descent, the algorithm performs Batch Gradient Descent. Second, it stores and reuses some of the intermediate results, assuming that the algorithm is applied to search in which the query-document structure can be leveraged.

Suppose that P is the set of document pairs (i, j) appearing in the training data. Further suppose that D is the whole set of documents, $P_{i\_}$ is the set of documents j for which $\{i, j\}$ is a pair in P, and $P_{\_j}$ is the set of documents i for which $\{i, j\}$ is a pair in P. Let m denote the number of queries and n denote the number of documents per query.

The algorithm calculates the gradient of loss function with respect to the parameter *over all the training data*.[4]

$$\frac{\partial L}{\partial \theta} = \sum_{(i,j)\in P} \frac{\partial L(s_i, s_j)}{\partial s_i}\frac{\partial s_i}{\partial \theta} + \frac{\partial L(s_i, s_j)}{\partial s_j}\frac{\partial s_j}{\partial \theta},$$

where $L(\cdot)$ denotes the loss function and θ denotes the parameters of model, and s_i and s_j are the scores of i and j.

The algorithm further rewrites the gradient as

$$\frac{\partial L}{\partial \theta} = \sum_{i\in D} \frac{\partial s_i}{\partial \theta} \sum_{j\in P_{i\_}} \frac{\partial L(s_i, s_j)}{\partial s_i} + \sum_{j\in D} \frac{\partial s_j}{\partial \theta} \sum_{i\in P_{\_j}} \frac{\partial L(s_i, s_j)}{\partial s_j}. \tag{4.15}$$

In each iteration of a query, n fprops are performed to compute the final score s_i. Next, for each document, $\sum_{j\in P_{i\_}} \frac{\partial L(s_i,s_j)}{\partial s_i}$ and $\sum_{i\in P_{\_j}} \frac{\partial L(s_i,s_j)}{\partial s_j}$ are computed and stored. After that, n fprops and n backprops are conducted to compute the gradients $\frac{\partial s_i}{\partial \theta}$ and $\frac{\partial s_j}{\partial \theta}$.

In this way, the efficiency of calculation with regard to $\frac{\partial L(s_i,s_j)}{\partial s_i}$ and $\frac{\partial L(s_i,s_j)}{\partial s_j}$ is enhanced from the order of $O(n^2)$ to the order of $O(n)$, where n is the number of documents per query.

4.8 LISTNET AND LISTMLE

ListNet and ListMLE are probabilistic and listwise methods for learning to rank, proposed by Cao et al. [15] and Xia et al. [115]. The methods exploit the Plackett-Luce model studied in statistics. See also [46].

4.8.1 PLACKETT-LUCE MODEL

Let us first look at the Plackett-Luce model (the PL model for short). PL model defines a probability distribution on permutations of objects, referred to as permutation probability. Suppose

[4]For ease of explanation, we change the notation $L(s_1, s_2) = L(s_1 - s_2)$.

that there is a set of objects $o = \{o_1, o_2, \cdots, o_n\}$. Let π denote a permutation (ranking list) of the objects and $\pi^{-1}(i)$ denote the object in the i-th rank (position) in π. Further suppose that there are non-negative scores assigned to the objects. Let $s = \{s_1, s_2, \cdots, s_n\}$ denote the scores of the objects.

The PL model defines the probability of permutation π based on scores s as follows.

$$P_s(\pi) = \prod_{i=1}^{n} \frac{s_{\pi^{-1}(i)}}{\sum_{j=i}^{n} s_{\pi^{-1}(j)}}.$$

The probabilities of permutations naturally form a probability distribution.

For example, suppose that there are three objects A, B, C and s_A, s_B, s_C are scores of the objects ($s_A > s_B > s_C$). The probability of permutation ABC is

$$P_s(ABC) = \frac{s_A}{s_A + s_B + s_C} \frac{s_B}{s_B + s_C} \frac{s_C}{s_C}.$$

The probability of permutation BCA is

$$P_s(BCA) = \frac{s_B}{s_A + s_B + s_C} \frac{s_C}{s_A + s_C} \frac{s_A}{s_A}.$$

It is easy to verify that

$$P_s(ABC) + P_s(ACB) + P_s(BAC) + P_s(BCA) + P_s(CAB) + P_s(CBA) = 1.$$

The permutation probability has the following interpretation, as explained below with the above example. Given three objects A, B, and C and their scores, we randomly generate a permutation on them. If we first select A from A, B, C based on A's relative score, then select B from B and C based on B's relative score, and finally select C, then we generate the permutation ABC with probability $P_s(ABC)$. Permutation probability $P_s(ABC)$ represents the likelihood of permutation ABC being generated in the process.

The PL model has some nice properties. First, the permutation in descending order of scores has the largest probability and the permutation in ascending order of scores has the smallest probability. In the above example, $P_s(ABC)$ is the largest and $P_s(CBA)$ is the smallest among the permutation probabilities. Furthermore, given the permutation in descending order of scores, swapping any two objects in the permutation will decrease the probability. In the above example, swapping B and C in ABC yields ACB and we have $P_s(ABC) > P_s(ACB)$.

The PL model also defines a probability distribution on top k subgroups, referred to as top k probability. Given objects and permutations of objects, we can define top k subgroups on the objects. Top k subgroup $g[o_1 \cdots o_k]$ represents all permutations whose top k objects are $o_1 \cdots o_k$. The top k probability of subgroup $g[o_1 \cdots o_k]$ is defined as

$$P_s(g[o_1 \cdots o_k]) = \prod_{i=1}^{k} \frac{s_{o_i}}{\sum_{j=i}^{n} s_{o_j}}.$$

In the above example, we have

$$P_s(g[A]) = \frac{s_A}{s_A + s_B + s_C}.$$

$$P_s(g[AB]) = \frac{s_A}{s_A + s_B + s_C} \frac{s_B}{s_B + s_C}.$$

It is easy to verify the following relation between top k probability and permutation probability holds, which is another property of the PL model.

$$P_s(g[o_1 \cdots o_k]) = \sum_{\pi \in g[o_1 \cdots o_k]} P_s(\pi)$$

For example,

$$P_s(g[A]) = P_s(ABC) + P_s(ACB).$$

4.8.2 LISTNET

ListNet makes use of a parameterized Plackett-Luce model. The model can be based on either permutation probability or top k probability, but due to efficiency consideration, it is usually based on top k probability. Time complexity of computation of permutation probabilities is of order $O(n!)$ while that of top k probabilities is of order $O(n!/(n-k)!)$.

In document retrieval, suppose that for query q and its associated documents $d_1, d_2, \cdots, d_n$, the corresponding relevance labels $y_1, y_2, \cdots, y_n$ are given. From query q and documents $d_1, d_2, \cdots, d_n$, feature vectors $x_1, x_2, \cdots, x_n$ are created.

Given feature vectors $x_1, x_2, \cdots, x_n$, the top k probability of subgroup $g[x_1 \cdots x_k]$ may be calculated as

$$P_{F(\mathbf{x};\theta)}(g[x_1 \cdots x_k]) = \prod_{i=1}^{k} \frac{\exp(f(x_i;\theta))}{\sum_{j=i}^{n_i} \exp(f(x_j;\theta))}, \tag{4.16}$$

where $f(x;\theta)$ is a Neural Network model with parameter θ and $F(\mathbf{x};\theta)$ is a list of scores given by the Neural Network. That is to say, the score of x_i is determined by an exponential function of the Neural Network, which works as ranking model.

The *corresponding* labels $y_1, y_2, \cdots, y_n$ can be transformed to scores as well. Specifically, the score of x_i by y_i can be determined by an exponential function of y_i. Then the top k probability by the labels may be calculated similarly.

$$P_{\mathbf{y}}(g[x_1 \cdots x_k]) = \prod_{i=1}^{k} \frac{\exp(y_i)}{\sum_{j=i}^{n_i} \exp(y_j)}. \tag{4.17}$$

If two scoring functions have a similar effect in ranking, then the permutation distributions (or top k probability distributions) by them should be similar in shape. Figure 4.15 gives two permutation distributions based on two scoring functions f and g. One can measure the difference between the two scoring functions by using KL Divergence. This is exactly the idea in ListNet.

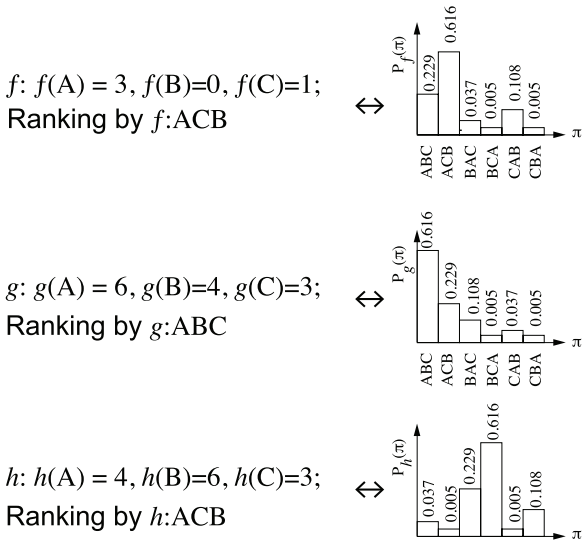

Figure 4.15: Examples of Permutation Probability Distributions by PL Model

ListNet measures the difference between the top k probability by the Neural Network model and the top k probability by the ground truth using KL Divergence. KL Divergence between two probability distributions is defined as $D(P||Q) = \sum_i p_i \log \frac{p_i}{q_i}$ where P and Q are the two probability distributions. Here, only $\sum_i -p_i \log q_i$ is used, since $\sum_i p_i \log p_i$ is constant. (Note that KL Divergence is asymmetric).

Suppose that the training data is given as $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. Each instance $(\mathbf{x}_i, \mathbf{y}_i)$ is given as $((x_{i,1}, x_{i,2}, \cdots, x_{i,n_i}), (y_{i,1}, y_{i,2}, \cdots, y_{i,n_i}))$. A global ranking function is defined based on a local ranking function: $F(\mathbf{x}_i) = (f(x_{i1}), f(x_{i2}), \cdots, f(x_{in_i}))$.

ListNet takes the KL Divergence over all the training instances as total loss and learns the ranking model by minimizing the total loss. The total loss function is defined as

$$L(\theta) = \sum_{i=1}^{m} L(\mathbf{y}_i, F(\mathbf{x}_i; \theta)).$$

Here the loss function for each instance is defined as

$$\begin{aligned} L(\mathbf{y}_i, F(\mathbf{x}_i; \theta)) &= -\sum_{g \in \mathcal{G}_i^k} P_{\mathbf{y}_i}(g) \log P_{F(\mathbf{x}_i;\theta)}(g) \\ &= -\sum_{g \in \mathcal{G}_i^k} \prod_{j=1}^{k} \frac{\exp(y_{i,j})}{\sum_{l=j}^{n_i} \exp(y_{i,l})} \log \prod_{j=1}^{k} \frac{\exp(f(x_{i,j}; \theta))}{\sum_{l=j}^{n_i} \exp(f(x_{i,l}; \theta))}, \end{aligned}$$

where $P_{\mathbf{y}_i}(g)$ denotes the top k probability of subgroup g by the ground truth $\mathbf{y}_i$, $P_{F(\mathbf{x}_i;\theta)}(g)$ denotes the top k probability of subgroup g by Neural Network $F(\mathbf{x}_i;\theta)$, and $\mathcal{G}_i^k$ denotes all top k subgroups.

ListNet employs Gradient Decent to perform the optimization. Figure 4.16 gives the learning algorithm.

$$\frac{\partial L(\theta)}{\partial \theta} = \sum_{i=1}^{m} \frac{\partial L(\mathbf{y}_i, F(\mathbf{x}_i;\theta))}{\partial \theta} \tag{4.18}$$

$$\frac{\partial L(\mathbf{y}_i, F(\mathbf{x}_i;\theta))}{\partial \theta} = -\sum_{g \in \mathcal{G}^k} \frac{P_{\mathbf{y}_i}(g)}{P_{F(\mathbf{x}_i;\theta)}(g)} \frac{\partial P_{F(\mathbf{x}_i;\theta)}(g)}{\partial \theta}. \tag{4.19}$$

When $k = 1$, we have

$$\begin{aligned} \frac{\partial L(\mathbf{y}_i, F(\mathbf{x}_i;\theta))}{\partial \theta} &= -\sum_{j=1}^{n_i} P_{y_i}(x_{i,j}) \frac{\partial f(x_{i,j};\theta)}{\partial \theta} \\ &\quad + \sum_{j=1}^{n_i} P_{y_i}(x_{i,j}) \sum_{l=1}^{n_i} P_{f(x_{i,l})}(x_{i,l}) \frac{\partial f(x_{i,l};\theta)}{\partial \theta}. \end{aligned} \tag{4.20}$$

Parameter: k (top positions), η (learning rate), and T (number of iterations)
Input: $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$
Initialize parameter θ
For $t = 1, \cdots, T$

- **For** $i = 1, \cdots, m$
  - Take input $(\mathbf{x}_i, \mathbf{y}_i)$
  - Compute gradient $\nabla L(\theta) = \partial L(\theta)/\partial\theta$ using (4.18)-(4.20)
  - Update $\theta = \theta - \eta \nabla L(\theta)$
- **End For**

End For
Output: the Neural Net model $f(x;\theta)$

Figure 4.16: Learning Algorithm of ListNet

4.8.3 LISTMLE

Another algorithm is ListMLE, which employs the parameterized Plackett-Luce model (4.16)-(4.17) and Maximum Likelihood Estimation. Specifically, it maximizes the following total loss

function based on logarithmic loss

$$L(\mathbf{y}_i, F(\mathbf{x}_i; \theta)) = -\sum_{i=1}^{m} \log \prod_{j=1}^{k} \frac{\exp(f(x_{i,\pi_i^{-1}(j)}; \theta))}{\sum_{l=j}^{n_i} \exp(f(x_{i,\pi_i^{-1}(l)}; \theta))},$$

where π_i is a perfect ranking by y_i. The learning algorithm of ListMLE is the similar to that of ListNet. Note that when $k = 1$, ListMLE degenerates to Logistic Regression.

