Foundations and Trends® in
sample
Vol. xx, No xx (2008) 1–112
© 2008 xxxxxxxxx
DOI: xxxxxx

Learning to Rank for Information Retrieval

Tie-Yan Liu[1]

[1] *Microsoft Research Asia, Sigma Center, No. 49, Zhichun Road, Haidian District, Beijing, 100190, P. R. China, Tie-Yan.Liu@microsoft.com*

Abstract

Learning to rank for information retrieval (IR) is a task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance. Many IR problems are by nature ranking problems, and many IR technologies can be potentially enhanced by using learning-to-rank techniques. The objective of this tutorial is to give an introduction to this research direction. Specifically, the existing learning-to-rank algorithms are reviewed and categorized into three approaches: the pointwise, pairwise, and listwise approaches. The advantages and problems with each approach are analyzed, and the relationships between the loss functions used in these approaches and IR evaluation measures are discussed. Then the empirical evaluations on typical learning-to-rank methods are shown, with the LETOR collection as a benchmark dataset, which seem to suggest that the listwise approach be the most effective one among all the approaches. After that, a statistical ranking theory is introduced, which can describe different learning-to-rank algorithms, and be used to analyze their query-level generalization abilities. At the end of the tutorial, we make a summary and discuss potential future work on learning to rank.

Contents

1

Introduction

With the fast development of the Web, every one of us is experiencing the flood of information. It was estimated that there are about 25 billion pages on the Web as of October 2008[1], which makes it generally impossible for common users to locate their desired information by browsing the Web. As a consequence, efficient and effective information retrieval (IR) has become more important than ever, and search engines (or IR systems) have become an essential tool for many people.

Ranking is a central problem in IR. Many IR problems are by nature ranking problems, such as document retrieval, collaborative filtering [58], key term extraction [30], definition finding [130], important email routing [23], sentiment analysis [94], product rating [36], and anti Web spam [56]. In this tutorial, we will mainly take document retrieval as an example. Note that document retrieval is not a narrow task. Web pages, emails, academic papers, books, and news stories are just a few of the many examples of documents. And there are also many different ranking scenarios for document retrieval of our interest.

Scenario 1: Rank the documents purely according to their rele-

[1] http://www.worldwidewebsize.com/

vance with regards to the query.

Scenario 2: Consider the relationships of similarity [118], website structure [35], and diversity [139] between documents in the ranking process. This is also referred to as relational ranking [102].

Scenario 3: Aggregate several candidate ranked lists to get a better ranked list. This scenario is also referred to as meta search [7]. The candidate ranked lists may come from different index servers, or different vertical search engines, and the target ranked list is the final result presented to users.

Scenario 4: Find whether and to what degree a property of a webpage influences the ranking result. This is referred to as "reverse engineering" in search engine optimization (SEO)[2].

To tackle the problem of document retrieval, many heuristic ranking models have been proposed and used in the literature of IR. Recently, given the amount of potential training data available, it has become possible to leverage machine learning (ML) technologies to build effective ranking models. Specifically, we call those methods that learn how to combine predefined features for ranking by means of discriminative learning "learning-to-rank" methods.

In recent years, learning to rank has become a very hot research direction in IR, and a large number of learning-to-rank algorithms have been proposed, such as [49] [73] [33] [90] [78] [34] [59] [114] [26] [9] [29] [14] [122] [47] [62] [97] [16] [117] [136] [134] [13] [104] [99] [17] [129]. We can foresee that learning to rank will have an even bigger impact on IR in the future.

When a research area comes to this stage, several questions as follows naturally arise.

- To what respect are these learning-to-rank algorithms similar and in which aspects do they differ? What are the strengths and weaknesses of each algorithm?
- Empirically, which of those many learning-to-rank algorithms perform the best? What kind of datasets can be used to make

[2] http://www.search-marketing.info/newsletter/reverse-engineering.htm

fair comparison among different learning-to-rank algorithms?
- Theoretically, is ranking a new machine learning problem, or can it be simply reduced to existing machine learning problems? What are the unique theoretical issues for ranking that should be investigated?
- Are there many remaining issues regarding learning to rank to study in the future? What are they?

The above questions have been brought to the attention of the IR and ML communities in a variety of contexts, especially during recent years. The aim of this tutorial is to review recent work that attempts to answer these questions. Needless to say, the comprehensive understanding of the task of ranking in IR is the key to finding the right answers. Therefore, we will first give a brief introduction of ranking in IR, and then formalize the problem of learning to rank so as to set the stage for the upcoming detailed reviews.

1.1 Ranking in Information Retrieval

In this section, we briefly review representative ranking models in the literature of IR, and introduce how these models are evaluated.

1.1.1 Conventional Ranking Models for IR

In the literature of IR, many ranking models have been proposed [8]. They can be roughly categorized as query-dependent models and query-independent models.

Query-dependent Models

The early models retrieve documents based on the occurrences of the query terms in the documents. Examples include the *Boolean model* [8]. Basically these models can only predict whether a document is relevant to the query or not, but cannot predict the degree of relevance.

To further model the relevance degree, the *Vector Space model* (VSM) was proposed [8]. Both documents and queries are represented as vectors in a Euclidean space, in which the inner product of two vectors can be used to measure their similarities. To get an effective vector representation of the query and the documents, TF-IDF weighting has been widely used[3]. The TF of a term t in a vector is defined as the normalized number of its occurrences in the document, and the IDF of it is defined as follows.

$$IDF(t) = \log \frac{N}{n(t)}, \tag{1.1}$$

where N is the total number of documents in the collection, and $n(t)$ is the number of documents containing term t.

While VSM implies the assumption on the independence between terms, *Latent Semantic Indexing* (LSI) [37] tries to avoid this assumption. In particular, Singular Value Decomposition (SVD) is used to linearly transform the feature space and thus a "latent semantic space" is generated. Similarity in this new space is then used to define the relevance between the query and the documents.

As compared with the above, models based on the probabilistic ranking principle [83] garnered more attention and achieved more success in the past decades. The famous ranking models like the *BM25 model* [111][4] and the *language model for IR*, can both be categorized as probabilistic ranking models.

The basic idea of BM25 is to rank documents by the log-odds of their relevance. Actually BM25 is not a single model, but actually defines a whole family of ranking models, with slightly different components and parameters. One of the popular instantiations of the model is as follows.

Given a query q, containing terms $t_1, \ldots, t_M$, the BM25 score of a document d is computed as below,

$$BM25(d,q) = \sum_{i=1}^{M} \frac{IDF(t_i) \cdot TF(t_i,d) \cdot (k_1 + 1)}{TF(t_i,d) + k_1 \cdot (1 - b + b \cdot \frac{LEN(d)}{avdl})}, \tag{1.2}$$

[3] Note that there are many different definitions of TF and IDF in the literature. Some are purely based on the frequency and the others include smoothing or normalization [116]. Here we just give some simple examples to illustrate the main idea.

[4] The name of the actual model is BM25. To the right context, however, it is usually referred to as "OKapi BM25", since the OKapi system was the first system to implement this model.

where $TF(t, d)$ is the term frequency of t in document d, $LEN(d)$ is the length (number of words) of document d, and $avdl$ is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters, $IDF(t)$ is the IDF weight of term t, computed by using Eqn.(1.1), for example.

The language model for IR [96] is an application of the statistical language model on IR. A statistical language model assigns a probability to a sequence of terms. When used in IR, a language model is associated with a document. With query q as input, documents are ranked based on the query likelihood, or the probability that the document's language model would generate the terms in the query (i.e., $P(q|d)$). By further assuming the independence among terms, one has $P(q|d) = \prod_{i=1}^{M} P(t_i|d)$, if query q contains terms $t_1, \cdots, t_M$.

To learn the document's language model, a maximum likelihood method is used. As in many maximum likelihood methods, the issue of smoothing the estimate is critical. Usually a background language model estimated using the entire collection is used for this purpose. Then, the document's language model can be constructed as follows.

$$p(t_i|d) = (1-\lambda)\frac{TF(t_i, d)}{LEN(d)} + \lambda p(t_i|C), \tag{1.3}$$

where $p(t_i|C)$ is the background language model for term t_i, and $\lambda \in [0, 1]$ is a smoothing factor.

There are many variants of the language model for IR, some of them even go beyond the query likelihood retrieval model (e.g., the models based on K-L divergence [140]). We would not introduce more about them, and readers are encouraged to read the tutorial authored by Zhai [138].

In addition to the above examples, many other models have also been proposed to compute the relevance between a query and a document. Some of them [119] take the proximity of the query terms into consideration, and some others consider the relationship between documents in terms of content similarity [118], hyperlink structure [113], website structure [101], and topic diversity [139].

Query-independent Models

In the literature of IR, there are also many models that rank documents based on their own importance. We will take PageRank [92] as an example for illustration. This model is particularly applicable to Web search because it makes use of the hyperlink structure of the Web for ranking.

PageRank uses the probability that a surfer randomly clicking on links will arrive at a particular webpage to rank the web pages. In the general case, the PageRank value for any page d_u can be expressed as

$$PR(d_u) = \sum_{d_v \in B_u} \frac{PR(d_v)}{U(d_v)}. \tag{1.4}$$

That is, the PageRank value for page d_u is dependent on the PageRank values for each page d_v out of the set B_u (containing all pages linking to page d_u), divided by $U(d_v)$, the number of outlinks from page d_v.

To get a meaningful solution of Eqn.(1.4), a smoothing term is introduced. When the random surfer walks on the link graph, she/he does not necessarily always follow the existing hyperlinks. There is a small probability that she/he will jump to any other page uniformly. This small probability can be represented by $(1-\alpha)$, where α is called the damping factor. Accordingly, the PageRank model is refined as follows,

$$PR(d_u) = \alpha \sum_{d_v \in B_u} \frac{PR(d_v)}{U(d_v)} + \frac{(1-\alpha)}{N}, \tag{1.5}$$

where N is the total number of pages on the Web.

There are many works discussing the theoretical properties, variations, and efficient implementations of the PageRank model. Furthermore, there are also many other link analysis algorithms, such as Hyperlink Induced Topic Search (HITS) [72] and TrustRank [57]. Some of these algorithms even leverage the content or topic information in the process of link analysis [91].

1.1.2 Query-level Position-based Evaluations in IR

Given the large number of ranking models as introduced in the previous subsection, a standard evaluation mechanism is needed to select the most effective model. For this purpose, one usually proceeds as follows.

- Collect a large number of (randomly sampled) queries to form a test set.
- For each query q,
  - Collect documents $\{d_j\}_{j=1}^{m}$ associated with the query.
  - Get the relevance judgment for each document by human assessment.
  - Use a given ranking model to rank the documents.
  - Measure the difference between the ranking results and the relevance judgment using an evaluation measure.
- Use the average measure on all the queries in the test set to evaluate the performance of the ranking model.

As for collecting the documents associated with a query, a number of strategies can be used. For example, one can simply collect all the documents containing the query word. One can also choose to use some predefined rankers to get documents that are more likely to be relevant. A popular strategy is the pooling method used in TREC[5]. In this method a pool of possibly relevant documents is created by taking a sample of documents selected by the various participating systems. In particular, top 100 documents retrieved in each submitted run for a given query are selected and merged into the pool for human assessment. On average, an assessor judges the relevance of approximately 1500 documents per query.

As for the relevance judgment, three strategies were used in the literature.

(1) Specifying whether a document is relevant or not to the query (i.e., binary judgment 1 or 0), or further specifying the degree

[5] http://trec.nist.gov/

of relevance (i.e., multiple ordered categories, e.g., Perfect, Excellent, Good, Fair, or Bad). Suppose for document d_j associated with query q, we get its relevance judgment as l_j. Then for two documents d_u and d_v, if $l_u \succ l_v$, we say that document d_u is more relevant than document d_v, with regards to query q, according to the relevance judgment.

(2) Specifying whether a document is more relevant than another with regards to a query. For example, if document d_u is judged to be more relevant than document d_v with regards to query q, we give the judgment $l_{u,v} = +1$; otherwise, $l_{u,v} = -1$. That is, this kind of judgment captures the relative preference between documents[6].

(3) Specifying the partial order or even total order of the documents with respect to a query. For the set of documents $\{d_j\}_{j=1}^m$ associated with query q, this kind of judgment is usually represented as a certain permutation of these documents, denoted as π_l, or a set of such permutations.