4.9 ADARANK

Since the evaluation measures in IR are based on lists, it is more natural and effective to directly optimize listwise loss functions in learning to rank. AdaRank, proposed by Xu & Li [119], is one of the direction optimization algorithms.

4.9.1 LOSS FUNCTION

Suppose that the training data is given as lists of feature vectors and their corresponding lists of labels (grades) $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$. We are to learn a ranking model $f(x)$ defined on object (feature vector) x. Given a new list of objects (feature vectors) $\mathbf{x}$, the learned ranking model can assign a score to each of the objects $x, x \in \mathbf{x}$. We can then sort the objects based on the scores to generate a ranking list (permutation) π. The evaluation is conducted at the list level, specifically, a listwise evaluation measure $E(\pi, \mathbf{y})$ is utilized.

In learning, ideally we would create a ranking model that can maximize the accuracy in terms of a listwise evaluation measure on training data, or equivalently, minimizes the loss function defined below,

$$L(f) = \sum_{i=1}^{m} (E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i)) = \sum_{i=1}^{m} (1 - E(\pi_i, \mathbf{y}_i)), \tag{4.21}$$

where π_i is the permutation on feature vector $\mathbf{x}_i$ by ranking model f and $\mathbf{y}_i$ is the corresponding list of grades. We refer to the loss function $L(\cdot)$ as the 'true loss function' (or 'empirical risk function') and those methods that manage to minimize the true loss function as the 'direct optimization approach'.

The loss function is not smooth and differentiable, and thus straightforward optimization of the evaluation might not work. Instead, we can consider optimizing an upper bound of the loss function.

Since inequality

$$\exp(-x) \geq 1 - x$$

holds, we can consider optimizing the following upper bound

$$\sum_{i=1}^{m} \exp(-E(\pi_i, \mathbf{y}_i)).$$

Other upper bounds can also be considered, for example,

$$\sum_{i=1}^{m} \log\left(1 + \exp(-E(\pi_i, \mathbf{y}_i))\right).$$

That is to say, the exponential function and logistic function may be exploited as 'surrogate' loss functions in learning. Note that both functions are continuous, differentiable, and even convex with respect to E.

4.9.2 LEARNING ALGORITHM

AdaRank minimizes the exponential loss function, by taking the Boosting approach. Mimicking the famous AdaBoost algorithm [38], AdaRank conducts stepwise minimization of the exponential loss function. More specifically, AdaRank repeats the process of re-weighting the training instances, creating a weak ranker, and assigning a weight to the weak ranker, to minimize the loss function. Finally, AdaRank linearly combines the weak rankers as the ranking model. Figure 4.17 shows the algorithm of AdaRank.

We can prove that AdaRank can continuously reduce the empirical loss function during the training process, under certain condition, as shown in [119]. When the evaluation measure is dot product, AdaRank can reduce the loss to zero.

One advantage of AdaRank is its simplicity, and it is perhaps one of the simplest learning to rank algorithms.

4.10 SVM MAP

Another approach of direct optimization tries to use the Structural SVM techniques to learn a ranking model. The algorithm SVM MAP developed by Yue et al. [122] is such an algorithm. Xu et al. [121] further generalize it to a group of algorithms, including PermuRank. See also [16, 65].

4.10.1 LOSS FUNCTION

In ranking, for query q_i the ranking model $f(x_{ij})$ assigns a score to each feature vector x_{ij} where x_{ij} is the feature vector derived from q_i and its associated document d_{ij}. The feature vectors $\mathbf{x}_i$ (documents $\mathbf{d}_i$) are then sorted based on their scores, and a ranking denoted as $\tilde{\pi}_i$ is obtained. The labels of feature vectors $\mathbf{x}_i$ are also given as $\mathbf{y}_i$.

For simplicity, suppose that the ranking model $f(x_{ij})$ is a linear model:

$$f(x_{ij}) = \langle w, x_{ij} \rangle, \tag{4.22}$$

where w denotes the weight vector.

We consider using a scoring function $S(\mathbf{x}_i, \pi_i)$ to measure the goodness of a given permutation (ranking) π_i. $S(\mathbf{x}_i, \pi_i)$ is defined as

$$S(\mathbf{x}_i, \pi_i) = \langle w, \sigma(\mathbf{x}_i, \pi_i) \rangle,$$

Input: $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$
Parameter: T (number of iterations)
Evaluation measure: E
Initialize $P_1(i) = 1/m$
For $t = 1, \cdots, T$

- Create weak ranker h_t with weighted distribution $\boldsymbol{P}_t$ on training data S
- Choose α_t

$$\alpha_t = \frac{1}{2} \cdot \ln \frac{\sum_{i=1}^{m} P_t(i)(1 + E(\pi_i, \mathbf{y}_i))}{\sum_{i=1}^{m} P_t(i)(1 - E(\pi_i, \mathbf{y}_i))}$$

- where $\pi_i = \text{sort}_{h_t}(\mathbf{x}_i)$
- Create f_t

$$f_t(x) = \sum_{k=1}^{t} \alpha_k h_k(x)$$

- Update $\boldsymbol{P}_{t+1}$

$$P_{t+1}(i) = \frac{\exp(-E(\pi_i, \mathbf{y}_i))}{\sum_{j=1}^{m} \exp(-E(\pi_j, \mathbf{y}_j))}$$

- where $\pi_i = \text{sort}_{f_t}(\mathbf{x}_i)$

End For
Output: the ranking model $f(x) = f_T(x)$

Figure 4.17: Learning Algorithm of AdaRank

where w is still the weight vector and vector $\sigma(\mathbf{x}_i, \pi_i)$ is defined as

$$\sigma(\mathbf{x}_i, \pi_i) = \frac{2}{n_i(n_i - 1)} \sum_{k,l:k<l} z_{kl}(x_{ik} - x_{il}),$$

where $z_{kl} = +1$ if $\pi_i(k) < \pi_i(l)$ (x_{ik} is ranked ahead of x_{il} in π_i), and -1, otherwise.

We can use the scoring function in learning. For query q_i, we calculate $S(\mathbf{x}_i, \pi_i)$ for each permutation π_i and select the permutation $\tilde{\pi}_i$ with the largest score:

$$\tilde{\pi}_i = \arg \max_{\pi_i \in \Pi_i} S(\mathbf{x}_i, \pi_i), \tag{4.23}$$

where Π_i denotes the set of all the possible permutations for $\mathbf{x}_i$.

It can be easily shown that, the ranking $\tilde{\pi}_i$ selected by Eq.(4.23) is equivalent to the ranking created by the ranking model $f(x_{ij})$ (when both of them are linear functions). Figure 4.18 gives an

example. It is easy to verify that both $f(x)$ and $S(\mathbf{x}_i, \pi)$ will output ABC as the most preferable ranking (permutation).

Objects: A, B, C
$f_A = \langle w, x_A \rangle$, $f_B = \langle w, x_B \rangle$, $f_C = \langle w, x_C \rangle$
Suppose $f_A > f_B > f_C$
For example:
Permutation1: ABC
Permutation2: ACB
$S_{ABC} = \frac{1}{6}\langle w, ((x_A - x_B) + (x_B - x_C) + (x_A - x_C))\rangle$
$S_{ACB} = \frac{1}{6}\langle w, ((x_A - x_C) + (x_C - x_B) + (x_A - x_B))\rangle$
$S_{ABC} > S_{ACB}$

Figure 4.18: Example of Scoring Function

In this way, we can view the problem of learning a ranking model as the optimization problem in which the following loss function is minimized.

$$\sum_{i=1}^{m} \max_{\pi_i^* \in \Pi_i^*; \pi_i \in \Pi_i \setminus \Pi_i^*} \left(\left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) \cdot \left[\left[S(\mathbf{x}_i, \pi_i^*) \leq S(\mathbf{x}_i, \pi_i) \right] \right] \right), \tag{4.24}$$

where $[[c]]$ is one if condition c is satisfied; otherwise, it is zero. $\pi_i^* \in \Pi_i^* \subseteq \Pi_i$ denotes any of the perfect permutations for q_i.

The loss function measures the loss when the most preferred ranking by the ranking model is not the perfect ranking. One can prove that the true loss function in (4.21) is upper bounded by the new loss function in (4.24).

4.10.2 LEARNING ALGORITHMS

The loss function (4.24) is still not continuous and differentiable. We can consider using continuous, differentiable, and even convex upper bounds of the loss function (4.24).

1) The 0-1 function in (4.24) can be replaced with its upper bounds, for example, hinge, exponential, and logistic functions, yielding

$$\sum_{i=1}^{m} \max_{\pi_i^* \in \Pi_i^*, \pi_i \in \Pi_i \setminus \Pi_i^*} \left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) \cdot \exp(-\left(S(\mathbf{x}_i, \pi_i^*) - S(\mathbf{x}_i, \pi_i) \right)),$$

$$\sum_{i=1}^{m} \max_{\pi_i^* \in \Pi_i^*, \pi_i \in \Pi_i \setminus \Pi_i^*} \left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) \cdot \log\left(1 + \exp(-\left(S(\mathbf{x}_i, \pi_i^*) - S(\mathbf{x}_i, \pi_i) \right))\right),$$

$$\sum_{i=1}^{m} \max_{\pi_i^* \in \Pi_i^*, \pi_i \in \Pi_i \setminus \Pi_i^*} \left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) \cdot [1 - \left(S(\mathbf{x}_i, \pi_i^*) - S(\mathbf{x}_i, \pi_i) \right)]_+,$$

$$\sum_{i=1}^{m}\left[\max_{\pi_i^*\in\Pi_i^*,\pi_i\in\Pi_i\setminus\Pi_i^*}\left(\left(E(\pi_i^*,\mathbf{y}_i)-E(\pi_i,\mathbf{y}_i)\right)-\left(S(\mathbf{x}_i,\pi_i^*)-S(\mathbf{x}_i,\pi_i)\right)\right)\right]_+,$$

where $[x]_+$ denotes function $\max(0,x)$.

2) The max function can also be replaced with its upper bound, the sum function. This is because $\sum_i x_i \geq \max_i x_i$ if $x_i \geq 0$ holds for all i.

3) Relaxations 1 and 2 can be applied simultaneously.

For example, utilizing hinge function and taking the true loss as MAP, we obtain SVM MAP. More precisely, SVM MAP solves the following optimization problem:

$$\begin{aligned}&\min_{w;\xi\geq 0}\ \tfrac{1}{2}||w||^2+\tfrac{C}{m}\textstyle\sum_{i=1}^{m}\xi_i\\ &s.t.\quad \forall i,\forall\pi_i^*\in\Pi_i^*,\forall\pi_i\in\Pi_i\setminus\Pi_i^*:\\ &S(\mathbf{x}_i,\pi_i^*)-S(\mathbf{x}_i,\pi_i)\geq E(\pi_i^*,\mathbf{y}_i)-E(\pi_i,\mathbf{y}_i)-\xi_i,\end{aligned}\tag{4.25}$$

where C is coefficient and ξ_i is the maximum loss among all the losses for permutations of query q_i.

Equivalently, SVM MAP minimizes the following regularized hinge loss function

$$\sum_{i=1}^{m}\left[\max_{\pi_i^*\in\Pi_i^*;\pi_i\in\Pi_i\setminus\Pi_i^*}(E(\pi_i^*,\mathbf{y}_i)-E(\pi_i,\mathbf{y}_i))-(S(\mathbf{x}_i,\pi_i^*)-S(\mathbf{x}_i,\pi_i))\right]_+ +\lambda||w||^2.\tag{4.26}$$

Intuitively, the first term calculates the total maximum loss when selecting the best permutation for each of the queries. Specifically, if the difference between the scores $S(\mathbf{x}_i,\pi_i^*)-S(\mathbf{x}_i,\pi_i)$ is less than the difference between the corresponding evaluation measures $E(\pi_i^*,\mathbf{y}_i)-E(\pi_i,\mathbf{y}_i)$, then there will be a loss, otherwise not. Next, the maximum loss is selected for each query, and they are summed up over all the queries. One can also consider an NDCG version of the method, with a similar formulation.

Since $c\cdot[[x\leq 0]]<[c-x]_+$ holds for all $c\in\Re^+$ and $x\in\Re$, it is easy to see that the upper bound in (4.26) also bounds the true loss function in (4.21).

Actually, it is possible to derive a number of algorithms for optimizing the upper bounds (surrogate loss functions). Xu et al. gives an example, which they call PermuRank, and they show that PermuRank can perform equally well as SVM MAP.