Given the vital role that relevance judgments play in a test collection, it is important to assess the quality of the judgments. In previous practices like TREC, both the completeness and the consistency of the relevance judgments are of interest. Completeness measures the degree to which all the relevant documents for a topic have been found; consistency measures the degree to which the assessor has marked all the "truly" relevant documents relevant and the "truly" irrelevant documents irrelevant.

Since the manual judgment is always time consuming, it is almost impossible to judge all the documents with regards to a query. Consequently, there are always unjudged documents returned by the ranking model. As a common practice, one regards the unjudged documents as irrelevant in the evaluation process[7].

[6] This kind of judgment can also be mined from click-through logs of search engines [68] [69] [105].

[7] In recent years, several new evaluation mechanisms [18] that consider the relevance probability of an unjudged document have also been proposed.

With the relevance judgment, several evaluation measures have been proposed and used in the literature of IR. It is clear that understanding these measures will be very important for learning to rank, since to some extent they define the "true" objective function of ranking. Below we list some popularly used measures.

Mean Reciprocal Rank (MRR)

For query q, the rank position of its first relevant document is denoted as $r(q)$. Then $\frac{1}{r(q)}$ is defined as MRR for query q. It is clear that documents ranked below $r(q)$ are not considered in MRR.

Mean Average Precision (MAP)

To define MAP [8], one needs to define Precision at position k ($P@k$) first,

$$P@k(q) = \frac{\#\{\text{relevant documents in the top } k \text{ positions}\}}{k}. \tag{1.6}$$

Then the Average Precision (AP) is defined below,

$$\text{AP}(q) = \frac{\sum_{k=1}^{m} P@k(q) \cdot l_k}{\#\{\text{relevant documents}\}}, \tag{1.7}$$

where m is the total number of documents associated with query q, and l_k is the binary judgment on the relevance of the document at the k-th position. The mean value of AP over all the test queries is named mean average precision (MAP).

Discounted Cumulative Gain (DCG)

While the aforementioned measures are mainly designed for binary judgments, DCG [65] [66] can leverage the relevance judgment in terms of multiple ordered categories, and has an explicit position discount factor in its definition. More formally, suppose the ranked list for query q is π, then DCG at position k is defined as follows,

$$\text{DCG}@k(q) = \sum_{r=1}^{k} G(\pi^{-1}(r))\eta(r), \tag{1.8}$$

where $\pi^{-1}(r)$ denotes the document ranked at position r of the list π, $G(\cdot)$ is the rating of a document (one usually sets $G(\pi^{-1}(r)) = (2^{l_{\pi^{-1}(r)}} - 1))$, and $\eta(r)$ is a position discount factor (one usually sets $\eta(r) = 1/\log_2(r+1))$.

By normalizing DCG@k with the maximum value of it (denoted as Z_k), we will get another measure named Normalized DCG (NDCG). That is,

$$\text{NDCG@}k(q) = \frac{1}{Z_k}\sum_{r=1}^{k} G(\pi^{-1}(r))\eta(r). \tag{1.9}$$

It is clear that NDCG takes value from 0 to 1.

Rank Correlation (RC)

The correlation between the ranked list given by the model (denoted as π) and the relevance judgment (denoted as π_l) can be used to define a measure. For example, when the weighted Kendall's τ is used, the rank correlation measures the weighted pairwise inconsistency between two lists. Its definition is given below,

$$\tau_K(q) = \frac{\sum_{u<v} w_{u,v}(1 + \text{sgn}((\pi(u) - \pi(v))(\pi_l(u) - \pi_l(v))))}{2\sum_{u<v} w_{u,v}}, \tag{1.10}$$

where $w_{u,v}$ is the weight, and $\pi(u)$ means the rank position of document d_u in permutation π.

To summarize, there are some common properties in these evaluation measures.

(1) All these evaluation measures are calculated at the *query level*. That is, first the measure is computed for each query, and then averaged over all queries in the test set. No matter how poorly the documents associated with a particular query are ranked, it will not dominate the evaluation process since each query contributes similarly to the average measure.
(2) All these measures are *position based*. That is, rank position is explicitly used. Considering that with small changes in the scores given by a ranking model the rank positions

will not change until one document's score passes another, the position based measures are usually non-continuous and non-differentiable with regards to the scores. This makes the optimization of these measures quite difficult. We will conduct more discussions on this in Section 4.1.

1.2 Learning to Rank

Many ranking models have been introduced in the previous section. Most of these models contain parameters. For example, there are parameters k_1 and b in BM25 (see Eqn.(1.2)), parameter λ in language models for IR (see Eqn.(1.3)), and parameter α in PageRank (see Eqn.(1.5)). In order to get a reasonably good ranking performance (in terms of IR evaluation measures), one needs to tune these parameters using a validation set. Nevertheless, the parameter tuning is far from trivial, especially considering that IR evaluation measures are non-continuous and non-differentiable with respect to the parameters. In addition, a model perfectly tuned on the validation set sometimes performs poorly on unseen test queries. This is usually called over-fitting. Another issue is about the combination of these ranking models. Given that many models have been proposed in the literature, it is natural to investigate how to combine these models and create an even more effective new model. This is, however, not straightforward either.

While IR researchers were suffering from these problems, machine learning has been demonstrating its effectiveness in automatically tuning parameters, in combining multiple evidences, and in avoiding over-fitting. Therefore, it seems quite promising to adopt machine learning technologies to solve the aforementioned problems.

1.2.1 Machine Learning Framework

In many machine learning researches (especially discriminative learning), attention has been paid to the following key components[8].

(1) The *input space*, which contains the objects under investi-

[8] For a more comprehensive introduction to the machine learning literature, please refer to [89].

gation. Usually objects are represented by feature vectors, extracted according to different applications.

(2) The *output space*, which contains the learning target with respect to the input objects. There are two related but different definitions of the output space in machine learning[9]. The first is the output space of the task, which is highly dependent on the application. For example, in regression, the output space is the space of real numbers $\mathbb{R}$; in classification it is the set of discrete categories $\{0, 1, \cdots, K-1\}$. The second is the output space to facilitate the learning process. This may differ from the output space of the task. For example, one can even use regression technologies to solve the problem of classification. In this case, the output space that facilitates learning is the space of real numbers but not discrete categories.

(3) The *hypothesis* space, which defines the class of functions mapping the input space to the output space. That is, the functions operate on the feature vectors of the input objects, and make predictions according to the format of the output space.

(4) In order to learn the optimal hypothesis, a training set is usually used, which contains a number of independent and identically distributed (i.i.d.) objects and their ground truth labels, sampled from the product of the input and output spaces. The *loss function* measures to what degree the prediction generated by the hypothesis is in accordance with the ground truth label. For example, widely used loss functions for classification include the exponential loss, the hinge loss, and the logistic loss. It is clear that the loss function plays a central role in machine learning, since it encodes the understanding of the target application (i.e., what prediction is correct and what is not). With the loss function, an empirical risk can be defined on the training set, and the optimal hypothesis is usually (but not always) learned by means of empirical risk minimization.

[9] In this tutorial, when we mention the output space, we mainly refer to the second type.

1.2.2 Learning-to-Rank Framework

In recent years, more and more machine learning technologies have been used to train the ranking model, and a new research area named "learning to rank" has gradually emerged. Especially in the past several years, learning to rank has become one of the most active research areas in IR.

In general, we can call all those methods that use machine learning technologies to solve the problem of ranking "learning-to-rank" methods[10], such as the work on relevance feedback[11] [112] [39] and automatically tuning the parameters of existing IR models [120] [60]. However, most of the state-of-the-art learning-to-rank algorithms learn the optimal way of combining features exacted from query-document pairs through discriminative training. Therefore, in this tutorial we define learning to rank in a more specific way to better summarize these algorithms. We call those ranking methods that have the following two properties learning-to-rank methods.

Feature based

"*Feature based*" means that all the documents under investigation are represented by feature vectors[12], reflecting the relevance of the documents to the query. That is, for a given query q, its associated document d can be represented by a vector $x = \Phi(d, q)$, where Φ is a feature extractor. Typical features used in learning to rank include the frequencies of the query terms in the document, the outputs of the BM25 model and the PageRank model, and even the relationship between this document and other documents. If one wants to know more about widely used features, please refer to Tables 6.2 and 6.3 in Chapter 6.

[10] In the literature of machine learning, there is a topic named label ranking. It is to predict the ranking of multiple class labels for an individual document, but not to predict the ranking of documents. In this regard, it is largely different from the task of ranking for IR.

[11] We will make further discussions on the relationship between relevance feedback and learning to rank in Chapter 2.

[12] Note that, hereafter in this tutorial, when we refer to a document, we will not use d any longer. Instead, we will directly use its feature representation x. Furthermore, since our discussions will focus more on the learning process, we will always assume the features are pre-specified, and will not purposely discuss how to extract them.

Even if a feature is the output of an existing retrieval model, in the context of learning to rank, one assumes that the parameter in the model is fixed, and only the optimal way of combining these features is learned. In this sense, the previous works on automatically tuning the parameters of existing models [60] [120] are not categorized as “learning-to-rank” methods.

The capability of combining a large number of features is a very important advantage of learning-to-rank methods. It is easy to incorporate any new progress on retrieval model, by including the output of the model as one dimension of the features. Such a capability is highly demanding for real search engines, since it is almost impossible to use only a few factors to satisfy complex information needs of Web users.

Discriminative training

“*Discriminative training*” means that the learning process can be well described by the four components of discriminative learning as mentioned in the previous subsection. That is, a learning-to-rank method has its specific input space, output space, hypothesis space, and loss function.

In the literature of machine learning, discriminative methods have been widely used to combine different kinds of features, without the necessity of defining a probabilistic framework to represent the objects and the correctness of prediction. In this sense, previous works that train generative ranking models are not categorized as “learning-to-rank” methods in this tutorial. If one has interest in such works, please refer to [74], [141], [85], etc.

Discriminative training is an automatic learning process based on the training data. This is also highly demanding for real search engines, because everyday these search engines will receive a lot of user feedback and usage logs indicating the poor ranking for some queries or documents. It is very important to automatically learn from the feedback and constantly improve the ranking mechanism.

Due to the aforementioned two characteristics, learning to rank has been widely used in commercial search engines[13], and has also

[13] See http://blog.searchenginewatch.com/050622-082709,

attracted great attention from academic research community.

Figure 1.1 shows the typical "learning to rank" flow. From the figure we can see that since learning to rank is a kind of supervised learning, a training set is needed. The creation of a training set is very similar to the creation of the test set for evaluation. For example, a typical training set consists of n training queries $q_i (i = 1, \cdots n)$, their associated documents represented by feature vectors $\mathbf{x}^{(i)} = \{x_j^{(i)}\}_{j=1}^{m^{(i)}}$ (where $m^{(i)}$ is the number of documents associated with query q_i), and the corresponding relevance judgments[14]. Then a specific learning algorithm is employed to learn the ranking model (i.e., the way of combining the features), such that the output of the ranking model can predict the ground truth label in the training set[15] as accurately as possible, in terms of a loss function. In the test phase, when a new query comes in, the model learned in the training phase is applied to sort the documents according to their relevance to the query, and return the corresponding ranked list to the user as the response to her/his query.

1.2.3 Approaches to Learning to Rank

Many learning-to-rank algorithms can fit into the above framework. In order to better understand them, we perform a categorization on these algorithms. In particular, we group the algorithms, according to the four pillars of machine learning, into three approaches: the pointwise approach, the pairwise approach, and the listwise approach. Note that different approaches model the process of learning to rank in different ways. That is, they define different input and output spaces, use different hypotheses, and employ different loss functions. Note that, the output space is used to facilitate the learning process, which can be different from the relevance judgments on the documents. That is, even if

http://blogs.msdn.com/msnsearch/archive/2005/06/21/431288.aspx, and http://glinden.blogspot.com/2005/06/msn-search-and-learning-to-rank.html.

[14] Please distinguish the judgment for evaluation and the judgment for constructing the training set, although the process may be very similar.

[15] Hereafter, when we mention the *ground truth labels* in the remainder of the tutorial, we will mainly refer to the ground truth labels in the training set, although we assume every document will have its intrinsic label no matter it is judged or not.

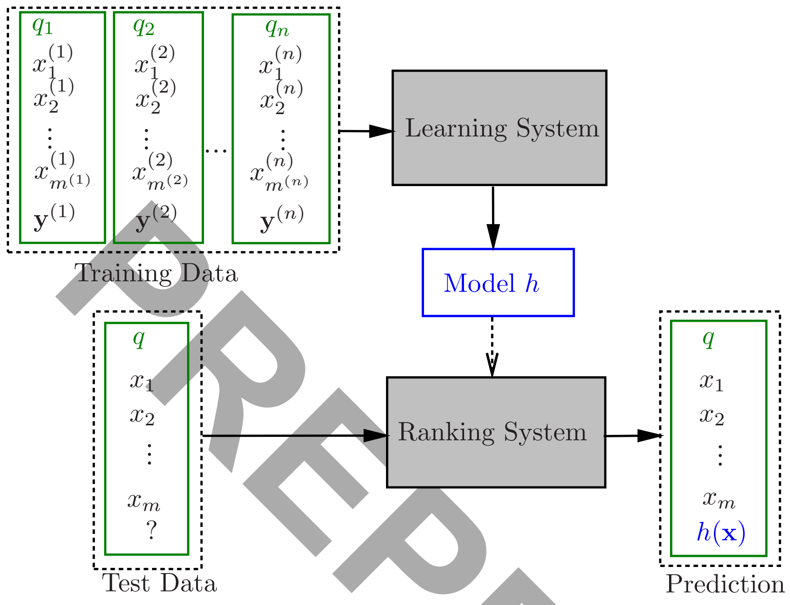

Fig. 1.1 Learing to Rank Framework

provided with the same format of judgments, one can derive different ground truth labels from it, and use them for different approaches.

The Pointwise Approach

The *input space* of the pointwise approach contains feature vector of each single document.

The *output space* contains the relevance degree of each single document. The ground truth label in the output space is usually defined in the following way. If the judgment is directly given as relevance degree l_j, the ground truth label for document x_j is defined as $y_j = l_j$. If the judgment is given as the total order π_l, one can get the ground truth label by using a mapping function[16]. However, if the judgment is given as pairwise preference $l_{u,v}$, it is not straightforward to make use of it to generate the ground truth label.

[16] For example, the position of the document in π_l can be used to define the relevance degree.

The *hypothesis* space contains functions that take the feature vector of a document as input, and predict the relevance degree of the document. We usually call such a function f the scoring function. Note that, based on the scoring function, one can sort all the documents and produce the final ranked list.

The *loss function* examines the accurate prediction of the ground truth label for each single document. In different pointwise ranking algorithms, ranking is modeled as regression, classification, and ordinal regression respectively (see Chapter 2). And therefore the corresponding regression loss, classification loss, and ordinal regression loss are used as the loss function. Note that the pointwise approach does not consider the inter-dependency among documents, and thus the position of a document in the final ranked list is invisible to its loss function. Furthermore, the approach does not make use of the fact that some documents are actually associated with the same query. Considering that most IR evaluation measures are query-level and position-based, intuitively speaking, the pointwise approach has its limitations.

Example algorithms belonging to the pointwise approach include [49] [73] [33] [31] [53] [90] [78] [34] [114] [24] [26] [25]. We will introduce some of them in Chapter 2.

The Pairwise Approach

The *input space* of the pairwise approach contains a pair of documents, both represented by feature vectors.

The *output space* contains the pairwise preference (which takes value from $\{+1, -1\}$) between each pair of documents. The ground truth label in the output space is usually defined in the following way. If the judgment is give as relevance degree l_j, then the order for document pair (x_u, x_v) can be defined as $y_{u,v} = 2 \cdot I_{\{l_u \succ l_v\}} - 1$. Here $I_{\{A\}}$ is an indicator function, which is defined to be 1 if predicate A holds and 0 otherwise. If the judgment is given directly as pairwise preference, then it is straightforward to set $y_{u,v} = l_{u,v}$. If the judgment is given as the total order π_l, one can define $y_{u,v} = 2 \cdot I_{\{\pi_l(u) < \pi_l(v)\}} - 1$.

The *hypothesis* space contains bi-variate functions h that take a pair of documents as input, and output the relative order between them. Some pairwise ranking algorithms directly define their hypotheses as such [29], however, in more algorithms, the hypothesis is still defined with a scoring function f for simplicity, i.e., $h(x_u, x_v) = 2 \cdot I_{\{f(x_u)>f(x_v)\}} - 1$.

The *loss function* measures the inconsistency between $h(x_u, x_v)$ and the ground truth label $y_{u,v}$. In some algorithms, ranking is modeled as a pairwise classification, and the corresponding classification loss on a pair of documents is used as the loss function. Note that the loss function used in the pairwise approach only considers the relative order between two documents. When one looks at only a pair of documents, however, the position of the documents in the final ranked list can hardly be derived. Furthermore, the approach ignores the fact that some pairs are generated from the documents associated with the same query. Considering that most IR evaluation measures are query-level and position-based, intuitively speaking, there is still a gap between this approach and ranking for IR.

Example algorithms belonging to the pairwise approach include [9] [29] [14] [122] [47] [62] [97] [16]. We will introduce some of them in Chapter 3.

The Listwise Approach

The *input space* of the listwise approach contains the entire group of documents associated with query q, e.g., $\mathbf{x} = \{x_j\}_{j=1}^m$.

There are two types of *output spaces* used in the listwise approach. For some listwise ranking algorithms, the *output space* contains the relevance degrees of all the documents associated with a query. In this case, the ground truth label $\mathbf{y} = \{y_j\}_{j=1}^m$ can be derived from the judgment in terms of relevance degree or total order, in a similar manner to that of the pointwise approach. For some other listwise ranking algorithms, the *output space* contains the ranked list (or permutation) of the documents. In this case, the ground truth label, denoted as π_y, can be generated in the following way. When the judgment is given as total order π_l, we can define $\pi_y = \pi_l$. Otherwise, we can also derive π_y by using the concept of *equivalent permutation set* (see Chapter 4). Note that when the ranked list comprises the output space that facilitates the learning process, the space is actually the same as the output space of the task. Therefore, the theoretical analysis on the listwise approach can have more direct value to understanding the real ranking problem than the other approaches where there are mismatches between the output space that facilitates learning and the real output space of the task.

The *hypothesis* space contains multivariate functions h that operate on a group of documents, and predict their relevance degrees or their permutation. For practical reasons, the hypothesis h is also usually implemented with a scoring function f. When the relevance degree comprises the output space, $h(\mathbf{x}) = f(\mathbf{x})$. When the ranked list (permutation) comprises the output space, h is defined as a compound function $h(\mathbf{x}) = \text{sort} \circ f(\mathbf{x})$. That is, first a scoring function f is used to give a score to each document, and then these documents are sorted in the descending order of the scores to produce the desired permutation.

There are also two types of *loss functions*, corresponding to the two types of output spaces. When the ground truth label is given as $\mathbf{y}$, the loss function is usually defined as an approximation or a bound of widely-used IR evaluation measures. When the ground truth label is given as π_y, the loss function measures the difference between the permutation given by the hypothesis and the ground truth permutation. As compared to the pointwise and pairwise approaches, the advantage of the listwise approach lies in that its loss function can naturally consider the positions of documents in the ranked list of all the documents associated with the same query. In this sense, the approach is in more accordance with the ranking task in IR.

Example algorithms that belong to the listwise approach include [117] [136] [134] [13] [104] [99] [17] [129]. We will introduce some of them in Chapter 4.

It is noted that different loss functions are used in different approaches, while the same IR evaluation measures are used for testing their performances. Then a natural question is about the relationship between these loss functions and the IR evaluation measures. The investigation on this can help us explain the empirical results of these algorithms. We will introduce some of such investigations in Chapter 5. In addition, in Chapter 6, we will introduce a benchmark dataset for the research on learning to rank, named LETOR, and report some empirical results of representative learning-to-rank algorithms on the dataset.

Furthermore, one may have noticed that the scoring function, which is widely used in defining the hypotheses of different approaches, is a kind of "pointwise" function. However, it is not to say that all the approaches are in nature pointwise approach. The categorization of the aforementioned three approaches is mainly based on the four pillars of machine learning. That is, different approaches regard the same training data as in different input and output spaces, and define different loss functions and hypotheses accordingly. From the machine learning point of view, they have different assumptions on the i.i.d. distribution of the data and therefore the theoretical properties (e.g., generalization ability) of their corresponding algorithms will be largely different. We will make more discussions on this in Chapter 7, with the introduction of a new theory, which we call the statistical ranking theory.

1.3 About this Tutorial

As for the writing of the tutorial, we do not aim to be fully rigorous. Instead we try to provide insights into the basic ideas. However, it is still unavoidable that we will use mathematics for better illustration of the problem, especially when we jump into the theoretical discussions on learning to rank. We will have to assume familiarity with basic concepts of probability theory and statistical learning in the corresponding discussions.

Furthermore, we will use the notation rules as listed in Table 1.1 throughout the tutorial. Here we would like to add one more note. Since in practice, the hypothesis h is usually defined with scoring function f, we sometimes use $L(h)$ and $L(f)$ interchangeably to represent the loss function. When we need to emphasize the parameter in the scoring function f, we will use $f(w, x)$ instead of $f(x)$ in the discussion, although they actually mean the same thing.

Table 1.1: Notation Rules

Meaning	Notation
Query	q, or q_i
A quantity z for query q_i	$z^{(i)}$
Number of training queries	n
Number of documents associated with query q	m
Number of document pairs associated with query q	$\tilde{m}$
Feature vector of a document associated with query q	x
Feature vectors of documents associated with query q	$\mathbf{x} = \{x_j\}_{j=1}^m$
Term frequency of query q in document d	$TF(q,d)$
Inverse document frequency of query q	$IDF(q)$
Length of document d	$LEN(d)$
Hypothesis	$h(\cdot)$
Scoring function	$f(\cdot)$
Loss function	$L(\cdot)$
Expected risk	$R(\cdot)$
Empirical risk	$\hat{R}(\cdot)$
Relevance degree for document x_j	l_j
Document x_u is more relevant than document x_v	$l_u \succ l_v$
Pairwise preference between documents x_u and x_v	$l_{u,v}$
Total order of document associated with the same query	π_l
Ground truth label for document x_j	y_j
Ground truth label for document pair $(x_u,\, x_v)$	$y_{u,v}$
Ground truth list for documents associate with query q	π_y
Ground truth permutation set for documents associate with query q	Ω_y
Original document index of the j-th element in permutation π	$\pi^{-1}(j)$
Rank position of document j in permutation π	$\pi(j)$
Number of classes	K
Index of class	k
VC dimension of a function class	V
Indicator function	$I_{\{\cdot\}}$
Gain function	$G(\cdot)$
Position discount function	$\eta(\cdot)$

2

The Pointwise Approach

When using the technologies of machine learning to solve the problem of ranking, probably the most straightforward way is to check whether existing learning methods can be directly applied. This is exactly what the pointwise approach does. When doing so, one assumes that the exact relevance degree of each document is what we are going to predict, although this may not be necessary when the target is to produce a ranked list of the documents.

According to different machine learning technologies used, the pointwise approach can be further divided into three subcategories: regression based algorithms, classification based algorithms and ordinal regression based algorithms. For regression based algorithms, the output space contains real-valued relevance scores; for classification based algorithms, the output space contains non-ordered categories; and for ordinal regression based algorithms, the output space contains ordered categories. Documents together with their ground truth labels in the training set are regarded as i.i.d. random variables sampled from the product of the input and output spaces.

In the following, we will first introduce representative algorithms in the three subcategories of the pointwise approach, and then make discussions on their advantages and problems.

2.1 Regression based Algorithms

In this subcategory, the problem of ranking is reduced to a problem of regression, by regarding the relevance degree as real numbers. Here we introduce two representative algorithms as examples.

Polynomial Regression Function

This is an early work on learning to rank [49], which uses least square regression to learn the scoring function f.

Given a set of documents $\mathbf{x} = \{x_j\}_{j=1}^m$ associated with a training query q, the ground truth label for x_j is defined as a vector. For binary judgments, $\vec{y}_j = (1, 0)$ if the document is judged as relevant, and $\vec{y}_j = (0, 1)$ otherwise. For multiple ordered categories, the k-th element of the vector $\vec{y}_j$ is set to 1 and the other elements are set to 0, if the document is judged as belonging to the k-th category.