PermuRank minimizes the following regularized hinge loss function.

$$\sum_{i=1}^{m}\sum_{\pi_i^*\in\Pi_i^*;\pi_i\in\Pi_i\setminus\Pi_i^*}\left(E(\pi_i^*,\mathbf{y}_i)-E(\pi_i,\mathbf{y}_i)\right)\cdot\left[1-\left(S(\mathbf{x}_i,\pi_i^*)-S(\mathbf{x}_i,\pi_i)\right)\right]_+.$$

The number of possible permutations is of exponential order, and thus it is not feasible to directly implement SVM MAP and PermuRank. Both SVM MAP and PermuRank utilize a working set to cope with the difficulty. The set contains arbitrary perfect and imperfect rankings at the beginning, and the most violated perfect and imperfect rankings are added to the set at each round of learning.

Figure 4.19 summarizes the learning algorithm of SVM MAP.

Parameter: C
Input: training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$
Solve the optimization problem in (4.25) to obtain the optimal ranking model
Output: the ranking model in (4.22)

Figure 4.19: Learning Algorithm of SVM MAP

4.11 SOFTRANK

SoftRank is a direct optimization method of learning to rank, proposed by Taylor et al. [45, 102]. Because the ranking evaluation results in IR are usually not smooth and not differentiable, SoftRank tries to optimize a probabilistic approximation of ranking evaluation result. Specifically, it introduces an approximation of NDCG called Soft NDCG, optimizes the ranking evaluation result in terms of Soft NDCG, and employs a Neural Network model and Gradient Descent to perform the learning task.

4.11.1 SOFT NDCG

Let us first look at the definition of Soft NDCG. For ease of explanation, suppose that the number of documents to rank for each query is the same and equals n.

We re-write the definition of NDCG and consider NDCG at position n for permutation (ranking) π as

$$G = G_{\max}^{-1} \sum_{j=1}^{n} G(j) D(r_j),$$

where $G(j)$ denotes the gain of document j, r_j denotes the rank (position) of document j, and $D(r_j)$ denotes the position discount of document j.

Suppose that each document j has a score s_j $(j = 1, \cdots, n)$. Then, sorting the documents according to their scores will yield a ranking of documents. An NDCG value can be calculated for the ranking using the definition above. The ranking evaluation result in terms of NDCG is determined by the scores as well as the sorting, which makes it non-smooth and non-differentiable.

In calculation of Soft NDCG, we assume that the ranking of documents is decided based on the scores of documents probabilistically rather than deterministically. We can calculate the probability of each document's being ranked at a position and the *expectation* of position discount of each document. In this way, the evaluation result based on NDCG can be approximated by that based on Soft NDCG and the use of sorting can be avoided.

Specifically, Soft NDCG is defined as

$$\mathcal{G} = G_{\max}^{-1} \sum_{j=1}^{n} G(j) E(D(r_j)) \tag{4.27}$$

$$= G_{\max}^{-1} \sum_{j=1}^{n} G(j) \sum_{r=1}^{n} D(r) P_j(r), \tag{4.28}$$

where $E(D(r_j))$ denotes the expectation of position discount of document j, and $P_j(r)$ denotes the probability of document j's being ranked at rank r. The question then is how to calculate the probability distribution $P_j(r)$ and then the expectation of position discount $E(D(r_j))$.

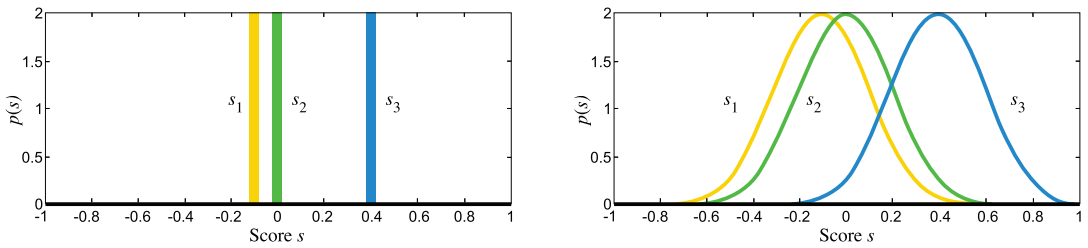

Figure 4.20: Deterministic Score v.s. Probabilistic Score

4.11.2 APPROXIMATION OF RANK DISTRIBUTION

There might be different ways to estimate the probability of a document's being ranked at a position.

Given a document, SoftRank calculates the probability of its being ranked at each position by recursively calculating the probabilities that the document is ranked ahead of or behind of the other $n-1$ documents in $n-1$ Bernoulli trials (with different success probabilities).

SoftRank assumes that the score of document x_i given by model $f(x_i;\theta)$ follows the Gaussian distribution $N((f(x_i;\theta),\sigma^2)$ with a known variance σ^2 (cf., Figure 4.20).

$$P(s \le s_i) = \int_{-\infty}^{s_i} N(s|(f(x_i;\theta),\sigma^2) ds.$$

Therefore, for any two documents x_i and x_j, the difference between their scores follows the Gaussian distribution $N((f(x_i;\theta) - f(x_j;\theta), 2\sigma^2)$. The probability that x_i is ranked ahead of x_j is thus

$$\pi_{ij} = P(s_i - s_j \ge 0) = \int_{0}^{\infty} N(s|(f(x_i;\theta) - f(x_j;\theta)), 2\sigma^2) ds.$$

SoftRank calculates the probability distribution $P_j(r)$ of document j over ranks r in a recursive manner. Suppose that it is to position document j among the n documents. First, there is only one rank, namely 1, available and document j is ranked at rank 1. The initial rank distribution $P_j^{(1)}(1)$ for document j is defined as

$$P_j^{(1)}(1) = 1.$$

Next, the remaining $n-1$ documents are assumed to be added one by one into the rank distribution. When there are $i-1$ documents in the rank distribution and document i is added, there are two possible results. The score of document i is larger than the score of document j, and thus document i is ranked ahead of document j. Or, the score of document i is smaller than the score of document j, and thus document i is ranked behind of document j. In the former case, the probability of document j being at rank r equals the probability of document j at rank $r-1$ in the previous iteration. In the latter case, the probability of document j being at rank r is the same as in the previous iteration. The two cases can be linearly combined and the rank distribution $P_j^{(i)}(r)$ for document j in the i-th iteration can be defined as

$$P_j^{(i)}(r) = \pi_{ij} P_j^{(i-1)}(r-1) + (1-\pi_{ij}) P_j^{(i-1)}(r).$$

During the calculation, it is assumed

$$P_j^{(i)}(r) = 0, \text{ if } r \leq 0.$$

Finally, the probability of document j ranked at rank r is defined as

$$P_j(r) = P_j^{(n)}(r).$$

In this way, each document has a distribution of ranks, as shown in Figure 4.21. Note that the distribution is an approximation of the true rank distribution.

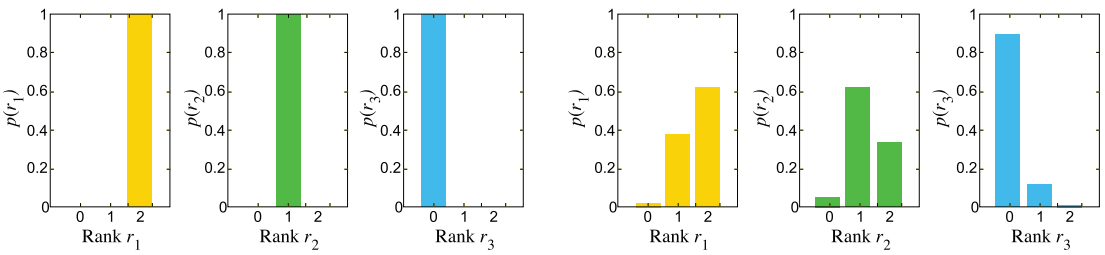

Figure 4.21: Deterministic Rank Distribution v.s. Probabilistic Rank Distribution

4.11.3 LEARNING ALGORITHM

SoftRank uses Neural Network as model and Gradient Descent as optimization technique. Suppose that there are k parameters in the model. In learning, SoftRank calculates the gradient of Soft NDCG of n documents with respect to the parameters

$$\frac{\partial \mathcal{G}}{\partial \theta} = \begin{bmatrix} \frac{\partial s_1}{\partial \theta_1} & \cdots & \frac{\partial s_n}{\partial \theta_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial s_1}{\partial \theta_k} & \cdots & \frac{\partial s_n}{\partial \theta_k} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{G}}{\partial s_1} \\ \cdots \\ \frac{\partial \mathcal{G}}{\partial s_n} \end{bmatrix}. \tag{4.29}$$

The gradient of Soft NDCG with respect to the score of document j is calculated as ($l = 1, \cdots, n$).

$$\frac{\partial \mathcal{G}}{\partial s_l} = G_{\max}^{-1} \sum_{j=1}^{n} G(j) \sum_{r=1}^{n} D(r) \frac{\partial P_j(r)}{\partial s_l}. \tag{4.30}$$

Since $P_j(r)$ is a recursively defined function, its derivative also needs to be calculated recursively. Denoting $\psi_{j,l}(r) = \frac{\partial P_j(r)}{\partial s_l}$, we recursively calculate the derivative as follows:

$$\psi_{j,l}^{(1)}(1) = 0 \tag{4.31}$$

$$\psi_{j,l}^{(i)}(r) = \psi_{j,l}^{(i-1)}(r-1)\pi_{ij} + \psi_{j,l}^{(i)}(r)(1-\pi_{ij}) + \left(P_j^{(i-1)}(r-1) - P_j^{(i-1)}(r)\right) \frac{\partial \pi_{ij}}{\partial s_l}. \tag{4.32}$$

Furthermore, $\frac{\partial \pi_{ij}}{\partial s_l}$ can be calculated in three cases (note that $i \neq j$).

$$\frac{\partial \pi_{ij}}{\partial s_l} = \begin{cases} N(0|s_l - s_j, 2\sigma^2) & l = i, l \neq j \\ -N(0|s_i - s_l, 2\sigma^2) & l \neq i, l = j \\ 0 & l \neq i, l \neq j. \end{cases} \tag{4.33}$$

4.12 LAMBDARANK

4.12.1 LOSS FUNCTION

The ranking evaluation result (the objective function in learning) is usually not continuous and differentiable, and it depends on sorting. (The sorting function itself is not continuous and differentiable as well). LambdaRank, proposed by Burges et al. [13, 34], considers employing Gradient Descent to optimize the evaluation result and tries to directly define and utilize the gradient function of the evaluation result.

Suppose that the ranking model, query, and documents are given. Then each document receives a score from the ranking model, and a ranking list can be created by sorting the documents based on the scores. Since the documents are assigned ground truth labels, a ranking evaluation result based on an IR measure can be obtained. Suppose that we use a surrogate loss function

Parameter: η (learning rate) and T (number of iterations)
Input: $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$
Initialize parameter θ
For $t = 1, \cdots, T$

- **For** $i = 1, \cdots, m$
  - Take input $(\mathbf{x}_i, \mathbf{y}_i)$
  - Compute gradient $\nabla\mathcal{G}(\theta) = \partial\mathcal{G}(\theta)/\partial\theta$ using (4.29)-(4.33)
  - Update $\theta = \theta - \eta\nabla\mathcal{G}(\theta)$
- **End For**

End For
Output Neural Net model $f(x; \theta)$

Figure 4.22: Learning Algorithm of SoftRank

L to approximate the IR evaluation measure. Then, an evaluation result based on the surrogate loss function L can also be obtained. It is this evaluation result which LambdaRank attempts to continuously optimize.

The surrogate loss function is defined on a list of documents. In that sense, LambdaRank can also be viewed as a listwise method. LambdaRank does not explicitly give the definition of the loss function. Instead it defines the *gradient* function of the surrogate loss function. More specifically, the gradient function is defined as

$$\frac{\partial L}{\partial s_i} = -\lambda_i(s_1, y_1, \cdots, s_n, y_n),$$

where $s_1, s_2, \cdots, s_n$ denote the scores of documents and $y_1, y_2, \cdots, y_n$ denote the labels of documents. Note that the index i is on a single document. That is to say, the gradient of a document depends on the scores and labels of the other documents. The sign is chosen such that a positive value for a document means that the document must reduce the loss. The gradients of documents are calculated after the current model generates a ranking list of documents for the query. The negative gradient function is called Lambda Function, and that is why the method is called LambdaRank.

The question is then how to specify the Lambda Function, so as to effectively optimize the ranking evaluation result. One idea is to increase the gradients of documents on top positions. Suppose that there are two relevant documents d_1 and d_2. One is at position 2 and the other $n-2$. The change of d_1's score to move it up to the top position should be less than the change of d_2's score to move it up to the top position. Therefore, we would prefer spending a little capacity moving d_1 up to spending a large capacity moving d_2 up. That is, the gradient (lambda score) of d_1

should be much larger than the gradient (lambda score) of d_2. In general, for any two documents ranked at two positions i_1 and i_2. Suppose that $i_1 \ll i_2$, (that is, the former document is ranked much higher than the latter document), we would have the gradients satisfy

$$|\frac{\partial L}{\partial s_{i_1}}| \gg |\frac{\partial L}{\partial s_{i_2}}|.$$

4.12.2 LEARNING ALGORITHM

When implementing LambdaRank, one only needs to consider a method for calculating the Lambda Function (the negative gradient of loss function). One common way is to calculate the lambda score of document d_i based on NDCG as shown below

$$\lambda_i = \sum_j \left(\frac{1 - S_{ij}}{2} - \frac{1}{1 + \exp(s_i - s_j)} \right) \left| G_{max}^{-1} (G_i - G_j)(D_i - D_j) \right| \tag{4.34}$$

where G_i, D_i, and s_i respectively denote the gain, discount, and score of document d_i, and G_j, D_j, and s_j respectively denote the gain, discount, and score of document d_j. G_{max}^{-1} denotes the normalizing factor of NDCG. S_{ij} is a sign function; it equals $+1$ if document d_i is more relevant than document d_j, and it equals -1 otherwise. The summation is taken over all the document pairs with d_i in the list. The Lambda Function is in fact a listwise loss function, because it is calculated on the basis of NDCG.

LambdaRank employs Neural Network as ranking model. In fact, it can be viewed as an extension of RankNet. The learning algorithm is similar to that of RankNet (the fast version), except that a different loss function is employed. Figure 4.23 summarizes the algorithm.

Parameter: η (learning rate) and T (number of iterations)
Input: $S = \{((x_i^{(1)}, y_i^{(2)}), P_i)\}_{i=1}^N$
Initialize parameter θ
For $t = 1, \cdots, T$

- Compute gradient $\nabla L(\theta) = \partial L(\theta)/\partial\theta$ using (4.15)(4.34)
- Update $\theta = \theta - \eta \nabla L(\theta)$

End For
Output: the ranking model $f(x; \theta)$

Figure 4.23: Learning Algorithm of LambdaRank

4.13 LAMBDAMART

LambdaMART is one of the most widely used algorithms in learning to rank [11, 113]. It should be viewed as a listwise algorithm, because its loss function is defined based on an IR evaluation measure, usually NDCG, on the whole ranking list of objects.

4.13.1 MODEL AND LOSS FUNCTION

LambdaMART takes lists of objects (feature vectors) and the corresponding lists of labels (grades) $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$ as training data. It learns a ranking model $f(x)$ defined on a single object (feature vector) x. Given a new list of objects (feature vectors) $\mathbf{x}$, the learned ranking model assigns a score to each of the objects $x, x \in \mathbf{x}$ and then ranks the objects based on their scores.

In learning, all the objects (feature vectors) in all the lists are combined together $\{x_i\}_{i=1}^{N}$, where N is the number of objects. LambdaMART employs Gradient Tree Boosting to construct the model, which is in the form of boosted regression trees (i.e., linear combination of regression trees).

LambdaMART follows the idea of LambdaRank [11]. It considers the use of NDCG as the true loss function, which is defined on a list of objects (a list of documents in search). Instead of directly defining the loss function on the list, it defines the negative gradient function of the (surrogate) loss function, referred to as Lambda Function. The lambda score of an object is calculated using all pairs of the other objects. It depends on the NDCG scores of the current object and the other objects in the list, as well as the entropy loss of all pairs between the current object and the other objects.

Specifically, the lambda score of object x_i is calculated as

$$\lambda_i = \sum_{j \in P} \lambda_{ij}$$

where λ_{ij} denotes the lambda score of pair (x_i, x_j) in the list and the summation is taken over all the pairs with object x_i in the list. Furthermore, the lambda score λ_{ij} is calculated as

$$\lambda_{ij} = \left(\frac{1 - S_{ij}}{2} - \frac{1}{1 + \exp(s_i - s_j)} \right) \left| G_{max}^{-1} (G_i - G_j)(D_i - D_j) \right|$$

where G_i, D_i, and s_i respectively denote the gain, discount, and score of object (document) x_i, and G_j, D_j, and s_j respectively denote the gain, discount, and score of object (document) x_j. G_{max}^{-1} denotes the normalizing factor of NDCG. S_{ij} is a sign function; it equals $+1$ if object x_i should be ranked ahead of object x_j (more relevant), and it equals -1 otherwise. There are two factors; the first factor is the derivative of the entropy loss between the two objects and the second fact is the difference of NDCG scores between the two objects (i.e., difference in NDCG when swapping the two objects).

4.13.2 LEARNING ALGORITHM

We employ a Gradient Tree Boosting algorithm. It directly uses the Lambda Function, the negative gradient of loss function, for Gradient Boosting (also known as MART). The specific algorithm is given in Figure 4.24.

Parameter: M (number of iterations), L (tree size, i.e., number of leaf nodes), and η (learning rate)
Input: $S = \{(x_i\}_{i=1}^{N}\}$ #corresponding to all documents for all queries
$F_0(x) = 0$,
For $m = 1, \cdots, M$

- **For** $i = 1, \cdots, N$
  - $y_i = \lambda_i$ #calculate Lambda Function
  - $w_i = \frac{\delta y_i}{\delta F_{m-1}(x_i)}$ #calculate derivative of Lambda Function
- $\{R_{l,m}\}_{l=1}^{L}$ = L-leaf node regression tree for $\{y_i, x_i\}_{i=1}^{N}$
- $\beta_{l,m} = \frac{\sum_{x_i \in R_{l,m}} y_i}{\sum_{x_i \in R_{l,m}} w_i}$
- $F_m(x) = F_{m-1}(x) + \eta \cdot \sum_{l=1}^{L} \beta_{l,m} \mathbf{1}[x \in R_{l,m}]$

Output: ranking model $F_M(x)$

Figure 4.24: Learning Algorithm of LambdaMART

4.14 BORDA COUNT

Borda Count is an unsupervised method for ranking aggregation. Aslam & Montague [8] propose employing Borda Count in meta search. In such case, Borda Count ranks documents in the final ranking based on their positions in the basic rankings. More specifically, in the final ranking, documents are sorted according to the numbers of documents that are ranked below them in the basic rankings. If a document is ranked high in many basic rankings, then it will be ranked high in the final ranking list.

The ranking scores of documents in the final ranking S_D are calculated as

$$S_D = F(\Sigma) = \sum_{i=1}^{k} S_i$$

$$S_i \equiv \begin{pmatrix} s_{i,1} \\ \vdots \\ s_{i,j} \\ \vdots \\ s_{i,n} \end{pmatrix}$$

$$s_{i,j} = n - \sigma_i(j),$$

where $s_{i,j}$ denotes the number of documents ranked behind document j in basic ranking σ_i, $\sigma_i(j)$ denotes the rank of document j in basic ranking σ_i, and n denotes the number of documents.

For example, documents A, B, C are ranked in three basic rankings: σ_1, σ_2, and σ_3.

$$\begin{matrix} \sigma_1 \\ \begin{pmatrix} A \\ B \\ C \end{pmatrix} \end{matrix} \quad \begin{matrix} \sigma_2 \\ \begin{pmatrix} A \\ C \\ B \end{pmatrix} \end{matrix} \quad \begin{matrix} \sigma_3 \\ \begin{pmatrix} B \\ A \\ C \end{pmatrix} \end{matrix}$$

The ranking scores of documents S_D are as follows.

$$S_D = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix}$$

The final ranking list π is created by Borda Count based on the scores S_D.

$$\begin{matrix} \pi \\ \begin{pmatrix} A \\ B \\ C \end{pmatrix} \end{matrix}$$

Borda Count can be viewed as a method of assigning a vector of k values (i.e., $n - \sigma_i(j)$) to each document and sorting the documents by L_1 norms of the vectors. One can easily come up with other alternatives, for example, sorting by medians of the vectors or L_p norms of the vectors. This leads to several different methods.

4.15 MARKOV CHAIN

The Markov Chain method for ranking aggregation, referred to as Markov Chain, assumes that there exists a Markov Chain on the documents to be ranked, and the preference relations between documents in the basic rankings represent the transitions between the documents in the Markov Chain. The stationary distribution of the Markov Chain is then utilized to rank the documents. Dwork et al. [36] have proposed four methods (denoted as MC1, MC2, MC3, and MC4) to construct the transition probability matrix of the Markov Chain.

MC1 is defined as follows. If the current state is document i, then the next state is chosen uniformly from the set of documents that are ranked higher than or equal to i in the basic rankings, that is, from the multiset $\cup_k \{j | j \succeq_k i\}$, where $j \succeq_k i$ means that j is ranked higher than or equal to i in ranking k. The transition probability matrix is defined as follows.

$$P \equiv (p(i,j))_{n\times n} = diag\left(\frac{1}{\sum_{j=1}^{n} q(1,j)}, \cdots, \frac{1}{\sum_{j=1}^{n} q(n,j)}\right) Q$$

$$Q \equiv (q(i,j))_{n\times n} = \sum_k Q_k$$

$$Q_k = (q_k(i,j))_{n\times n}$$

$$q_k(i,j) = \begin{cases} 1 & j \succeq_k i \\ 0 & \text{otherwise.} \end{cases}$$

MC2 is defined as follows. If the current state is document i, then the next state is determined by first selecting a basic ranking σ_k uniformly from all rankings and then selecting document j uniformly from the set of documents that are ranked higher than or equal to i: $\{j | j \succeq_k i\}$.

$$P \equiv (p(i,j))_{n\times n} = \frac{1}{k} \sum_k P_k$$

$$P_k \equiv (p_k(i,j))_{n\times n}$$

$$p_k(i,j) = \begin{cases} \frac{1}{m} & j \succeq_k i \\ 0 & \text{otherwise,} \end{cases}$$

where $m = |\{j | j \succeq_k i\}|$.

In MC3, if the current state is document i, then the next state is determined as follows. First, we choose ranking σ_k uniformly from the basic rankings, next for document j, if $j \succ_k i$, then we go to j; otherwise, we stay at i.

$$P \equiv (p(i,j))_{n\times n} = \frac{1}{k}\sum_k P_k$$

$$P_k \equiv (p_k(i,j))_{n\times n}$$

$$p_k(i,j) = \begin{cases} \frac{1}{n} & j \succ_k i \\ \frac{n-m}{n} & j =_k i \\ 0 & \text{otherwise,} \end{cases}$$

where $m = |\{j \,|\, j \succ_k i\}|$.

In MC4, if the current state is document i, then the next state is decided as follows. Document j is selected uniformly from the union of all documents. If $j \succ_k i$ holds for the majority of the basic rankings, then we go to j; otherwise, we stay at i.

$$P \equiv (p(i,j))_{n\times n}$$

$$p(i,j) = \begin{cases} \frac{1}{n} & q(i,j) > q(j,i) \\ \frac{n-m}{n} & j = i \\ 0 & \text{otherwise,} \end{cases}$$

where $m = |\{j \,|\, q(i,j) > q(j,i)\}|$

$$Q = (q(i,j))_{n\times n} = \sum_k Q_k$$

$$Q_k \equiv (q_k(i,j))_{n\times n}$$

$$q_k(i,j) = \begin{cases} 1 & j \succ_k i \\ 0 & \text{otherwise.} \end{cases}$$

4.16 CRANKING

The unsupervised methods described above conduct majority voting in their final ranking decisions. In fact, the methods treat all the basic ranking lists equally and give high scores to those documents ranked high in most of basic ranking lists. The uniform weight assumption may not hold in practice, however. For example, in meta search, ranking lists generated by different search engines may have different accuracies and reliabilities. One may want to learn the weights of basic ranking lists. Supervised learning methods like Cranking proposed by Lebanon & Lafferty [66] can address the problem.

4.16.1 MODEL

Cranking employs the following probability model

$$P(\pi|\theta, \Sigma) = \frac{1}{Z(\theta, \Sigma)} \exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi, \sigma_j)), \tag{4.35}$$

where π denotes the final ranking, $\Sigma = (\sigma_1, \cdots, \sigma_k)$ denotes the basic rankings, d denotes the distance between two rankings, and θ denotes weight parameters. Distance d can be, for example, Kendal's Tau. Furthermore, Z is the normalizing factor over all the possible rankings, as defined below.

$$Z(\theta, \Sigma) = \sum_{\pi} \exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi, \sigma_j)).$$

The model in Cranking is an extension of the Mallows model in statistics, in which there is only a single 'basic ranking'.

$$P(\pi|\theta, \sigma) = \frac{1}{Z(\theta, \Sigma)} \exp(\theta \cdot d(\pi, \sigma))$$

$$Z(\theta, \sigma) = \sum_{\pi} \exp(\theta \cdot d(\pi, \sigma)).$$

4.16.2 LEARNING ALGORITHM

In learning, the training data is given as $S = \{(\Sigma_i, \pi_i)\}_{i=1}^{m}$, and the goal is to build the model for ranking aggregation based on the data. We can consider employing Maximum Likelihood Estimation to learn the parameters of the model. If the final ranking and the basic rankings are all full ranking lists in the training data, then the log likelihood function is calculated as follows.

$$L(\theta) = \log \prod_{i=1}^{m} P(\pi_i|\theta, \Sigma_i) = \sum_{i=1}^{m} \log \frac{\exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi_i, \sigma_{i,j}))}{\sum_{\pi_i \in \Pi} \exp \sum_{j=1}^{k} \theta_j \cdot d(\pi_i, \sigma_{i,j})}.$$

We can employ Gradient Descent to estimate the optimal parameters.