Then, the scoring function is defined as $\vec{f} = (f_1, f_2, \cdots)$, with each element f_k as a predictor of the k-th element in $\vec{y}_j$. Here, f_k is supposed to be selected from the polynomial function class, i.e.,

$$\begin{aligned} f_k(x_j) = w_{k,0} + w_{k,1} \cdot x_{j,1} + \cdots + w_{k,T} \cdot x_{j,T} \\ +w_{k,T+1} \cdot x_{j,1}^2 + w_{k,T+2} \cdot x_{j,1} \cdot x_{j,2} + \cdots , \end{aligned} \tag{2.1}$$

where $w_{k,l}$ is the combination coefficient, $x_{j,l}$ is the l-th feature in the feature vector x_j, and T is the number of features in the representation of a document.

Next, the loss function is defined as the following square loss,

$$L(\vec{f}; x_j, \vec{y}_j) = \|\vec{y}_j - \vec{f}(x_j)\|^2. \tag{2.2}$$

Suppose we are given the binary judgment. Then the loss function indicates that for a relevant document, only if the scoring function can exactly output $(1, 0)$, there will be zero loss. Otherwise even if the output is $(2, 0)$ which seems to be an even stronger prediction of relevance for this document, there will be some loss. This is, in some sense, not very reasonable.

Subset Ranking with Regression

Cossock and Zhang [33] also solves the problem of ranking by reducing it to regression.

Given $\mathbf{x} = \{x_j\}_{j=1}^m$, a set of documents associated with training query q, and the ground truth labels $\mathbf{y} = \{y_j\}_{j=1}^m$ of these documents in terms of multiple ordered categories, suppose a scoring function f is used to rank these documents. The authors define the loss function as the following regression loss when performing learning to rank,

$$L(f; x_j, y_j) = (y_j - f(x_j))^2. \tag{2.3}$$

Furthermore, they have conducted some theoretical analysis on the use of such a loss function. The basic conclusion is that the regression loss can bound the ranking error (1−NDCG) (See Chapter 5 for more discussions).

However, it is clear that this work also suffers from the same problem as the polynomial regression function [49]. In many cases, it is not right to accurately predict the value of y_j since y_j is actually a qualitative judgment but not a quantitative value at all.

2.2 Classification based Algorithms

Analogously to reducing ranking to regression, one can also consider reducing ranking to a classification problem. Since the classification based algorithms do not regard the ground truth label as a quantitative value, it is more reasonable than the regression based algorithms. Here we introduce two representative algorithms in this subcategory.

Discriminative Model for IR

While most conventional ranking models for IR can be regarded as generative models (e.g., the language model for IR), the literature of machine learning has shown that discriminative models are preferred over generative models in many recent situations. Therefore, it is worth trying to find whether discriminative classifiers can lead to retrieval performances similar to or even better than retrieval performances of those generative IR ranking models. Actually, there have been several works that studied the use of discriminative classification model for relevance ranking in IR, such as [31], [53] and [90]. Here we take [90] as an example to illustrate the basic idea.

Given documents $\mathbf{x} = \{x_j\}_{j=1}^m$, and their binary relevance judgments $\mathbf{y} = \{y_j\}_{j=1}^m$ associated with query q, one regards all the relevant documents (i.e., $y_j = 1$) as positive examples while all the irrelevant documents (i.e., $y_j = 0$) as negative examples, and adopts the classification technology to learn the ranking model.

Two representative classification models, the Maximum Entropy (ME) model [54] and the Support Vector Machines (SVM) model [126] [125], were investigated in [90]. The principle of ME is to model all that is known and to assume nothing about the rest. So one can put all the information contained in the training data into a constraint set, and then maximize the entropy of the conditional probability with these constraints. SVM maximizes the margin on the constraint set of the training data. SVM has been proved to be one of the best classifiers in many classification tasks. It is also associated with a nice generalization theory based on the VC dimension, and therefore is theoretically guaranteed to have good performance even if the number of training samples is small.

Experiments on ad-hoc retrieval indicate that the ME-based algorithm is significantly worse than language models, but the SVM-based algorithm is comparable with and sometimes slightly better than language models. Based on this, the author argued that SVM is still preferred because of its ability to learn arbitrary features automatically, fewer assumptions, and expressiveness [90].

Multi-class Classification for Ranking (McRank)

Li, et al. [78] proposed using multi-class classification to solve the problem of ranking.

Given documents $\mathbf{x} = \{x_j\}_{j=1}^m$ associated with query q, and their relevance judgment $\mathbf{y} = \{y_j\}_{j=1}^m$. Suppose we have a multi-class classifier, which makes prediction $\hat{y}_j$ on x_j. Then the loss function used to learn the classifier is defined as an upper bound of the following 0-1 classification error,

$$L(\hat{y}_j, y_j) = I_{\{y_j \neq \hat{y}_j\}}. \tag{2.4}$$

In practice, different upper bounds yield different loss functions, such as the exponential loss, the hinge loss, and the logistic loss. All of them can be used to learn the classifier.

As for the testing process, the authors discussed how to convert classification results into ranking scores. In particular, the output of the classifier is converted to a probability using a logistic function, indicating the probability of a document belonging to a specific category. Suppose this probability is $P(\hat{y}_j = k), k = 0, \cdots, K-1$ (where K is the number of the categories given in the judgment), then the following weighted combination is used to determine the final ranking scores of a document.

$$f(x_j) = \sum_{k=0}^{K-1} k \cdot P(\hat{y}_j = k). \tag{2.5}$$

2.3 Ordinal Regression based Algorithms

Ordinal regression[1] takes the ordinal relationship among the ground truth labels into consideration when learning the ranking model.

Suppose there are K ordered categories. The goal of ordinal regression is to find a scoring function f, such that one can easily use thresholds $b_0 \leq b_2 \leq \cdots \leq b_{K-1}$ to distinguish the outputs of the scoring function (i.e., $f(x_j), j = 1, \cdots, m$) into different ordered categories[2], although this may not be necessary from the ranking point of view.

In the literature, there are several methods in this subcategory, such as [34], [114], [24], [26], and [25]. We will introduce some of them as follows.

[1] Ordinal regression sometimes was also directly referred to as "ranking" in previous works [115].

[2] Note that, there are some algorithms, such as [68], which were also referred to as ordinal regression based algorithms in the literature. According to our categorization, however, they belong to the pairwise approach since they do not really care about the accurate assignment of a document to one of the ordered categories. Instead, they focus more on the relative order between two documents.

Perceptron based Ranking (PRanking)

PRanking is a famous algorithm on ordinal regression [34]. The goal of PRanking is to find a direction defined by a parameter vector w, after projecting the documents onto which one can easily use thresholds to distinguish the documents into different ordered categories.

This goal is achieved by means of an iterative learning process. On iteration t, the learning algorithm gets an instance x_j associated with query q. Given x_j, the algorithm predicts $\hat{y}_j = \arg\min_k\{w^T x_j - b_k < 0\}$. It then receives the ground truth label y_j. If the algorithm makes a mistake by predicting the category of x_j as $\hat{y}_j$ instead of y_j then there is at least one threshold, indexed by k, for which the value of $w^T x_j$ is on the wrong side of b_k. To correct the mistake, we need to move the values of $w^T x_j$ and b_k toward each other.

Let us see an example shown in Figure 2.1. Suppose now we have model parameter w and document x_j. According to the output of the scoring function, this document seems to belong to the second category. However, its ground truth label indicates that it should belong to the fourth category. Then, the algorithm will lower down thresholds b_2 and b_3. After that, the model parameter w is adjusted as $w = w + x_j$, just as in many perceptron based algorithms. This process is repeated until the training process converges.

Harrington [59] later proposed using random sub-sampling to further improve the performance of PRanking. They first sub-sample the training data, and learn a PRanking model using each sample. After that, the weights and thresholds associated with the models are averaged to produce the final model. It has been proved that in this way a better generalization ability can be achieved [63].

Ranking with Large Margin Principles

Shashua and Levin [114] tried to use the SVM technology to learn the model parameter w and the thresholds $b_k (k = 0, \cdots, K-1)$, for ordinal regression.

Specifically, two strategies were proposed. The first one is referred to as the *fixed margin strategy*.

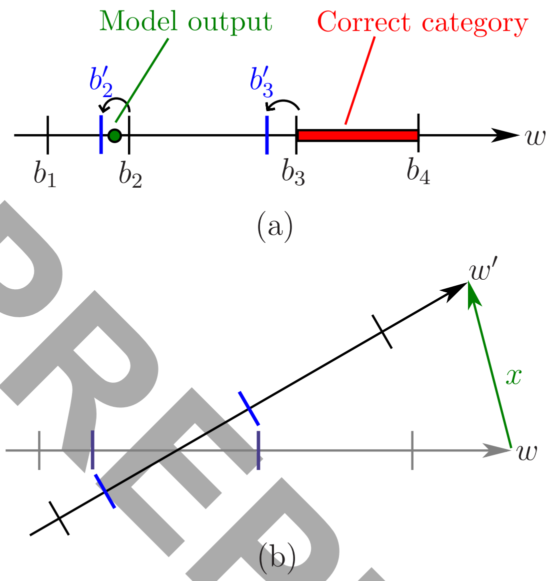

Fig. 2.1 Learning Process of PRanking

Given n training queries $\{q_i\}_{i=1}^n$, their associated documents $\mathbf{x}^{(i)} = \{x_j^{(i)}\}_{j=1}^{m^{(i)}}$, and the corresponding relevance judgments $\mathbf{y}^{(i)} = \{y_j^{(i)}\}_{j=1}^{m^{(i)}}$, the learning process is defined below, where the adoption of a linear scoring function is assumed. The constraints basically require every document to be correctly classified into its target ordered category, i.e., for documents in category k, $w^T x_j^{(i)}$ should exceed threshold b_{k-1} but be smaller than threshold b_k, with certain soft margins (i.e., $1 - \xi_{j,k-1}^{(i)*}$ and $1 - \xi_{j,k}^{(i)}$ respectively). The margin term $\frac{1}{2}\|w\|^2$ is borrowed from support vector machines, which can control the complexity of the model w.

$$\begin{aligned}
&\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\sum_{j=1}^{m^{(i)}}\sum_{k=0}^{K-2}(\xi_{j,k}^{(i)} + \xi_{j,k+1}^{(i)*}) \\
&\text{s.t. } w^T x_j^{(i)} - b_k \le -1 + \xi_{j,k}^{(i)}, \text{if } y_j^{(i)} = k, \\
&w^T x_j^{(i)} - b_k \ge 1 - \xi_{j,k+1}^{(i)*}, \text{if } y_j^{(i)} = k+1, \\
&\xi_{j,k}^{(i)} \ge 0, \xi_{j,k+1}^{(i)*} \ge 0, \\
&j = 1, \cdots, m^{(i)}, i = 1, \cdots, n, k = 0, \cdots, K-2.
\end{aligned} \tag{2.6}$$

The second strategy is called the *sum of margins strategy*. In this strategy, some additional thresholds a_k are introduced, such that for category k, b_{k-1} is its lower-bound threshold and a_k is its upper-bound threshold. Accordingly, the constraints become that for documents in category k, $w^T x_j^{(i)}$ should exceed threshold b_{k-1} but be smaller than threshold a_k, with certain soft margins (i.e., $1 - \xi_{j,k-1}^{(i)*}$ and $1 - \xi_{j,k}^{(i)}$ respectively). The corresponding learning process can be expressed as follows, from which we can see that the margin term $\sum_{k=0}^{K-1}(a_k - b_k)$ really has the meaning of "margin" (in Figure 2.2, $(b_k - a_k)$ is exactly the margin between category $k+1$ and category k).

$$\begin{aligned}
&\min \sum_{k=0}^{K-1}(a_k - b_k) + C\sum_{i=1}^{n}\sum_{j=1}^{m^{(i)}}\sum_{k=0}^{K-2}(\xi_{j,k}^{(i)} + \xi_{j,k+1}^{(i)*}) \\
&\text{s.t. } a_k \le b_k \le a_{k+1}, \\
&w^T x_j^{(i)} \le a_k + \xi_{j,k}^{(i)}, \text{if } y_j^{(i)} = k, \\
&w^T x_j^{(i)} \ge b_k - \xi_{j,k+1}^{(i)*}, \text{if } y_j^{(i)} = k+1, \\
&\|w\|^2 \le 1, \xi_{j,k}^{(i)} \ge 0, \xi_{j,k+1}^{(i)*} \ge 0, \\
&j = 1, \cdots, m^{(i)}, i = 1, \cdots, n, k = 0, \cdots, K-2.
\end{aligned} \tag{2.7}$$

Ideally in the above methods, one requires $b_k (k = 0, \cdots, K-1)$ to be in an increasing order, i.e., $b_{k-1} \le b_k$. However, in practice, since there are no clear constraints on the thresholds in the optimization problem, the learning process cannot always guarantee this. To tackle

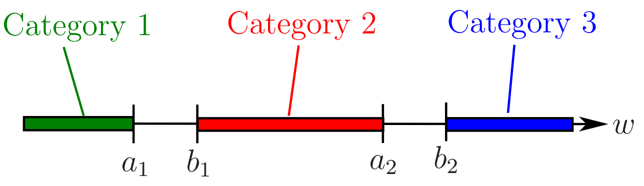

Fig. 2.2 Sum of Margin Strategy

the problem, Chu and Keerthi [26] proposed adding explicit or implicit constraints on the thresholds to the optimization problem. The explicit constraint simply takes the form of $b_{k-1} \leq b_k$, while the implicit constraint uses redundant training examples to guarantee the ordinal relationship among thresholds.