In practice, sometimes only partial lists are given in the training data. If the final ranking lists are given as partial lists, then the likelihood function is calculated as

$$\begin{aligned} L(\theta) &= \log \prod_{i=1}^{m} P(\pi_i|\theta, \Sigma_i) = \log \prod_{i=1}^{m} \sum_{\pi_i' \in G(\pi_i)} P(\pi_i'|\theta, \Sigma_i) \\ &= \sum_{i=1}^{m} \log \frac{\sum_{\pi_i' \in G(\pi_i)} \exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi_i', \sigma_{i,j}))}{\sum_{\pi_i' \in \Pi} \exp \sum_{j=1}^{k} \theta_j \cdot d(\pi_i', \sigma_{i,j})}, \end{aligned} \tag{4.36}$$

where π_i is a partial list, and $G(\pi_i)$ denotes the group of full lists with π_i as the top partial list.

If both the final ranking lists and the basic ranking lists are given as partial lists, then the likelihood function is calculated as

$$L(\theta) = \log \prod_{i=1}^{m} P(\pi_i|\theta, \Sigma_i) = \log \prod_{i=1}^{m} \sum_{\pi_i' \in G(\pi_i)} \frac{1}{\prod_{j=1}^{k} |G(\sigma_{i,j})|} \sum_{\sigma_{i,j}' \in G(\sigma_{i,j})} P(\pi_i'|\theta, \Sigma'_i) \quad (4.37)$$

$$= \sum_{i=1}^{m} \log \sum_{\pi_i' \in G(\pi_i)} \frac{1}{\prod_{j=1}^{k} |G(\sigma_{i,j})|} \sum_{\sigma_{i,j}' \in G(\sigma_{i,j})} \frac{\exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi_i', \sigma_{i,j}'))}{\sum_{\pi_i' \in \Pi} \exp(\sum_{j=1}^{k} \theta_j \cdot d(\pi_i', \sigma_{i,j}'))},$$

where π_i is a partial list and $G(\pi_i)$ denotes the group of full lists with π_i as the top partial list, and similarly $\sigma_{i,j}$ is a partial list and $G(\sigma_{i,j})$ denotes the group of full lists with $\sigma_{i,j}$ as the top partial list. Furthermore, it is assumed here that the full lists in group $G(\sigma_{i,j})$ are uniformly distributed.

The above two likelihood functions (Eq. (4.36)-(4.37)) cannot be directly optimized. Cranking employs Markov Chain Monte Carlo (MCMC) to perform parameter estimation. Figure 4.25 summarizes the learning algorithm.

Input: training data $\{(\Sigma_i, \pi_i)\}_{i=1}^{m}$
Learn parameter θ using (Eq. (4.36)-(4.37)) and MCMC
Output: ranking model $P(\pi|\theta, \Sigma)$ (4.35)

Figure 4.25: Learning Algorithm of Cranking

4.16.3 PREDICTION

In prediction, given the learned model (i.e., the parameters θ) and the basic rankings Σ, Cranking first calculates the probability distribution of final ranking π: $P(\pi|\theta, \Sigma)$. It uses the probability distribution to calculate the expected rank of each document.

$$E(\pi(i)|\theta, \Sigma) = \sum_{r=1}^{n} r \cdot P(\pi(i) = r|\theta, \Sigma) = \sum_{r=1}^{n} r \cdot \sum_{\pi \in \Pi, \pi(i)=r} P(\pi|\theta, \Sigma).$$

It then sorts the documents based on their expected ranks.

CHAPTER 5

Applications of Learning to Rank

Learning to rank can be applied to a wide variety of applications in information retrieval and natural language processing. Typical applications are document retrieval, expert search, definition search, meta-search, personalized search, online advertisement, collaborative filtering, question answering, keyphrase extraction, document summarization, and machine translation.

In the applications, the objects (offerings) to be ranked can be documents, document units such as sentences and paragraphs, entities such as people and products. Ranking can be based on importance, preference, and quality, and it can be employed as an end-to-end solution or as a part of a solution.

This chapter introduces some example applications of learning to rank (ranking creation).

WEB SEARCH

Learning to rank has been successfully applied to web search. It is known that the ranking models at several web search engines are built by learning to rank technologies. Usually, a large number of signals representing relevance are used as features in the models. Training data is created by a group of professional judges. Moreover, powerful computing platforms for scalable and efficient training of ranking models are employed.

Learning to rank is also applied to different problems in web search, including context aware search [116], recency ranking [33], federated search [85], personalized search, online advertisement, etc.

COLLABORATIVE FILTERING

Collaborative filtering, also known as recommender system, is a task as follows. The users are asked to give ratings to the items. The system examines the ratings of items by the users and offers each user a ranking list of items. The ranking lists represent recommendations to the users from the system, while higher ranked items are more likely to be preferred by the users.

Collaborative filtering can be formalized as an ordinal classification or classification problem because users give ratings to items. Sometimes it is more natural to formalize it as ranking (ranking creation). This is because ratings from different users are on different scales and are not directly comparable, and thus it is better to view the ratings from each user as a ranking list.

Freund et al. [39] have applied RankBoost to collaborative filtering, specifically movie recommendation. RankBoost is a pairwise method for ranking in which AdaBoost is employed as the learning algorithm. In Freund et al.'s method, the ratings from the target user are viewed as training data, and the ratings from the other users are utilized as features. A RankBoost model is created with the training data. The trained model is then used for ranking of all the movies (including unrated movies) for the target user. See also [48].

DEFINITION SEARCH

In definition search, given a terminology query, the system returns a ranking list of definitions (definitional paragraphs) of the terminology. Xu et al. propose a method of definition search using learning to rank [118].

The method first automatically extracts all likely definitional paragraphs from the documents with several heuristic rules. For example, paragraphs with the first sentence being "X is a" are taken as candidates. Their method then applies a Ranking SVM model to assign to all the candidate paragraphs scores representing their likelihood of being good definitions, removes redundant paragraphs, and stores the paragraphs in a database with the terminologies as keys (e.g., X in 'X is a'). In definition search, given a terminology the system retrieves the related definitional paragraphs and returns the ranking list of definitional paragraphs.

The Ranking SVM model utilizes a number of features, including both positive and negative features. For example, if the term (e.g., X in 'X is a') repeatedly occurs in the paragraph, then it is likely the paragraph is a definition of the term. If words like 'she', 'he', or 'said' occur in the paragraph, it is likely the paragraph is not a definition.

KEYPHRASE EXTRACTION

Keyphrase extraction is a problem as follows. Given a document, a number of phrases (usually noun phrases) are output, which can precisely and compactly represent the content of the document. Traditionally keyphrase extraction is formalized as classification and classification methods such as decision tree and Naive Bayes are employed. Jiang et al. formalize the keyphrase extraction problem as ranking instead of classification [57]. In fact, keyphrase extraction can be viewed as the inverse problem of document retrieval.

Suppose that there are some training data in which a number of documents are assigned keyphrases and non-keyphrases. Jiang et al.'s method takes ordered phrase pairs as training instances, each of which consists of a keyphrase and a non-keyphrase, and builds a Ranking SVM model with the training data. The method then sorts the candidate phrases of a new document with the trained model, and selects the top ranked candidate phrases as keyphrases. Experimental results show that Ranking SVM statistically significantly outperforms the classification methods of SVM and Naive Bayes.

Jiang et al. give two reasons on the better performance of the ranking approach over the classification approach. First, it is more natural to consider the likelihood of a phrase's being a

keyphrase in a relative sense than in an absolute sense. Second, features for determining whether a phrase is a keyphrase are also relative.

QUERY DEPENDENT SUMMARIZATION

When the search system presents a search result to the user, it is important to show the titles and summaries of the documents because they are helpful for the user to judge whether the documents are relevant or not. This is the problem referred to as query-dependent summarization or snippet generation. Query dependent summarization usually contains two steps: relevant sentence selection and summary composition. In sentence selection, the most informative sentences are identified.

Metzler & Kanungo [81] propose using learning to rank techniques to conduct sentence selection in query dependent summarization. They apply GBRank and Support Vector Regression to the task. Given a query and a retrieved document, their method treats all the sentences in the document as candidate sentences and ranks the sentences based on their relevance to the query and appropriateness as a sentence in the summary. They train a model for the ranking with some labeled data. A number of features are defined in the model. For example, whether the query has an exact match in the sentence, the fraction of query terms in the sentence, the length of sentence (neither short nor long sentence is preferred), and the position of sentence in the document. They demonstrate that GBRank is an effective algorithm for the task.

MACHINE TRANSLATION

Re-ranking in machine translation is also a typical ranking problem. The state-of-the-art machine translation approach generates many candidate translations using a generative model, conducts re-ranking on the candidate translations using a discriminative model, and then selects the top ranked result. Features that can discriminate between good and bad translations are used in the re-ranking model.

There are several advantages by taking the re-ranking approach. First, the accuracy of translation may be enhanced because the discriminative model can further leverage global features and discriminative training in the final translation selection. Second, the efficiency of translation may be improved. The top n candidates are first selected with the generative model, and then the best translation is chosen from a small set of candidates.

For example, Shen at al. propose using learning to rank techniques in re-ranking of machine translation. They have proposed two methods [99] similar to the Prank algorithm. One of the algorithms is called Splitting, which tries to find parallel hyperplanes separating the top k good translations, the bottom l bad translations and the translations in between, for each sentence, where k and l are pre-determined. Figure 5.1 illustrates the Splitting model.

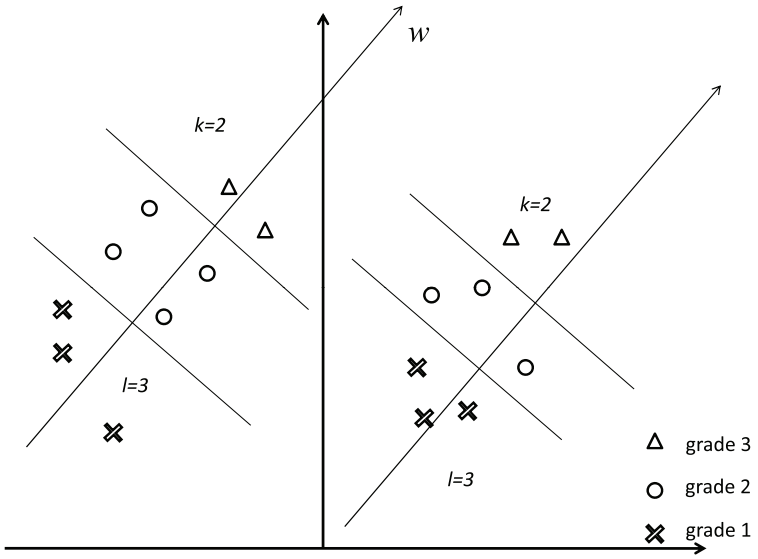

Figure 5.1: Splitting Model for Machine Translation

CHAPTER 6

Theory of Learning to Rank

This chapter gives a statistical learning formulation of learning to rank (ranking creation) and explains the issues in theoretical study of learning to rank.

6.1 STATISTICAL LEARNING FORMULATION

Learning to rank (ranking creation) is a supervised learning task. Suppose that $\mathcal{X}$ is the input space consisting of lists of feature vectors and $\mathcal{Y}$ is the output space consisting of lists of grades. Further suppose that $\mathbf{x}$ is an element of $\mathcal{X}$ representing a list of feature vectors and $\mathbf{y}$ is an element of $\mathcal{Y}$ representing a list of grades. Let $P(X, Y)$ be an unknown joint probability distribution where random variable X takes $\mathbf{x}$ as its value and random variable Y takes $\mathbf{y}$ as its value.

Assume that F is a function mapping from a list of feature vectors $\mathbf{x}$ to a list of scores. The goal of the learning task is to automatically learn a function $\hat{F}(\mathbf{x})$, given training data $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$. Each training instance is comprised of feature vectors $\mathbf{x}_i$ and the corresponding grades $\mathbf{y}_i$ $(i = 1, \cdots, m)$. Here m denotes the number of training instances.

$F(\mathbf{x})$ and $\mathbf{y}$ can be further written as $F(\mathbf{x}) = [f(x_1), f(x_2), \cdots, f(x_n)]$ and $\mathbf{y} = [y_1, y_2, \cdots, y_n]$. Here $f(x)$ denotes a local ranking function and n denotes the number of feature vectors and grades. The feature vectors correspond to the objects to be ranked, denoted as $O = [1, 2, \cdots, n]$.

We make use of a loss function $L(\cdot, \cdot)$ to evaluate the prediction result of $F(\mathbf{x})$. First, the feature vectors $\mathbf{x}$ are ranked according to $F(\mathbf{x})$. Then the ranking results are evaluated against the corresponding grades $\mathbf{y}$. If the feature vectors with higher grades are ranked higher, then the loss will be small. Otherwise, the loss will be large. The loss function is specifically represented as

$$L(F(\mathbf{x}), \mathbf{y}).$$

Note that the loss function for ranking is slightly different from the loss functions in other statistical learning tasks, in the sense that it makes use of sorting.

We further define the risk function $R(\cdot)$ as the expected loss function with respect to the joint distribution $P(X, Y)$,

$$R(F) = \int_{\mathcal{X} \times \mathcal{Y}} L(F(\mathbf{x}), \mathbf{y}) dP(\mathbf{x}, \mathbf{y}).$$

Given training data, we calculate the empirical risk function as follows,

$$\hat{R}(F) = \frac{1}{m}\sum_{i=1}^{m} L(F(\mathbf{x}_i), \mathbf{y}_i).$$

We can formalize the learning task as minimization of the empirical risk function, as in other learning tasks. We can also introduce regularizer to conduct minimization of the regularized empirical risk function.

The minimization of empirical risk function could be difficult due to the nature of the loss function (it is not continuous and it uses sorting). We can consider using a surrogate loss function denoted as

$$L'(F(\mathbf{x}), \mathbf{y}).$$

The corresponding risk function and empirical risk functions are defined as follows.

$$R'(F) = \int_{\mathcal{X}\times\mathcal{Y}} L'(F(\mathbf{x}), \mathbf{y}) dP(\mathbf{x}, \mathbf{y})$$

$$\hat{R}'(F) = \frac{1}{m}\sum_{i=1}^{m} L'(F(\mathbf{x}_i), \mathbf{y}_i).$$

In such case, the learning problem becomes that of minimization of (regularized) empirical risk function based on surrogate loss.

Note that we adopt a machine learning formulation here. In IR, the feature vectors $\mathbf{x}$ are derived from a query and its associated documents. The grades $\mathbf{y}$ represent the relevance degrees of the documents with respect to the query. We make use of a global ranking function $F(\mathbf{x})$. In practice, it is usually a local ranking function $f(x)$. The possible number of feature vectors in $\mathbf{x}$ can be very large, even infinite. The evaluation (loss function) is, however, only concerned with n results. In IR, n can be determined by the pooling strategy (cf., Section 2.2.2).

6.2 LOSS FUNCTIONS

In binary classification, the true loss function is usually 0-1 loss. In contrast, in ranking, there are different ways to define the true loss function. In IR, the true loss functions can be those defined based on NDCG (Normalized Discounted Cumulative Gain) and MAP (Mean Average Precision). Specifically, we have

$$L(F(\mathbf{x}), \mathbf{y}) = 1 - NDCG \tag{6.1}$$

and

$$L(F(\mathbf{x}), \mathbf{y}) = 1 - MAP. \tag{6.2}$$

Given permutation π by $F(\mathbf{x})$, NDCG of it (for n objects) is defined as follows.

$$NDCG = \frac{1}{G_{max}} \sum_{i:\pi(i)\leq n} G(i)D(\pi(i)) \tag{6.3}$$
$$G(i) = 2^{y_i} - 1, \quad D(\pi(i)) = \frac{1}{\log_2(1+\pi(i))},$$

where y_i is the grade of object i, $\pi(i)$ is the rank of object i in π, $G(\cdot)$ is the gain function, $D(\cdot)$ is the position discount function, and G_{max} is the normalizing factor.

Given permutation π by $F(\mathbf{x})$, MAP of it (for n objects)[1] is defined as follows.

$$MAP = \frac{\sum_{i=1}^{n} P(i)\cdot y_i}{\sum_{i=1}^{n} y_i},$$

where y_i is the grade of object i taking on 1 or 0 as value, $\pi(i)$ is the rank of object i in π, and $P(i)$ represents the precision until the rank of object i, defined as

$$P(i) = \frac{\sum_{j:\pi(j)\leq\pi(i)} y_j}{\pi(i)}.$$

Note that the true loss functions (NDCG loss and MAP loss) are not continuous, and they depend on sorting by $F(\mathbf{x})$.

For the surrogate loss function, there are also different ways to define it, which leads to different approaches to learning to rank. For example, one can define pointwise loss, pairwise loss, and listwise loss functions, respectively.

Squared Loss, which is a pointwise loss, is defined as

$$L'(F(\mathbf{x}), \mathbf{y}) = \sum_{i=1}^{n} (f(x_i) - y_i)^2.$$

The loss function is the one used in Subset Regression.

The pointwise loss in McRank is as follows

$$L'(F(\mathbf{x}), \mathbf{y}) = \sum_{i=1}^{n} I[\text{classifier}(f(x_i)) \neq y_i],$$

where $I[\cdot]$ is the indicator function and the output of classifier$(f(x_i))$ is a label (grade).

Pairwise losses can be hinge loss, exponential loss, and logistic loss as defined as follows. They are used in Ranking SVM, RankBoost, and RankNet, respectively.

[1]Here, we abuse terminology for ease of explanation. Mean Average Precision is in fact averaged over queries. The MAP here is only defined on one query. In that case, it should be called AP (Average Precision).

$$L'(F(\mathbf{x}),\mathbf{y}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}[1-\text{sign}(y_i-y_j)(f(x_i)-f(x_j))]_+, \quad \text{when } y_i \neq y_j, \tag{6.4}$$

where it is assumed that $L' = 0$, when $y_i = y_j$.

$$L'(F(\mathbf{x}),\mathbf{y}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\exp\left(-\text{sign}(y_i-y_j)(f(x_i)-f(x_j))\right), \quad \text{when } y_i \neq y_j. \tag{6.5}$$

$$L'(F(\mathbf{x}),\mathbf{y}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\log\left(1+\exp(-\text{sign}(y_i-y_j)(f(x_i)-f(x_j)))\right), \quad \text{when } y_i \neq y_j. \tag{6.6}$$

Listwise losses can be KL loss and logarithmic loss utilized in ListNet and ListMLE, respectively.

KL Loss in ListNet is defined as

$$L'(F(\mathbf{x}),\mathbf{y}) = D(P_{\mathbf{y}}(\pi)||P_F(\pi)), \tag{6.7}$$

where $D(\cdot||\cdot)$ is KL Divergence, $P_{\mathbf{y}}(\pi)$ is the permutation probability distribution (or top k probability distribution) induced by $\mathbf{y}$, and $P_F(\pi)$ is the permutation probability distribution (or top k probability distribution) induced by $F(\mathbf{x})$. Both distributions are calculated by the Plackett-Luce model.

Logarithmic Loss in ListMLE is defined as

$$L'(F(\mathbf{x}),\mathbf{y}) = -\log P_F(\pi^*_{\mathbf{y}}), \tag{6.8}$$

where $P_F(\pi^*_{\mathbf{y}})$ is the probability of perfect permutation by $\mathbf{y}$, calculated by $F(\mathbf{x})$ and the Plackett-Luce model.

Obviously, the surrogate loss function in AdaRank is also a listwise loss.

$$L'(F(\mathbf{x}),\mathbf{y}) = \exp(-NDCG),$$

where $NDCG$ is calculated on the basis of $F(\mathbf{x})$ and $\mathbf{y}$.

6.3 RELATIONS BETWEEN LOSS FUNCTIONS

Previous work has shown that the pointwise losses, pairwise losses (6.4-6.6) and listwise loss (6.8) in existing methods are upper bounds of the true losses (6.1-6.2).

$$L(F(\mathbf{x}),\mathbf{y}) \leq L'(F(\mathbf{x}),\mathbf{y}).$$

That means that existing learning to rank methods, such as Subset Ranking, McRank, Ranking SVM, RankBoost, RankNet, ListMLE, and AdaRank are methods of optimizing different surrogate loss functions.

Below we give a summary of the relations between the surrogate loss functions used in existing methods and the true loss function (1-$NDCG$).

The pointwise loss function in Subset Ranking is an upper bound of (1-$NDCG$) [30].

$$1 - NDCG \leq \frac{1}{G_{max}} \left(2 \sum_{i=1}^{n} D(\pi(i))^2 \right)^{1/2} L'(F(\mathbf{x}), \mathbf{y})^{1/2},$$

where $D(\pi(i))$ is the position discount of object i and $L'(F(\mathbf{x}), \mathbf{y})$ is the surrogate loss function in Subset Ranking.

The pointwise loss function in McRank is an upper bound of (1-$NDCG$) [71].

$$1 - NDCG \leq \frac{15\sqrt{2}}{G_{max}} \left(\sum_{i=1}^{n} D(\pi(i))^2 - n \prod_{i=1}^{n} D(\pi(i))^{2/n} \right)^{1/2} L'(F(\mathbf{x}), \mathbf{y})^{1/2},$$

where $D(\pi(i))$ is the position discount of object i and $L'(F(\mathbf{x}), \mathbf{y})$ is the surrogate loss function in McRank.

The pairwise loss functions in Ranking SVM, RankBoost, and RankNet are upper bounds of (1-$NDCG$) [21].

$$1 - NDCG \leq \frac{\max_i (G(i) D(\pi(i)))}{G_{max}} L'(F(\mathbf{x}), \mathbf{y}),$$

where $G(i)$ is the gain of object i and $D(\pi(i))$ is the position discount of object i and $L'(F(\mathbf{x}), \mathbf{y})$ is the surrogate loss function in the above pairwise methods.

The listwise loss function in ListMLE is an upper bound of (1-$NDCG$) [21].

$$1 - NDCG \leq \frac{\max_i (G(i) D(\pi(i)))}{\ln 2 \cdot G_{max}} L'(F(\mathbf{x}), \mathbf{y}),$$

where $G(i)$ is the gain of object i and $D(\pi(i))$ is the position discount of object i, and $L'(F(\mathbf{x}), \mathbf{y})$ is the surrogate loss function in ListMLE.

6.4 THEORETICAL ANALYSIS

There are two major issues with regard to theoretical analysis of learning to rank, namely generalization ability and statistical consistency.

Generalization ability of a method represents the relation between the empirical risk function and the expected risk function. It is usually represented by a bound between the two risk functions. Cossock & Zhang show the generalization ability of Subset Ranking. Lan et al. give

generalization bounds of Ranking SVM, IR SVM, ListNet, and ListMLE [63, 64]. Recently, Chen et al. have proved a generalization bound of pairwise methods, in a more natural framework [22].

Statistical consistency is to answer the question whether optimization of a surrogate loss function can lead to optimization of the true loss function. Xia et al. have studied the consistency of ListNet and ListMLE [114, 115].

For other work on theoretical analysis of learning to rank, see [2, 5, 26, 29].

CHAPTER 7

Ongoing and Future Work

Learning to rank is a hot area in machine learning and related fields, including information retrieval, natural language processing and data mining, and intensive study is being conducted.

In Chapter 4, methods of learning to rank have been described. It is still necessary to develop more advanced technologies. It is also clear from the discussions in Chapters 5 and 6, there are still many open questions with regard to theory and applications of learning to rank.

Let us look at some ongoing and future work on several topics with regard to learning to rank, particularly learning for ranking creation.

- Training data creation
- Semi-supervised learning and active learning
- Feature learning
- Scalable and efficient training
- Domain adaptation
- Ranking by ensemble learning
- Ranking with multiple measures
- Practical issues in search
- Ranking of objects in graph

TRAINING DATA CREATION

The quality of training data largely affects the performance of learning to rank, as in other machine learning tasks. If the quality of training data is low, then the accuracy of the trained model will also be low. The so-called 'garbage in garbage out' phenomenon also occurs in learning to rank. In addition, reducing the cost of training data construction is another issue which needs to be considered.

In IR, training data for ranking is usually annotated by humans, which is costly and error prone. As a result, the amount of training data tends to be small and the quality of data cannot be guaranteed.

As explained, one way to cope with the challenge is to automatically derive training data from click-through data. Click-through data represents users' implicit feedbacks and thus is a

valuable data source for training data creation. The problem which we need to address is to eliminate noise and position bias. For example, one can employ the method proposed by Joachims [59], to use the skips of documents in search as signals of relative relevance judgments. Another method developed by Radlinski and Joachims [93] can also be exploited, which makes use of the queries and clicks in search sessions. Specifically, a clicked document with the current query is preferred over an examined but not clicked document with the previous query in the same session, under the assumption that the user may have not found relevant result with the previous query.

Since training data labeled by human judges inevitably contains errors, another related issue is to automatically correct errors in the training data. One approach is to use click-through data to make error detection and correction. For example, Xu et al. [117] propose a method for detecting human labeling errors using click-through data. A discriminative model for predicting relevance labels from click-through patterns is employed.

For other methods with regard to training data creation, see also [4, 7, 94].

SEMI-SUPERVISED LEARNING AND ACTIVE LEARNING

Since creation of training data is expensive, using both labeled and unlabeled data in learning to rank naturally arises as an important issue to investigate. A key question then is how to leverage the useful information in the unlabeled data to enhance the performance of learning. Several methods on semi-supervised learning have been proposed [6, 35, 51, 58, 70]. Further investigations on the problem appear to be necessary.

Another related issue is active learning. Long et al. [77] point out that a general principle for active learning, named Expected Loss Minimization (ELO), can be employed in ranking just like in classification, regression, and other tasks. ELO suggests selecting the queries or documents with the largest expected losses. They propose an algorithm called ELO-DCG for active learning at both the query and document levels.

FEATURE LEARNING

In practice, the features used in the ranking model are more critical for the accuracy of learning to rank. Developing powerful features is an important step in building practical ranking systems.

In IR, BM25 and LM4IR (unsupervised ranking models) can be used as features of a ranking model. BM25 and LM4IR actually represent the relevance of query and document, using the matching degree of their terms. How to enrich a matching model and learn the model from data is an interesting topic. Metzler & Croft [80] propose employing a Markov Random Field model to represent the matching degree between query and document and to use the model in relevance ranking. The key idea is to take into account dependency between the terms in the query and represent their relations in a probabilistic dependency graph (MRF). An algorithm for learning the MRF model is also developed. See also [103].

PageRank is a document feature widely used in learning to rank. One can also think about enhancing the model. For example, Liu et al. propose exploiting user browsing graph built upon

user behavior data, constructing a continuous time Markov model on the graph, and calculating the stationary distribution as page importance. Their algorithm referred to as BrowseRank is a natural extension of PageRank [76].