2.4 Discussions

In this section, we first discuss the relationship between the pointwise approach and some early learning method in IR, such as relevance feedback. Then, we discuss the problems with the pointwise approach by considering the distinct properties of IR.

2.4.1 Relationship with Relevance Feedback

The pointwise approach to learning to rank, especially the classification based algorithms, has strong correlation with the relevance feedback algorithms [112] [39]. The relevance feedback algorithms, which have played an important role in the literature of IR, also leverage supervised learning technologies to improve the retrieval accuracy. The basic idea is to learn from explicit, implicit, or blind feedback to update the original query. Then the new query is used to retrieve a new set of documents. By doing this in an iterative manner, we can bring the original query closer to the optimal query so as to improve the retrieval performance.

Here we take the famous Rocchio algorithm [112] as an example to make discussions on the relationship between relevance feedback and learning to rank. The specific way that the Rocchio algorithm updates the query is as follows. First, both the query q and its associated documents are represented in a vector space. Second, through relevance feedback, $\{x_j\}_{j=1}^{m^+}$ are identified as relevance documents (i.e., $y_j = 1$), and $\{x_j\}_{j=m^++1}^{m^++m^-}$ are identified as irrelevant documents (i.e., $y_j = 0$). Third, the query vector is updated according to the following heuristic,

$$\tilde{q} = \alpha q + \beta \frac{1}{m^+} \sum_{j=1}^{m^+} x_j - \gamma \frac{1}{m^-} \sum_{j=m^++1}^{m^++m^-} x_j. \tag{2.8}$$

If we use the VSM model for retrieval, the documents will then be ranked according to their inner product with the new query vector $\tilde{q}$. Mathematically, we can define the corresponding ranking function as

$$f(x_j) = \tilde{q}^T x_j. \tag{2.9}$$

In this sense, we can regard the query vector as the model parameter. For ease of discussion, we use w to represent this vector, i.e., $w = \tilde{q}$. Then, as shown in [77], there is actually a hidden loss function behind the above query update process, which is a function of w and x. That is,

$$L(f, x_j, y_j) = \begin{cases} \frac{1}{m^+}\left(\frac{1-\alpha}{4}\|w\|^2 - \beta w^T x_j\right), & y_j = 1, \\ \frac{1}{m^-}\left(\frac{1-\alpha}{4}\|w\|^2 + \gamma w^T x_j\right), & y_j = 0. \end{cases} \tag{2.10}$$

In other words, the Rocchio algorithm also minimizes a certain pointwise loss function. In this sense, it looks quite similar to the pointwise approach to learning to rank. However, we would like to point out its significant differences from what we call learning to rank, as shown below.

- The feature space in the Rocchio algorithm is the standard vector space as used in VSM. In this space, both query and documents are represented as vectors, and their inner product defines the relevance. In contrast, in learning to rank, the feature space contains features extracted from each query-document pair. Only documents have feature representations, and the query is not a vector in the same feature space.

- The Rocchio algorithm learns the model parameter from the feedback on a given query, and then uses the model to rank the documents associated with the same query. It does not consider the generalization of the model across queries. However, in learning to rank, we learn the ranking model from a training set, and mainly use it to rank the documents associated with unseen test queries.
- The model parameter w in the Rocchio algorithm actually has its physical meaning, i.e., it is the updated query vector. However, in learning to rank, the model parameter does not have such a meaning and only corresponds to the importance of each feature to the ranking task.
- The goal of the Rocchio algorithm is to update the query formulation for a better retrieval but not to learn an optimal ranking function. In other words, after the query is updated, the fixed ranking function (e.g., the VSM model) is used to return a new set of related documents.

2.4.2 Problems with the Pointwise Approach

Theoretically speaking, the algorithms belonging to the pointwise approach are highly correlated with each other. For example, when the number of categories $K = 2$, the ordinal regression will naturally reduce to a binary classification. Therefore these algorithms have similar problems when dealing with the task of learning to rank.

Since the input object in the pointwise approach is a single document for all the algorithms, the relative order between documents cannot be naturally considered in their learning processes, although ranking is more about predicting relative order than accurate relevance degree. Furthermore, the two intrinsic properties of the evaluation measures for ranking (i.e., query-level and position based) cannot be well considered by the pointwise approach. First, the fact is ignored in these algorithms that some documents are associated with the same query and some others are not. As a result, when the number of associated documents varies largely for different queries[3], the overall loss function will be dominated by those queries with a large number of documents. Second, the position of each document in the ranked list is invisible to the pointwise loss functions. Therefore, the pointwise loss function may unconsciously emphasize too much those unimportant documents (which are ranked low in the final ranked list and thus do not affect user experiences).

Considering the above problems, the pointwise approach can only be a sub-optimal solution to ranking. To tackle the problem, people have made attempts on regarding document pair, or the entire group of documents associated with the same query, as the input object. This results in the pairwise and listwise approaches to learning to rank. With the pairwise approach, the relative order among documents can be better modeled. With the listwise approach, the position information can be visible to the learning-to-rank process.

[3] For the re-ranking scenario, the number of documents to rank for each query may be very similar, e.g., top 1000 documents per query. However, if we consider all the documents containing the query word, the difference between the number of documents for popular queries and that for tail queries may be very large.

3

The Pairwise Approach

The pairwise approach[1] does not focus on accurately predicting the relevance degree of each document, instead, it cares about the relative order between two documents. In this sense, it is closer to the concept of "ranking" than the pointwise approach.

In the pairwise approach, ranking is reduced to a classification on document pairs. That is, the goal of learning is to minimize the number of miss-classified document pairs. In the extreme case, if all the document pairs are correctly classified, all the documents will be correctly ranked. Note that this classification differs from the classification in the pointwise approach, since it operates on every two documents under investigation. A natural concern is that document pairs are not independent, which violates the basic assumption of classification. The explanation is that although in some cases this assumption really does not hold, one can still use classification technology to learn the ranking model. However, a different theoretical framework is needed to analyze the generalization of the learning process. We will make discussions on this in Chapter 7.

[1] Also referred to as preference learning in the literature.

In the rest of this chapter, we will introduce several representative algorithms that belong to the pairwise approach.

3.1 Example Algorithms

Ordering with Preference Function

In [29], a hypothesis $h(x_u, x_v)$ directly defined on a pair of documents is studied (i.e., without use of the scoring function f). In particular, given two documents x_u and x_v associated with a training query q, the loss function is defined below,

$$L(h; x_u, x_v, y_{u,v}) = \frac{|y_{u,v} - h(x_u, x_v)|}{2}, \tag{3.1}$$

where the hypothesis is defined as $h(x_u, x_v) = \sum_t w_t h_t(x_u, x_v)$ and $h_t(x_u, x_v)$ is called the base preference function.

Suppose $h_t(x_u, x_v)$ only takes a value from $\{+1, -1\}$, where a value of $+1$ indicates that document x_u is ranked before x_v, and a value of -1 indicates the opposite. Then, we can easily find if the ground truth label indicates that document x_u should be ranked before document x_v (i.e., $y_{u,v} = +1$) but $h(x_u, x_v) = -1$, there will be a loss of one for this pair of documents. When all the pairs are incorrectly ranked, the average loss on the training set will reach its maximum value of one. On the other hand, when all the pairs are correctly ranked, we can get the minimum loss of zero.

With the above loss function, the weighted majority algorithm, e.g., the Hedge algorithm, is used to learn the parameters in the hypothesis h. Note that the hypothesis h is actually a preference function, which cannot directly output the ranked list of the documents. In this case, an additional step is needed to convert the pairwise preference between any two documents to the total order of all the documents. To this end, one needs to find the ranked list π, which has the largest agreement with the pairwise preferences. This process is described below.

$$\max_{\pi} \sum_{u<v} h(x_{\pi^{-1}(u)}, x_{\pi^{-1}(v)}). \tag{3.2}$$

As we know, this is a typical problem called rank aggregation. It has been proved NP-hard to find the optimal solution to the above optimization problem. To tackle the challenge, a greedy ordering algorithm was proposed in [29], which can be much more efficient, and its agreement with the pairwise preferences is at least half the agreement of the optimal algorithm.

RankNet and FRank

RankNet [14] is probably the first learning-to-rank algorithm used by commercial search engines[2].

In RankNet, the loss function is also defined on a pair of documents, but the hypothesis is defined with the use of a scoring function f. Given two documents x_u and x_v associated with a training query q, a target probability $\bar{P}_{u,v}$ is constructed based on their ground truth labels. For example, we can define $\bar{P}_{u,v} = 1$, if $y_{u,v} = +1$; $\bar{P}_{u,v} = 0$, otherwise. Then, the modeled probability $P_{u,v}$ is defined based on the difference between the scores of these two documents given by the scoring function, i.e.,

$$P_{u,v}(f) = \frac{\exp(f(x_u) - f(x_v))}{1 + \exp(f(x_u) - f(x_v))}. \tag{3.3}$$

Then the cross entropy between the target probability and the modeled probability is used as the loss function, which we refer to as the *cross entropy loss* for short.

$$L(f; x_u, x_v, y_{u,v}) = -\bar{P}_{u,v} \log P_{u,v}(f) - (1 - \bar{P}_{u,v}) \log (1 - P_{u,v}(f)). \tag{3.4}$$

A neural network is then used as the model, gradient descent as the optimization algorithm to learn the scoring function f. In [84], a nested ranker is built on top of RankNet to further improve the

[2] As far as we know, Microsoft Live Search (http://www.live.com/) is using the model trained with a variation of RankNet.

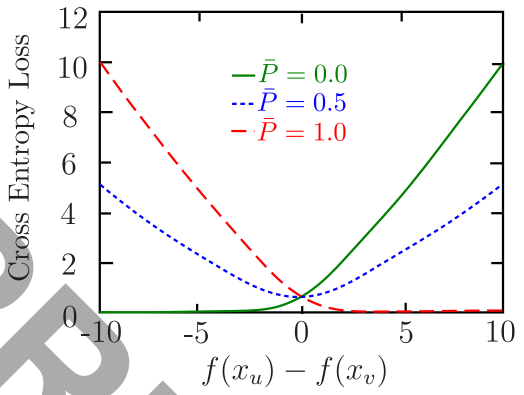

Fig. 3.1 Cross entropy loss as a function of $f(x_u) - f(x_v)$.

retrieval performance.

Tsai, et al. [122] pointed out some problems with the loss function used in RankNet. The curve of the cross entropy loss as a function of $f(x_u) - f(x_v)$ is plotted in Figure 3.1. From this figure, one can see that in some cases, the cross entropy loss has a non-zero minimum, indicating that there will always be some loss no matter what kind of model is used. This may not be in accordance with our intuition of a loss function. Furthermore, the loss is not bounded, which may lead to the dominance of some difficult document pairs in the training process.