More studies on supervised or unsupervised learning of features for ranking are certainly needed. We refer the reader to the survey paper on the topic [69], called semantic matching. Automatic selection of features also needs more investigations [44].

SCALABLE AND EFFICIENT TRAINING

Training data for learning to rank can also be large as in other learning tasks. How to make the training of learning to rank scalable and efficient is an important issue. Chapelle & Keerthi [17] have developed an efficient algorithm for training Ranking SVM. They employ the primal Newton method to speed up the training process and show that their implementation of Ranking SVM is five orders of magnitude faster than SVMLight, the widely used Ranking SVM learning tool.

DOMAIN ADAPTATION

Domain adaptation or transfer learning is a popular research topic in machine learning, which is also true for ranking. Another related issue is multi-task learning. There are domains for which it is easy to obtain training data and build reliable ranking models, while there are domains for which this is not the case. How to adapt a model trained in one domain to another domain then becomes important.

Methods for domain adaptation, transfer learning, and multi-task learning have been proposed [9, 18, 20, 42]. For example, Chapelle et al. [18] propose a Boosting algorithm to multi-task ranking. Their method learns a joint model for several different tasks, which addresses the specifics of each task with task-specific parameters and the commonalities among the tasks with shared parameters.

RANKING BY ENSEMBLE LEARNING

To enhance the accuracy of ranking, a divide-and-conquer approach can be effective. That is, for different queries in document retrieval one creates and utilizes different rankers and maximizes the overall ranking accuracy.

In general web search, users' search needs are very diverse, and thus it appears more necessary to adopt the query dependent approach. How to automatically classify queries into classes, train a ranking model for each class, and combine the ranking models becomes an important area to explore. Geng et al. [43] propose a query dependent ranking method. Given a query, the method tries to find the k nearest training queries and construct a ranking model with the data in the neighborhood. Efficient ways of performing k nearest neighbor training are given. There exist challenging yet interesting problems along the direction.

RANKING WITH MULTIPLE MEASURES

In document retrieval, ranking should be performed based on not only relevance, but also diversity, novelty, etc. Qin et al. [89] propose employing a Continuous Conditional Random Fields model for the purpose. The model represents documents and their scores as vertices and relations between document scores as edges in an undirected graph. A method for learning the CRF model from supervised learning data is also developed. They show that the CRF mode can effectively utilize similarity relation between documents and link relation between documents. Yue et al. [123] propose a method for conducting ranking based on diversity (for ambiguous queries like 'jaguar', it is better to rank the relevant documents of all the major senses on the top). The method takes the relevant documents as input and then groups them into diverse subsets. It formalizes the training of the model as a learning problem using Structural SVM. For other related work, see [32, 54, 56, 90, 95, 100].

PRACTICAL ISSUES IN SEARCH

Several practical issues need to be solved when applying learning to rank to document retrieval. Usually effectiveness of search (relevance) is the major focus of learning to rank. In practice, however, not only effectiveness but also efficiency are of importance for a search system. Wang et al.[112] propose a method for jointly optimizing effectiveness and efficiency in search. They learn a cascade ranking model, on the basis of Boosting, which exploits a sequence of increasingly complex ranking functions to continuously prune some irrelevant documents and re-rank the remaining documents. See also [111].

Usually one only labels a small set of documents for each query and uses the labeled data to train a ranking model. Lan et al. a [62] tries to answer the question of whether only utilizing the top-k ground truth is appropriate for learning to rank. They prove that the loss functions defined on top-k positions are tighter upper bounds of $1 - NDCG$ than those defined on the full list, and show that the performances of ranking algorithms quickly increase to a stable level when k increases. MacDonald et al. have studied the effect of sample size (number of documents per query) on ranking performance [79]. They find that ranking performance can vary according to the following factors: the information need (navigational v.s. informational) of query, the evaluation measure utilized in ranking, and the information used in pooling of documents. Their results indicate that a sufficiently large sample size is preferred for effective document retrieval. See also [83].

RANKING OF NODES IN GRAPH

Sometimes information on the relations between the objects to be ranked is also available. The relations are often represented in a directed or undirected graph on the objects. Therefore, how to leverage the information in ranking becomes an interesting question. This kind of setting is particularly common in social search and social data mining. Note that PageRank [84] and BrowseRank

[76] are also methods of ranking objects in a graph, but they only make use of the link information and are unsupervised learning methods.

Agrawal et al. [1], for example, propose a supervised learning method for ranking the objects in a graph. Their method employs the Markov random walk model, as in PageRank, and automatically learns the transition probabilities from the preference pairs of objects in the training data. The method formalizes the learning task as a constrained network flow problem in which the objective is maximum entropy, and the Markov property and the preference pairs are represented as constraints.

Bibliography

[1] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23, 2006. DOI: 10.1145/1150402.1150409 91

[2] Shivani Agarwal and Partha Niyogi. Stability and generalization of bipartite ranking algorithms. In *COLT*, pages 32–47, 2005. 86

[3] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 19–26, New York, NY, USA, 2006. ACM. DOI: 10.1145/1148170.1148177 17

[4] Rakesh Agrawal, Alan Halverson, Krishnaram Kenthapadi, Nina Mishra, and Panayiotis Tsaparas. Generating labels from clicks. In *WSDM*, pages 172–181, 2009. DOI: 10.1145/1498759.1498824 88

[5] Nir Ailon and Mehryar Mohri. An efficient reduction of ranking to classification. In *COLT*, pages 87–98, 2008. 86

[6] Massih Reza Amini, Tuong Vinh Truong, and Cyril Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 99–106, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390334.1390354 88

[7] Javed A. Aslam, Evangelos Kanoulas, Virgil Pavlu, Stefan Savev, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 468–475, New York, NY, USA, 2009. ACM. DOI: 10.1145/1571941.1572022 88

[8] Javed A. Aslam and Mark Montague. Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 276–284, New York, NY, USA, 2001. ACM. DOI: 10.1145/383952.384007 35, 71

[9] Jing Bai, Ke Zhou, Guirong Xue, Hongyuan Zha, Gordon Sun, Belle Tseng, Zhaohui Zheng, and Yi Chang. Multi-task learning for learning to rank in web search. In *CIKM*

'09: Proceeding of the 18th ACM conference on Information and knowledge management, pages 1549–1552, New York, NY, USA, 2009. ACM. DOI: 10.1145/1645953.1646169 89

[10] Michael Bendersky, W. Bruce Croft, and Yanlei Diao. Quality-biased ranking of web documents. In *WSDM*, pages 95–104, 2011. DOI: 10.1145/1935826.1935849

[11] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010. 29, 37, 70

[12] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005. DOI: 10.1145/1102351.1102363 26, 37, 50

[13] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems 18*, pages 395–402. MIT Press, Cambridge, MA, 2006. 29, 37, 54, 67

[14] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking SVM to document retrieval. In *SIGIR' 06*, pages 186–193, 2006. DOI: 10.1145/1148170.1148205 26, 37, 46

[15] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007. DOI: 10.1145/1273496.1273513 28, 37, 54

[16] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 88–96, New York, NY, USA, 2008. ACM. DOI: 10.1145/1401890.1401906 60

[17] Olivier Chapelle and S. Sathiya Keerthi. Efficient algorithms for ranking with svms. *Inf. Retr.*, 13(3):201–215, 2010. DOI: 10.1007/s10791-009-9109-9 89

[18] Olivier Chapelle, Pannagadatta K. Shivaswamy, Srinivas Vadrevu, Kilian Q. Weinberger, Ya Zhang, and Belle L. Tseng. Multi-task learning for boosting with application to web search ranking. In *KDD*, pages 1189–1198, 2010. DOI: 10.1145/1835804.1835953 89

[19] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Inf. Retr.*, 13(3):216–235, 2010. DOI: 10.1007/s10791-009-9110-3 27

[20] Depin Chen, Yan Xiong, Jun Yan, Gui-Rong Xue, Gang Wang, and Zheng Chen. Knowledge transfer for cross domain learning to rank. *Inf. Retr.*, 13(3):236–253, 2010. DOI: 10.1007/s10791-009-9111-2 89

[21] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *NIPS '09*, 2009. 85

[22] Wei Chen, Tie-Yan Liu, and Zhi-Ming Ma. Two-layer generalization analysis for ranking using rademacher average. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 370–378. 2010. 86

[23] Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier. Decision tree and instance-based learning for label ranking. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 161–168, New York, NY, USA, 2009. ACM. DOI: 10.1145/1553374.1553395 23

[24] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *J. Mach. Learn. Res.*, 6:1019–1041, 2005.

[25] Wei Chu and S. Sathiya Keerthi. New approaches to support vector ordinal regression. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 145–152, 2005. DOI: 10.1145/1102351.1102370 23

[26] Stéphan J.M. Clémençon and Nicolas Vayatis. Empirical performance maximization for linear rank statistics. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 305–312. 2009. 86

[27] W. William Cohen, R. E. Schapire, and Yoram Singer. Learning to order things. *JAIR*, 10:243–270, 1999. 35

[28] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. DOI: 10.1023/A:1022627411411 41

[29] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. Magnitude-preserving ranking algorithms. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 169–176, New York, NY, USA, 2007. ACM. DOI: 10.1145/1273496.1273518 86

[30] David Cossock and Tong Zhang. Subset ranking using regression. In *COLT '06: Proceedings of the 19th Annual Conference on Learning Theory*, pages 605–619, 2006. DOI: 10.1007/11776420_44 24, 85

[31] Koby Crammer and Yoram Singer. Pranking with ranking. In *NIPS*, pages 641–647, 2001. 21, 23, 37

[32] N. Dai, M. Shokouhi, and B. D. Davison. Learning to rank for freshness and relevance. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 95–104. ACM, 2011. 90

[33] Anlei Dong, Yi Chang, Zhaohui Zheng, Gilad Mishne, Jing Bai, Ruiqiang Zhang, Karolina Buchner, Ciya Liao, and Fernando Diaz. Towards recency ranking in web search. In *WSDM*, pages 11–20, 2010. DOI: 10.1145/1718487.1718490 77

[34] Pinar Donmez, Krysta M. Svore, and Christopher J.C. Burges. On the local optimality of lambdarank. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 460–467, New York, NY, USA, 2009. ACM. DOI: 10.1145/1571941.1572021 37, 67

[35] Kevin Duh and Katrin Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 251–258, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390334.1390379 88

[36] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 613–622, New York, NY, USA, 2001. ACM. DOI: 10.1145/371920.372165 35, 37, 73

[37] Jonathan L. Elsas, Vitor R. Carvalho, and Jaime G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 55–64, New York, NY, USA, 2008. ACM. DOI: 10.1145/1341531.1341542 26

[38] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999. 60

[39] Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003. DOI: 10.1162/jmlr.2003.4.6.933 26, 78

[40] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189-1232, 2001. DOI: 10.1214/aos/1013203450 29, 41, 50

[41] Jianfeng Gao, Haoliang Qi, Xinsong Xia, and Jian-Yun Nie. Linear discriminant model for information retrieval. In *SIGIR*, pages 290–297, 2005. DOI: 10.1145/1076034.1076085 11

[42] Wei Gao, Peng Cai, Kam-Fai Wong, and Aoying Zhou. Learning to rank only using training data from related domain. In *Proceeding of the 33rd international ACM SIGIR*

conference on Research and development in information retrieval, SIGIR '10, pages 162–169, New York, NY, USA, 2010. ACM. DOI: 10.1145/1835449.1835478 89

[43] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390334.1390356 22, 89

[44] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 407–414, New York, NY, USA, 2007. ACM. DOI: 10.1145/1277741.1277811 89

[45] John Guiver and Edward Snelson. Learning to rank with softrank and gaussian processes. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 259–266, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390334.1390380 37, 64

[46] John Guiver and Edward Snelson. Bayesian inference for plackett-luce ranking models. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 377–384, New York, NY, USA, 2009. ACM. DOI: 10.1145/1553374.1553423 54

[47] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 576–587. VLDB Endowment, 2004.

[48] Edward F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *ICML*, pages 250–257, 2003. 78

[49] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Support vector learning for ordinal regression. May 20 1999. DOI: 10.1049/cp:19991091 37, 41

[50] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large Margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000. 26, 37, 41

[51] Steven C. H. Hoi and Rong Jin. Semi-supervised ensemble ranking. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 634–639. AAAI Press, 2008. 88

[52] Yunhua Hu, Hang Li, Yunbo Cao, Dmitriy Meyerzon, and Qinghua Zheng. Automatic extraction of titles from general documents using machine learning. In Mary Marlino, Tamara Sumner, and Frank M. Shipman III, editors, *JCDL*, pages 145–154. ACM, 2005. 17

[53] Yunhua Hu, Guomao Xin, Ruihua Song, Guoping Hu, Shuming Shi, Yunbo Cao, and Hang Li. Title extraction from bodies of HTML documents and its application to web page retrieval. In Ricardo A. Baeza-Yates, Nivio Ziviani, Gary Marchionini, Alistair Moffat, and John Tait, editors, *SIGIR*, pages 250–257. ACM, 2005. 17

[54] Jim C. Huang and Brendan J. Frey. Structured ranking learning using cumulative distribution networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 697–704. 2009. 90

[55] Kalervo Järvelin and Jaana Kekäläinen. In evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, pages 41–48, New York, NY, USA, 2000. ACM. DOI: 10.1145/345508.345545 17

[56] Shihao Ji, Ke Zhou, Ciya Liao, Zhaohui Zheng, Gui-Rong Xue, Olivier Chapelle, Gordon Sun, and Hongyuan Zha. Global ranking by exploiting user clicks. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42, New York, NY, USA, 2009. ACM. DOI: 10.1145/1571941.1571950 90

[57] Xin Jiang, Yunhua Hu, and Hang Li. A ranking approach to keyphrase extraction. In James Allan, Javed A. Aslam, Mark Sanderson, ChengXiang Zhai, and Justin Zobel, editors, *SIGIR*, pages 756–757. ACM, 2009. 78

[58] Rong Jin, Hamed Valizadegan, and Hang Li. Ranking refinement and its application to information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 397–406, New York, NY, USA, 2008. ACM. DOI: 10.1145/1367497.1367552 88

[59] T. Joachims. Optimizing search engines using clickthrough data. In *KDD' 02*, pages 133–142, 2002. DOI: 10.1145/775047.775067 15, 88

[60] Thorsten Joachims, Hang Li, Tie-Yan Liu, and ChengXiang Zhai. Learning to rank for information retrieval (LR4IR 2007). *SIGIR Forum*, 41(2):58–62, 2007. DOI: 10.1145/1328964.1328974 4

[61] Alexandre Klementiev, Dan Roth, and Kevin Small. Unsupervised rank aggregation with distance-based models. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 472–479, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390156.1390216 35

[62] Y. Lan, S. Niu, J. Guo, and X. Cheng. Is top-k sufficient for ranking? In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1261–1270. ACM, 2013. 90

[63] Yanyan Lan, Tie-Yan Liu, Zhiming Ma, and Hang Li. Generalization analysis of listwise learning-to-rank algorithms. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 577–584, New York, NY, USA, 2009. ACM. 86

[64] Yanyan Lan, Tie-Yan Liu, Tao Qin, Zhiming Ma, and Hang Li. Query-level stability and generalization in learning to rank. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 512–519, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390156.1390221 86

[65] Quoc V. Le and Alexander J. Smola. Direct optimization of ranking measures. *CoRR*, abs/0704.3359, 2007. informal publication; informal publication. 60

[66] Guy Lebanon and John D. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 363–370, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. 35, 37, 74

[67] Hang Li, Tie-Yan Liu, and ChengXiang Zhai. Learning to rank for information retrieval (LR4IR 2008). *SIGIR Forum*, 42(2):76–79, 2008. DOI: 10.1145/1480506.1480519 4

[68] Hang Li, Tie-Yan Liu, and ChengXiang Zhai. Learning to rank for information retrieval (LR4IR 2009). *SIGIR Forum*, 43(2):41–45, 2009. DOI: 10.1145/1670564.1670571 4

[69] H. Li and J. Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 8, 2014. 89

[70] Ming Li, Hang Li, and Zhi-Hua Zhou. Semi-supervised document retrieval. *Inf. Process. Manage*, 45(3):341–355, 2009. DOI: 10.1016/j.ipm.2008.11.002 88

[71] Ping Li, Christopher Burges, and Qiang Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 897–904. MIT Press, Cambridge, MA, 2008. 21, 23, 37, 40, 85

[72] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. DOI: 10.1561/1500000016 4

[73] Tie-Yan Liu, Thorsten Joachims, Hang Li, and Chengxiang Zhai. Introduction to special issue on learning to rank for information retrieval. *Inf. Retr.*, 13(3):197–200, 2010. DOI: 10.1007/s10791-009-9120-1 4

[74] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007. DOI: 10.1561/1500000016 4

[75] Yu-Ting Liu, Tie-Yan Liu, Tao Qin, Zhi-Ming Ma, and Hang Li. Supervised rank aggregation. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 481–490, New York, NY, USA, 2007. ACM. DOI: 10.1145/1242572.1242638 35

[76] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li. Browserank: letting web users vote for page importance. In Sung-Hyon Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat-Seng Chua, and Mun-Kew Leong, editors, *SIGIR*, pages 451–458. ACM, 2008. 89, 91

[77] Bo Long, Olivier Chapelle, Ya Zhang, Yi Chang, Zhaohui Zheng, and Belle L. Tseng. Active learning for ranking through expected loss optimization. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *SIGIR*, pages 267–274. ACM, 2010. 88

[78] Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 437–444, New York, NY, USA, 2006. ACM. DOI: 10.1145/1148170.1148246 22

[79] C. Macdonald, R. L. Santos, and I. Ounis. The whens and hows of learning to rank for web search. *Information Retrieval*, 16(5):584–628, 2013. 90

[80] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479, New York, NY, USA, 2005. ACM Press. DOI: 10.1145/1076034.1076115 88

[81] Donald Metzler and Tapas Kanungo. Machine learned sentence selection strategies for query-biased summarization. sigir learning to rank workshop, 2008. 79

[82] Taesup Moon, Alex J. Smola, Yi Chang, and Zhaohui Zheng. Intervalrank: isotonic regression with listwise and pairwise constraints. In *WSDM*, pages 151–160, 2010. DOI: 10.1145/1718487.1718507 26

[83] S. Niu, J. Guo, Y. Lan, and X. Cheng. Top-k learning to rank: labeling, ranking and evaluation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 751–760. ACM, 2012. 90

[84] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University, Stanford, CA*, 1998. 17, 90

[85] Ashok Kumar Ponnuswami, Kumaresh Pattabiraman, Qiang Wu, Ran Gilad-Bachrach, and Tapas Kanungo. On composition of a federated web search result page: using online users to provide pairwise preference for heterogeneous verticals. In *WSDM*, pages 715–724, 2011. DOI: 10.1145/1935826.1935922 77

[86] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM. DOI: 10.1145/290941.291008 11

[87] Tao Qin, Tie-Yan Liu, and Hang Li. A general approximation framework for direct optimization of information retrieval measures. *Inf. Retr.*, 13(4):375–397, 2010. DOI: 10.1007/s10791-009-9124-x 28

[88] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13(4):346–374, 2010. DOI: 10.1007/s10791-009-9123-y 29

[89] Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. Global ranking using continuous conditional random fields. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1281–1288. 2009. 90

[90] Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, Wen-Ying Xiong, and Hang Li. Learning to rank relational objects and its application to web search. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 407–416, New York, NY, USA, 2008. ACM. DOI: 10.1145/1367497.1367553 90

[91] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. Query-level loss functions for information retrieval. *Inf. Process. Manage.*, 44(2):838–855, 2008. DOI: 10.1016/j.ipm.2007.07.016 27

[92] Tao Qin, Xu-Dong Zhang, De-Sheng Wang, Tie-Yan Liu, Wei Lai, and Hang Li. Ranking with multiple hyperplanes. In *Proceedings of the 30th annual international ACM SIGIR conference*, pages 279–286, 2007. DOI: 10.1145/1277741.1277791 26

[93] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, 2005. DOI: 10.1145/1081870.1081899 16, 88

[94] Filip Radlinski and Thorsten Joachims. Active exploration for learning rankings from clickthrough data. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 570–579, New York, NY, USA, 2007. ACM. DOI: 10.1145/1281192.1281254 16, 88

[95] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *ICML '08: Proceedings of the 25th international confer-*

ence on Machine learning, pages 784–791, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390156.1390255 90

[96] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc. 11, 16

[97] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. DOI: 10.1037/h0042519 37

[98] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press. 21, 23, 37, 38, 40

[99] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *HLT-NAACL*, pages 177–184, 2004. 79

[100] K. M. Svore, M. N. Volkovs, and C. J. Burges. Learning to rank with multiple objective functions. In *Proceedings of the 20th international conference on World wide web*, pages 367–376. ACM, 2011. 90

[101] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 295–302, New York, NY, USA, 2007. ACM. DOI: 10.1145/1277741.1277794

[102] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 77–86, New York, NY, USA, 2008. ACM. DOI: 10.1145/1341531.1341544 28, 37, 64

[103] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, New York, NY, USA, 2006. ACM. DOI: 10.1145/1183614.1183698 88

[104] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual international ACM SIGIR conference*, pages 383–390, 2007. DOI: 10.1145/1277741.1277808 27

[105] Nicolas Usunier, David Buffoni, and Patrick Gallinari. Ranking with ordered weighted pairwise classification. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1057–1064, New York, NY, USA, 2009. ACM. DOI: 10.1145/1553374.1553509 26

[106] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. Learning to rank by optimizing ndcg measure. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1883–1891. 2009. 27

[107] Maksims N. Volkovs and Richard S. Zemel. Boltzrank: learning to maximize expected ranking gain. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1089–1096, New York, NY, USA, 2009. ACM. DOI: 10.1145/1553374.1553513 27

[108] M. N. Volkovs and R. S. Zemel. A flexible generative model for preference aggregation. In *Proceedings of the 21st international conference on World Wide Web*, pages 479–488. ACM, 2012. 35

[109] M. N. Volkovs and R. S. Zemel. New learning methods for supervised and unsupervised preference aggregation. *Journal of Machine Learning Research*, 15:1135–1176, 2014. 35

[110] Ellen M. Voorhees and Donna Harman. *TREC: Experiment and Evaluation in Information Retrieval*. MIT, 2005. 17

[111] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 138–145. ACM, 2010. 90

[112] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 105–114. ACM, 2011. 90

[113] Qiang Wu, Christopher J. C. Burges, Krysta Marie Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, 2010. DOI: 10.1007/s10791-009-9112-1 29, 37, 70

[114] Fen Xia, Tie-Yan Liu, and Hang Li. Statistical consistency of top-k ranking. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2098–2106. 2009. 86

[115] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390156.1390306 28, 37, 54, 86

[116] Biao Xiang, Daxin Jiang, Jian Pei, Xiaohui Sun, Enhong Chen, and Hang Li. Context-aware ranking in web search. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *SIGIR*, pages 451–458. ACM, 2010. 77

[117] Jingfang Xu, Chuanliang Chen, Gu Xu, Hang Li, and Elbio Renato Torres Abib. Improving quality of training data for learning to rank using click-through data. In Brian D. Davison, Torsten Suel, Nick Craswell, and Bing Liu, editors, *WSDM*, pages 171–180. ACM, 2010. 88

[118] Jun Xu, Yunbo Cao, Hang Li, and Min Zhao. Ranking definitions with supervised learning methods. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 811–819, New York, NY, USA, 2005. ACM. DOI: 10.1145/1062745.1062761 78

[119] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, New York, NY, USA, 2007. ACM. DOI: 10.1145/1277741.1277809 28, 37, 59, 60

[120] Jun Xu, Hang Li, and Chaoliang Zhong. Relevance ranking using kernels. In Pu-Jen Cheng, Min-Yen Kan, Wai Lam, and Preslav Nakov, editors, *AIRS*, volume 6458 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2010.

[121] Jun Xu, Tie-Yan Liu, Min Lu, Hang Li, and Wei-Ying Ma. Directly optimizing evaluation measures in learning to rank. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 107–114, New York, NY, USA, 2008. ACM. DOI: 10.1145/1390334.1390355 28, 60

[122] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference*, pages 271–278, 2007. DOI: 10.1145/1277741.1277790 28, 37, 60

[123] Yisong Yue and T. Joachims. Predicting diverse subsets using structural SVMs. In *International Conference on Machine Learning (ICML)*, pages 271–278, 2008. DOI: 10.1145/1390156.1390310 90

[124] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22:179–214, April 2004. DOI: 10.1145/984321.984322 11

[125] Zhaohui Zheng, Hongyuan Zha, Keke Chen, and Gordon Sun. A regression framework for learning ranking functions using relative relevance judgments. In *Proceedings of the 30th*

annual international ACM SIGIR conference, 2007. DOI: 10.1145/1277741.1277792 37, 48

[126] Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. A general boosting method and its application to learning ranking functions for web search. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1697–1704. MIT Press, Cambridge, MA, 2008. 26, 37, 48

[127] Ke Zhou, Gui-Rong Xue, Hongyuan Zha, and Yong Yu. Learning to rank with ties. In *Proceedings of the 31st annual international ACM SIGIR conference*, pages 275–282, 2008. DOI: 10.1145/1390334.1390382 26

Author's Biography

HANG LI

Hang Li is chief scientist of the Noah's Ark Lab of Huawei Technologies. He is also adjunct professors of Peking University and Nanjing University. His research areas include information retrieval, natural language processing, statistical machine learning, and data mining. He graduated from Kyoto University in 1988 and earned his PhD from the University of Tokyo in 1998. He worked at the NEC lab in Japan during 1991 and 2001, and Microsoft Research Asia during 2001 and 2012. He joined Huawei Technologies in 2012. Hang has published three technical books and more than 100 scientific papers at top international journals including CL, NLE, TOIS, IPM, IRJ, TWEB, TKDE, and TIST, and top international conferences including SIGIR, WWW, WSDM, ACL, EMNLP, ICML, NIPS, and SIGKDD. He and his colleagues' papers received the SIGKDD'08 best application paper award, the SIGIR'08 best student paper award, and the ACL'12 best student paper award. Hang worked on the development of several products such as Microsoft SQL Server 2005, Microsoft Office 2007 and Office 2010, Microsoft Live Search 2008, Microsoft Bing 2009, Bing 2010. He has more than 35 granted US patents. Hang has also been very active in the research communities and is serving top international conferences including SIGIR, WWW, WSDM, ACL, EMNLP, NIPS, SIGKDD, and ICDM, and top international journals including CL, TIST, IRJ, JASIST, and JCST.