To tackle these problems, a new loss function named the *fidelity loss* was proposed [122], which has the following form,

$$L(f; x_u, x_v, y_{u,v}) = 1 - \sqrt{\bar{P}_{u,v} P_{u,v}(f)} - \sqrt{(1 - \bar{P}_{u,v})(1 - P_{u,v}(f))}. \quad (3.5)$$

The fidelity was originally used in quantum physics to measure the difference between two probabilistic states of a quantum. When being used to measure the difference between the target probability and the modeled probability, the fidelity loss has the shape as shown in Figure 3.2 as a function of $f(x_u) - f(x_v)$. By comparing the fidelity loss with the cross entropy loss, we can see that the fidelity loss is bounded between 0 and 1, and always has a zero minimum. These properties

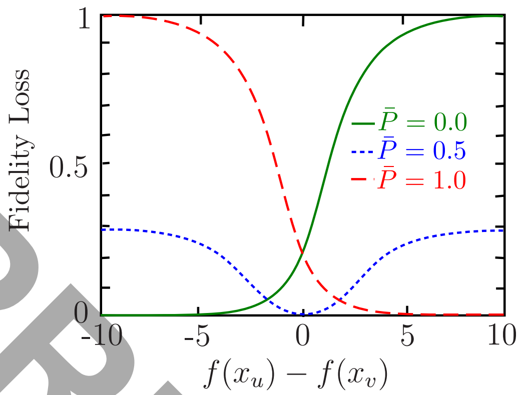

Fig. 3.2 Fidelity loss as a function of $f(x_u) - f(x_v)$.

are nicer than those of the cross entropy loss. On the other hand, however, while the cross-entropy loss is convex, the fidelity loss becomes non-convex. In theory, such a non-convex objective is more difficult to optimize. But overall speaking, according to the experimental results reported in [122], FRank outperforms RankNet on several datasets.

RankBoost

The method of RankBoost [47] adopts AdaBoost [48] for the classification over document pairs. The only difference between RankBoost and AdaBoost is that the distribution in RankBoost is defined on document pairs while that in AdaBoost is defined on individual documents.

The algorithm flow of RankBoost is given in Algorithm 1, where $\mathcal{D}_t$ is the distribution on document pairs, f_t is the weak ranker selected at the t-th iteration, and α_t is the weight for linearly combining the weak rankers. The RankBoost algorithm actually minimizes the exponential loss defined below,

$$L(f; x_u, x_v, y_{u,v}) = \exp(-y_{u,v}(f(x_u) - f(x_v))). \tag{3.6}$$

From Algorithm 1, one can see that RankBoost learns the optimal weak ranker f_t and its coefficient α_t based on the current distribution

Algorithm 1 Learning Algorithm for RankBoost

```
Input: document pairs
Given: initial distribution $\mathcal{D}_1$ on input document pairs.
For $t = 1, \cdots, T$
    Train weak ranker $f_t$ based on distribution $\mathcal{D}_t$.
    Choose $\alpha_t$
    Update $\mathcal{D}_{t+1}(x_u^{(i)}, x_v^{(i)}) = \frac{1}{Z_t}\mathcal{D}_t(x_u^{(i)}, x_v^{(i)}) \exp(\alpha_t(f_t(x_u^{(i)}) - f_t(x_v^{(i)})))$
    where $Z_t = \sum_{i=1}^n \sum_{u,v:y_{u,v}^{(i)}=1} \mathcal{D}_t(x_u^{(i)}, x_v^{(i)}) \exp(\alpha_t(f_t(x_u^{(i)}) - f_t(x_v^{(i)})))$.
Output: $f(x) = \sum_t \alpha_t f_t(x)$.
```

of the document pairs ($\mathcal{D}_t$). Three ways of choosing α_t are discussed in [47].

- First, most generally, for any given weak ranker f_t, it can be shown that Z_t, viewed as a function of α_t, has a unique minimum, which can be found numerically via a simple binary search.
- The second method is applicable in the special case that f_t takes a value from {0,1}. In this case, one can minimize Z_t analytically as follows. For $b \in \{-1, 0, +1\}$, let

$$W_{t,b} = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} \mathcal{D}_t(x_u^{(i)}, x_v^{(i)}) I_{\{f_t(x_u^{(i)}) - f_t(x_v^{(i)}) = b\}}. \quad (3.7)$$

  Then

$$\alpha_t = \frac{1}{2}\log(\frac{W_{t,-1}}{W_{t,+1}}). \quad (3.8)$$

- The third way is based on the approximation of Z_t, which is applicable when f_t takes a real value from [0, 1]. In this case, if we define

$$r_t = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} \mathcal{D}_t(x_u^{(i)}, x_v^{(i)})(f_t(x_u^{(i)}) - f_t(x_v^{(i)})). \quad (3.9)$$

  Then

$$\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t}). \quad (3.10)$$

Because of the analogy to AdaBoost, RankBoost inherits many nice properties from AdaBoost, such as the ability in feature selection, convergence in training, and certain generalization abilities.

Ranking SVM

Ranking SVM [62][68] applies the SVM technology to the task of pairwise classification. Given n training queries $\{q_i\}_{i=1}^n$, their associated document pairs $(x_u^{(i)}, x_v^{(i)})$, and the corresponding ground truth label $y_{u,v}^{(i)}$, the mathematical formulation of Ranking SVM is as shown below, where a linear scoring function is used, i.e., $f(x) = w^T x$,

$$\begin{aligned} &\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\sum_{u,v:y_{u,v}^{(i)}=1} \xi_{u,v}^{(i)} \\ &\text{s.t. } w^T(x_u^{(i)} - x_v^{(i)}) \geq 1 - \xi_{u,v}^{(i)}, \text{if } y_{u,v}^{(i)} = 1, \\ &\qquad \xi_{u,v}^{(i)} \geq 0, i = 1, \cdots, n. \end{aligned} \tag{3.11}$$

As we can see, the objective function in Ranking SVM is exactly the same as in SVM, where the margin term $\frac{1}{2}\|w\|^2$ controls the complexity of the model w. The difference between Ranking SVM and SVM lies in the constraints, which are constructed from document pairs. The loss function in Ranking SVM is a hinge loss defined on document pairs. For example, for a training query q, if document x_u is labeled as being more relevant than document x_v (mathematically, $y_{u,v} = 1$), then if $w^T x_u$ is larger than $w^T x_v$ by a margin of 1, there is no loss. Otherwise, the loss will be $\xi_{u,v}$.

Since Ranking SVM is well rooted in the framework of SVM, it inherits nice properties of SVM. For example, with the help of margin maximization, Ranking SVM can have good generalization ability. Kernel tricks can also be applied to Ranking SVM, so as to handle complex non-linear problems.

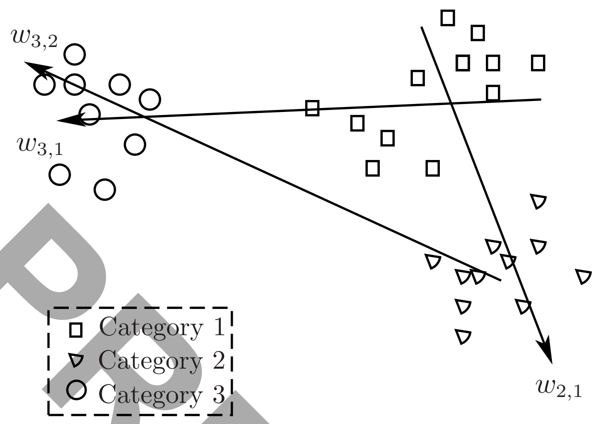

Fig. 3.3 Training Multiple Rankers

3.2 Discussions

3.2.1 Extension of the Pairwise Approach

Note that in the above algorithms, pairwise preference is used as the ground truth label. When we are given the relevance judgment in terms of multiple ordered categories, however, converting it to pairwise preference will lead to the missing of the information about the finer granularity in the relevance judgment.

To tackle the problem, Qin, et al. [97] proposed a new algorithm named the multiple hyperplane ranker (MHR). The basic idea is "divide-and-conquer". Suppose there are K different categories of judgments, then one can train $K(K-1)/2$ Ranking SVM models in total, with each model trained from the document pairs with two specific categories of judgments (see Figure 3.3). At the test phase, rank aggregation is used to merge the ranking results given by each model to produce the final ranking result. For instance, suppose that the model trained from categories k and l is denoted by $f_{k,l}$, then the final ranking results can be attained by using the weighted Borda Count aggregation.

$$f(x) = \sum_{k,l} \alpha_{k,l} f_{k,l}(x). \tag{3.12}$$

Here the combination coefficient $\alpha_{k,l}$ can be pre-specified or learned from a separate validation set. The experimental results in [97] show that by considering more information about the judgment, the ranking performance can be significantly improved over Ranking SVM. Note that the technology used in MHR can actually be extended to any other pairwise ranking algorithm.

3.2.2 Improvement of the Pairwise Approach

It seems that the pairwise approach has its advantages as compared to the pointwise approach, since it can model the relative order between documents. However, in some cases, it faces even larger challenges than the pointwise approach. In Section 2.4, we have mentioned the problem of the pointwise approach when documents are distributed in an imbalanced manner across queries. Here this issue becomes even more serious in the pairwise approach. Considering that every two documents associated with the same query can create a document pair if their relevance degrees are different, in the worse case, the pair number can be quadratic to the document number. As a result, the difference in the numbers of document pairs of different queries is usually significantly larger than the difference in the numbers of documents. This phenomenon has been observed in some previous studies. For example, as reported in [104][99], the distributions of pair numbers per query can be very skewed even if the document numbers of different queries are similar to each other (see Figure 3.4 for the distribution of a dataset from a commercial search engine), indicating that the above problem is really very serious in practice.

In this case, the pairwise loss function will be seriously dominated by the queries with large numbers of document pairs, and as a result the pairwise loss function will become inconsistent with the query-level IR evaluation measures. To tackle the problem, Cao, et al. [16] and Qin, et al. [104][99] proposed introducing query-level normalization to the pairwise loss function. That is, the pairwise loss for a query will be normalized by the total number of document pairs associated with that query. In this way, the normalized pairwise losses with regards to different queries will become comparable with each other in their

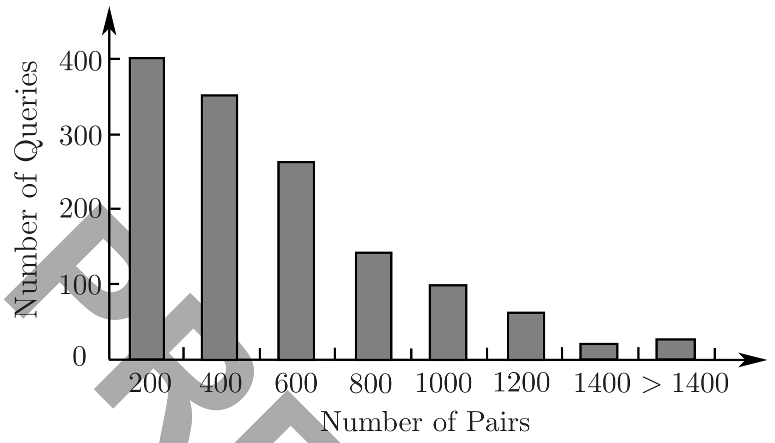

Fig. 3.4 Distribution of Pair Numbers per Query

magnitude, no matter how many document pairs they are originally associated with. With this kind of query-level normalization, Ranking SVM will become a new algorithm, which we call IR-SVM [16]. Specifically, given n training queries $\{q_i\}_{i=1}^n$, their associated document pairs $(x_u^{(i)}, x_v^{(i)})$, and the corresponding relevance judgment $y_{u,v}^{(i)}$, IR-SVM solves the following optimization problem.

$$\begin{aligned} &\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \frac{\sum_{u,v:y_{u,v}^{(i)}=1} \xi_{u,v}^{(i)}}{\tilde{m}^{(i)}} \\ &\text{s.t. } w^T(x_u^{(i)} - x_v^{(i)}) \geq 1 - \xi_{u,v}^{(i)}, \text{if } y_{u,v}^{(i)} = 1 \\ &\qquad \xi_{u,v}^{(i)} \geq 0; i = 1, \cdots, n, \end{aligned} \tag{3.13}$$

where $\tilde{m}^{(i)}$ is the number of document pairs associated with query q_i.

According to the experimental results in [16] [104] [99], a significant performance improvement has been observed after the query-level normalization is introduced.

4

The Listwise Approach

The listwise approach can be divided into two sub-categories. For the first sub-category, the output space contains the relevance degrees of all the documents associated with a query (i.e., $\mathbf{y}$), and the loss function is defined as an approximation or a bound of widely-used IR evaluation measures. For the second sub-category, the output space contains the permutation of the documents associated with the same query (i.e., π_y), and the loss function measures the difference between the permutation given by the hypothesis and the ground truth permutation.

In the following, we will introduce both sub-categories and their representative algorithms.

4.1 Direct Optimization of IR Evaluation Measures

It might be the most straightforward choice to learn the ranking model by directly optimizing what is used to evaluate the ranking performance. This is exactly the motivation of the first sub-category of the listwise approach, which we call the direct optimization methods. However, the task is not as trivial as it seems. As we mentioned before, IR evaluation measures such as NDCG and MAP are position based, and thus non-continuous and non-differentiable [110] [132]. The difficulty of optimizing such objective functions stems from the fact that most existing optimization techniques were developed to handle continuous and differentiable cases.

To tackle the challenges, several attempts have been made. First, one can choose to optimize a continuous and differentiable approximation of the IR evaluation measure. By doing so, many existing optimization technologies can be leveraged. Example algorithms include SoftRank [117] and AppRank [98]. Second, one can alternatively optimize a continuous and differentiable (and sometimes even convex) bound of the IR evaluation measure. Example algorithms include SVM^{map} [136], SVM^{ndcg} [20], and PermuRank [132]. Actually, this trick has also been used in classification[1]. Third, one can choose to use optimization technologies that are able to optimize complex objectives. For example, one can leverage the Boosting framework for this purpose (the corresponding algorithm is called AdaRank [131]), or adopt the genetic algorithm for the optimization (the corresponding algorithm is called RankGP [134]).

In the rest of this section, we will take SoftRank, SVM^{map}, AdaRank, and RankGP for instance to introduce the direct optimization methods.

SoftRank

SoftRank [117] introduces randomness to the ranking scores of the documents, and then uses the expectation of NDCG as an approximation of the original IR evaluation measure NDCG.

First, SoftRank defines the score distribution. Given $\mathbf{x} = \{x_j\}_{j=1}^{m}$ associated with a training query q, the score s_j of document x_j is treated as no longer a deterministic value but a random variable. The random variable is governed by a Gaussian distribution whose variance is σ_s and mean is $f(x_j)$, the original score outputted by the scoring function. That is,

[1] Since the 0-1 classification loss is non-differentiable, convex upper bounds like the exponential loss have been used instead.

$$p(s_j) = N(s_j|f(x_j), \sigma_s^2). \tag{4.1}$$

Second, SoftRank defines the rank distribution. Due to the randomness in the score, every document has the probability of being ranked at any position. Specifically, based on the score distribution, the probability of a document being ranked before another can be deduced as follows.

$$p_{u,v} = \int_0^\infty N(s|f(x_u) - f(x_v), 2\sigma_s^2)ds. \tag{4.2}$$

On this basis, the rank distribution can be derived in an iterative manner. Let us consider adding the documents into the ranked list one after another. Suppose we already have document x_j in the ranked list, when adding document x_u, if document x_u can beat x_j the rank of x_j will be increased by one. Otherwise the rank of x_j will remain unchanged. Mathematically, the probability of x_j being ranked at position r (denoted as $p_j(r)$) can be computed as follows,

$$p_j^u(r) = p_j^{(u-1)}(r-1)p_{u,j} + p_j^{(u-1)}(r)(1 - p_{u,j}). \tag{4.3}$$

Third, with the rank distribution, SoftRank computes the expectation of NDCG@m (where m is the total number of documents associated with the query) as the objective function for learning to rank (which we call SoftNDCG[2]). In other words, (1−SoftNDCG) corresponds to the loss function in SoftRank.

$$\text{SoftNDCG} \triangleq \frac{1}{Z_m}\sum_{j=1}^{m}(2^{y_j} - 1)\sum_{r=0}^{m-1}\eta(r)p_j(r). \tag{4.4}$$

In order to learn the ranking model f by maximizing SoftNDCG, one can use a neural network as the model, and gradient descent as the

[2] For ease of reference, we also refer to the objective functions like SoftNDCG as the surrogate measure.

optimization algorithm. In [55], the Gaussian process is used to further enhance the SoftRank algorithm, where σ_s is no longer a pre-specified constant but a learned parameter.

SVM^{map}

SVM^{map} [136] uses the framework of structured SVM [123] [70] to optimize the IR evaluation measure Average Precision (AP).

Suppose $\mathbf{x} = \{x_j\}_{j=1}^m$ represents all the documents associated with training query q, its corresponding ground truth label is $\mathbf{y} = \{y_j\}_{j=1}^m$ ($y_j = 1$, if document x_j is labeled as relevant; $y_j = 0$, otherwise), and any incorrect label of $\mathbf{x}$ is represented as $\mathbf{y}^c$. Then SVM^{map} is formulated as follows, where AP is used in the constraints of structured SVM. It has been proved that the sum of slacks in SVM^{map} can bound (1−AP) from above.

$$\begin{aligned} &\min \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi^{(i)} \\ &\text{s.t. } \forall \mathbf{y}^{c(i)} \neq \mathbf{y}^{(i)}, \\ &\quad w^T\Psi(\mathbf{y}^{(i)},\mathbf{x}^{(i)}) \geq w^T\Psi(\mathbf{y}^{c(i)},\mathbf{x}^{(i)}) + 1 - \mathrm{AP}(\mathbf{y}^{c(i)}) - \xi^{(i)}. \end{aligned} \tag{4.5}$$

Here Ψ is called the joint feature map, which definition is given as below.

$$\Psi(\mathbf{y},\mathbf{x}) = \sum_{u,v:y_u=1,y_v=0}(x_u - x_v). \tag{4.6}$$

$$\Psi(\mathbf{y}^c,\mathbf{x}) = \sum_{u,v:y_u=1,y_v=0}(y_u^c - y_v^c)(x_u - x_v). \tag{4.7}$$

As we know, there are an exponential number of incorrect labels for the documents, and thus the optimization problem has an exponential number of constraints for each query. Therefore, it is a big challenge to directly solve such an optimization problem. To tackle the challenge, a working set is maintained, which only contains those constraints with the largest violation (defined below), and the optimization is performed only with respect to this working set.

$$\text{Violation} \triangleq 1 - \text{AP}(\mathbf{y}^c) + w^T\Psi(\mathbf{y}^c, \mathbf{x}). \tag{4.8}$$

Then the problem turns out to efficiently find the most violated constraints for a given scoring function $f(x) = w^T x$. To this end, the property of AP is considered. That is, if the relevance at each position is fixed, AP will be the same no matter which document appears at that position. Furthermore, with the same AP, if the documents are sorted according to the descending order of their scores, $w^T\Psi(\mathbf{y}^c, \mathbf{x})$ will be maximized. Therefore, an efficient strategy to find the most violated constraint can be designed [136], which time complexity is $O(m \log m)$, where m is the number of documents associated with query q.

In [20] [21], the idea of SVM^{map} is further extended to optimize other IR evaluation measures, and the corresponding algorithms are named as SVM^{ndcg} and SVM^{mrr}. Basically, different feature maps or different strategies of searching the most violated constraints are used in these extensions, but the idea remains the same as that of SVM^{map}.

AdaRank

Xu and Li [131] found that IR evaluation measures can be plugged into the framework of Boosting and get effectively optimized. This process does not require the IR evaluation measures to be continuous and differentiable. The resultant algorithm is called AdaRank.

As we know, in the conventional AdaBoost algorithm, the exponential loss is used to update the distribution of input objects and to determine the combination coefficient α_t at each round of iteration. Analogously, in AdaRank, IR evaluation measures are used to update the distribution of queries and to compute the combination coefficient. The algorithm flow is shown below, where $M(f, \mathbf{x}, \mathbf{y})$ represents the IR evaluation measure.

Due to the analogy to AdaBoost, AdaRank can focus on those hard queries. Furthermore, a condition for the convergence of the training process was given in [131], with a similar derivation technique to that for AdaBoost. The condition requires $|M(\sum_{s=1}^{t} \alpha_s f_s, \mathbf{x}, \mathbf{y}) - M(\sum_{s=1}^{t-1} \alpha_s f_s, \mathbf{x}, \mathbf{y}) - \alpha_t M(f_t, \mathbf{x}, \mathbf{y})|$ to be very small. This implies the

Algorithm 2 Learning Algorithms for AdaRank

```
Input: document group for each query
Given: initial distribution $\mathcal{D}_1$ on input queries
For $t = 1, \cdots, T$
    Train weak ranker $f_t(\cdot)$ based on distribution $\mathcal{D}_t$.
    Choose $\alpha_t = \frac{1}{2} \log \frac{\sum_{i=1}^{n} \mathcal{D}_t(i)(1+M(f_t,\mathbf{x}^{(i)},\mathbf{y}^{(i)}))}{\sum_{i=1}^{n} \mathcal{D}_t(i)(1-M(f_t,\mathbf{x}^{(i)},\mathbf{y}^{(i)}))}$
    Update $\mathcal{D}_{t+1}(i) = \frac{\exp(-M(\sum_{s=1}^{t} \alpha_s f_s,\mathbf{x}^{(i)},\mathbf{y}^{(i)}))}{\sum_{j=1}^{n} \exp(-M(\sum_{s=1}^{t} \alpha_s f_s,\mathbf{x}^{(j)},\mathbf{y}^{(j)}))}$,
Output: $\sum_t \alpha_t f_t(\cdot)$.
```

assumption on the linearity of the IR evaluation measure M, as a function of f_t. However, this assumption may not be well satisfied in practice. Therefore, it is possible that the training process of AdaRank does not always naturally converge and some additional stopping criteria are needed.

Genetic Programming based Algorithms

There are some methods originally designed for optimizing complex objectives. Genetic programming is just one of such methods. In the literature of learning to rank, there have been several attempts on using genetic programming to optimize IR evaluation measures. Representative algorithms include [134] [4] [41] [42] [43] [46] [44] [121] [45].

Here we take the algorithm named RankGP [134] as an example to illustrate how genetic programming can be used to learn the ranking model. In this algorithm, the ranking model is defined as a tree, whose leaf nodes are features or constants, and non-leaf nodes are operators such as $+, -, \times, \div$ (see Figure 4.1). Then single population genetic programming is used to perform the learning on the tree. Cross-over, mutation, reproduction, and tournament selection are used as evolution mechanisms, and the IR evaluation measure is used as the fitness function.

In addition to the examples introduced above, there are also some other works [86] [110] [13] that directly optimize IR evaluation mea-

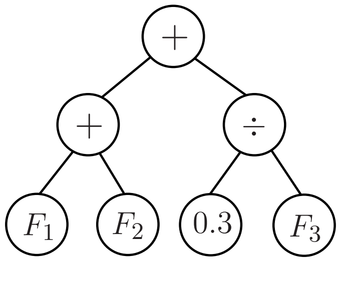

Fig. 4.1 Ranking function used in RankGP

sures. Due to space restrictions, we will not introduce them in detail.

4.2 Minimization of Listwise Ranking Losses

In the second sub-category of the listwise approach, the loss function measures the inconsistency between the output of the ranking model and the ground truth permutation π_y. Although IR evaluation measures are not directly optimized here, if one can consider the distinct properties of ranking in IR in the design of the loss function, it is also possible that the model learned can have good performance in terms of IR evaluation measures. We refer to these algorithms as "algorithms that minimize listwise ranking losses". In this section, we will introduce two representative algorithms of them, i.e., ListNet [17] and ListMLE [129].

ListNet

In [17], a listwise ranking loss is proposed, which is based on the probability distribution on permutations.

Actually the probability distributions on permutations have been well studied in the field of probability theory. Many famous models have been proposed to represent permutation probability distributions, such as the Luce model [81] [95] and the Mallows model [82]. Since a permutation has a natural one-to-one correspondence with a ranked list, these researches can be naturally applied to ranking. ListNet [17] is just such an example, demonstrating how to apply the Luce model to learning to rank.

Given the relevance scores of the documents outputted by the scoring function f (i.e., $\mathbf{s} = \{s_j\}_{j=1}^m$, where $s_j = f(x_j)$), the Luce model defines a probability for each possible permutation π of the documents, based on the chain rule, as follows,

$$P(\pi|\mathbf{s}) = \prod_{j=1}^{m} \frac{\varphi(s_{\pi^{-1}(j)})}{\sum_{u=j}^{m} \varphi(s_{-1\pi(u)})}, \tag{4.9}$$

where $\pi^{-1}(j)$ denotes the document ranked at the j-th position of permutation π, φ is a transformation function, which can be linear, exponential, or sigmoid. Each item $\frac{\varphi(s_{\pi^{-1}(j)})}{\sum_{u=j}^{m} \varphi(s_{\pi^{-1}(u)})}$ is a conditional probability as shown in the following example.

Suppose we have in total three documents A, B and C associated with query q. The probability of permutation $\pi = (A, B, C)$ is equal to the product of the following three probabilities (i.e., $P_\pi = P_1 P_2 P_3$).

- P_1: the probability of document A being ranked at the top one position in π. This probability is computed by comparing the score of A with those of all the other documents.

$$P_1 = \frac{\varphi(s_A)}{\varphi(s_A) + \varphi(s_B) + \varphi(s_C)}. \tag{4.10}$$

- P_2: the conditional probability of document B being ranked at the second position given that A has already been ranked at the first position. This probability is computed by comparing the score of B with those of the other documents except A. In this simple example, there is only document C left to be compared with.

$$P_2 = \frac{\varphi(s_B)}{\varphi(s_B) + \varphi(s_C)}. \tag{4.11}$$

- P_3: the conditional probability of document C being ranked on the third position given that documents A and B have already been ranked in the top two positions respectively. In our simple case, it is easy to see $P_3 = 1$.

With the Luce model, for a given query q, ListNet first defines the permutation probability distribution based on the scores given by the scoring function f. Then it defines another permutation probability distribution $P_y(\pi)$ based on the ground truth label[3]. For the next step, ListNet uses the K-L divergence between these two distributions to define its listwise ranking loss (which we call the *K-L divergence loss* for short).

$$L(f;\mathbf{x},\pi_y) = D(P(\pi|\varphi(f(w,\mathbf{x})))\|P_y(\pi)). \tag{4.12}$$

A neural network model is employed in ListNet, and the gradient descent approach is used to minimize the K-L divergence loss. As shown in [17], the training curve of ListNet well demonstrates the correlation between the K-L divergence loss and $1 - \text{NDCG@5}$.

As one may have noticed, there is a computational issue with the ListNet algorithm. Although due to the use of the scoring function to define the hypothesis, the testing complexity of ListNet can be the same as those of the pointwise and pairwise approaches, the training complexity of ListNet is much higher. The training complexity of ListNet is in the exponential order of m (and thus intractable in practice), because the evaluation of the K-L divergence loss for each query q requires the addition of m-factorial terms (with a complexity of $O(m!)$). Comparatively speaking, the training complexities of the pointwise and pairwise approaches are roughly proportional to the number of documents (i.e., $O(m)$) and the number of document pairs (i.e., $O(\tilde{m})$). To tackle the problem, a top-k version of the K-L divergence loss is further introduced in [17], which is based on the top-k Luce model and can reduce the training complexity from m-factorial to the polynomial order of m.

[3] For example, if the ground truth is given in terms of relevance degree, it can be directly substituted into the Luce model to define a probability distribution. If the ground truth label is given as a ranked list, one can simply define the probability distribution as a delta function, or use a mapping function to map it to real-valued scores of the documents and then apply the Luce model. One can also use other ways such as the Mallows model to define the permutation probability distribution of the ground truth label.

ListMLE

Even if the top-k K-L divergence loss is used in ListNet, one still cannot avoid its following limitation. When k is set to be large, the time complexity of evaluating the K-L divergence loss is still very high. However, when k is set to be small, information about the permutation will be significantly lost and the effectiveness of the ListNet algorithm is questionable [17].

To tackle these problems, a new algorithm named ListMLE was proposed [129]. ListMLE is also based on the Luce model. For each query q, with the permutation probability distribution defined with the output of the scoring function, it uses the negative log likelihood of the ground truth permutation as the listwise ranking loss. We denote this new listwise ranking loss as the *likelihood loss* for short.

$$L(f; \mathbf{x}, \pi_y) = -\log P(\pi_y | \varphi(f(w, \mathbf{x}))). \tag{4.13}$$

It is clear, in this way the training complexity can be greatly reduced as compared to ListNet, since one only needs to compute the probability of a single permutation π_y but not all the permutations. Once again, one can use a neural network model to optimize the likelihood loss.

It should be noted that the likelihood loss has assumed the ground truth labels to be given as the total order of the documents. This is an advantage when it is really the case, however, if the judgments are given in other terms, the ListMLE algorithm will not work. To tackle this challenge and further enhance the application scope of ListMLE, one needs to conduct the following preprocessing of the training data.

Given the judgment in terms of relevance degree, one can define an *equivalent permutation set* as follows.

$$\Omega_y = \{\pi_y | u < v, \text{ if } l_{\pi_y^{-1}(u)} \succ l_{\pi_y^{-1}(v)}\}.$$

Similarly, given the judgment in terms of pairwise preferences, one can define Ω_y as below.

$$\Omega_y = \{\pi_y | u < v, \text{ if } l_{\pi_y^{-1}(u), \pi_y^{-1}(v)} = 1\}.$$

As compared to the judgment in terms of total order, we can regard the above judgments as incomplete. In other words, they are the necessary conditions of being ground truth permutations. However, permutations satisfying these constraints might not always be the ground truth permutations. This situation is very similar to that in multi-instance learning. Therefore, one can define the loss function, following a similar idea proposed for multi-instance learning [6], as below.

$$L(f; \mathbf{x}, \Omega_y) = \min_{\pi_y \in \Omega_y} \left(-\log P(\pi_y | \varphi(f(w, \mathbf{x})))\right). \tag{4.14}$$

In [129], some analyses have been made on ListNet and ListMLE. Basically, it has been proved that both the K-L divergence loss and the likelihood loss are continuous, differentiable, and convex. In this way, they can be easily optimized using existing methods, such as the gradient descent algorithm.

4.3 Discussions

As shown in this chapter, different kinds of listwise ranking algorithms have been proposed. Intuitively speaking, they model the ranking problem in a more natural way than the pointwise and pairwise approaches, and thus can address some problems that these two approaches have encountered. As we have discussed in the previous chapters, for the pointwise and pairwise approaches, the position information is invisible to their loss functions, and they ignore the fact that some documents (or document pairs) are associated with the same query. Comparatively speaking, the listwise approach takes all the documents associated with the same query as the input instance and their ranked list (or their relevance degrees) as the output. In this way, it has the potential to distinguish documents from different queries, and to consider the position information in the output ranked list in its learning process. According to some previous studies, the performances of the listwise ranking algorithms are really better than previous approaches [129]. This is also verified by the discussions in Chapter 6, which is about the empirical ranking performances of different learning-to-rank algorithms, with the LETOR benchmark dataset as the experimental platform.

On the other hand, the listwise approach also has certain aspects to be improved. For example, the training complexities of some listwise ranking algorithms (e.g., ListNet) are high since the evaluation of their loss functions are permutation based. More efficient learning algorithm is needed to make the listwise approach more practical. Moreover, we would like to point out that the use of the position information in some listwise ranking algorithms is insufficient. For example, there is no explicit position discount considered in the loss functions of ListNet and ListMLE. As a result, even if the algorithms can see different positions in the output ranked list, they have not really distinguished them in the learning process. By introducing certain position discount factors, the performance improvement of these algorithms can be expected.

5

Analysis of the Approaches

In the previous three chapters, we have introduced the pointwise, pairwise, and listwise approaches to learning to rank. The major differences between these approaches are the loss functions. Note that the loss functions are mainly used to guide the learning process, while the evaluation of the learned ranking model is based on IR evaluation measures. Therefore, an important issue to discuss is the relationship between the loss functions used in these approaches and the IR evaluation measure. This is exactly the motivation of this chapter. Without loss of generality, we will take NDCG@m as an example in the discussions. Here m is the total number of documents associated with the query. For simplicity, we refer to NDCG@m as NDCG.

For ease of our discussion, we assume that the judgment is given in terms of multiple ordered categories, and the ground truth label is represented by a permutation set Ω_y as defined in Eqn.(4.14)(in other cases, one can obtain similar results). With this assumption, NDCG with respect to a given ranked list π can be defined as follows[1].

$$\mathrm{NDCG}(\pi, \Omega_y) = \frac{1}{Z_m} \sum_{t=1}^{m} G(\pi_y^{-1}(t))\eta(\pi(\pi_y^{-1}(t)), \forall \pi_y \in \Omega_y. \tag{5.1}$$

It is easy to verify that for $\forall \pi_y \in \Omega_y$, the right-hand side of the equation takes the same value. Sometimes, we need to emphasize the ranking model, i.e., when $\pi = \mathrm{sort} \circ f$. In this case, we will denote the above defined NDCG as $\mathrm{NDCG}(f, \mathbf{x}, \Omega_y)$.

5.1 The Pointwise Approach

As mentioned in Chapter 2, Cossock and Zhang [33] have established the theoretical foundation for reducing ranking to regression[2]. Given $\mathbf{x} = \{x_j\}_{j=1}^m$, a set of documents associated with training query q, and the ground truth $\mathbf{y} = \{y_j\}_{j=1}^m$ of these documents in terms of multiple ordered categories, suppose a scoring function f is used to rank these documents. The authors proved a theory showing that the ranking error in terms of NDCG can be bounded by the following regression loss.

$$1 - \mathrm{NDCG}(f, \mathbf{x}, \Omega_y) \leq \frac{1}{Z_m} \left(2 \sum_{j=1}^{m} \eta(j)^2\right)^{\frac{1}{2}} \left(\sum_{j=1}^{m} (f(x_j) - y_j)^2\right)^{\frac{1}{2}}, \tag{5.2}$$

where Z_m is the maximum DCG value and $\eta(j)$ is the discount factor used in NDCG.

In other words, if one can really minimize the regression loss to zero, one can also minimize (1−NDCG) to zero. This seems to be a very nice property of the regression based methods.

With similar proof techniques to those used in [33], Li, et al. [78] showed that (1−NDCG) can also be bounded by the multi-class classification loss as shown below (it is assumed K=5 in the inequality).

[1] Note that this definition of NDCG is equivalent to that given in Eqn.(1.9), although the index of the summation changes from the rank position in π (i.e., r) to the position in the ground truth list (i.e., t).

[2] Note that the bounds given in the original papers are with respect to DCG, and here we give their equivalent form in terms of NDCG for ease of comparison.

$$1 - \mathrm{NDCG}(f, \mathbf{x}, \Omega_y) \leq \frac{15}{Z_m}\sqrt{2\left(\sum_{j=1}^{m}\eta(j)^2 - m\prod_{j=1}^{m}\eta(j)^{\frac{2}{m}}\right)} \cdot \sqrt{\sum_{j=1}^{m} I_{\{y_j \neq \hat{y}_j\}}}, \tag{5.3}$$

where $\hat{y}_j$ is the prediction on the label of x_j by the multi-class classifier, and $f(x_j) = \sum_{k=0}^{K-1} k \cdot P(\hat{y}_j = k)$.

In other words, if one can really minimize the classification loss to zero, one can also minimize (1−NDCG) to zero at the same time.

However, on the other hand, please note that when (1−NDCG) is zero (i.e., the documents are perfectly ranked), the regression loss and the classification loss might not be zero (and can still be very large). In other words, the minimization of the regression loss and the classification loss is only a sufficient condition but not a necessary condition for optimal ranking in terms of NDCG.

Let us have a close look at the classification bound in (5.3) with an example[3]. Note that a similar example has been given in [1] to show the problem of reducing ranking to binary classification.

Suppose for a particular query q, we have four documents (i.e., $m = 4$) in total, and their ground truth labels are 4, 3, 2, and 1 respectively (i.e., $y_1 = 4$, $y_2 = 3$, $y_3 = 2$, $y_4 = 1$). We use the same discount factor and gain function as used in [78]. Then it is easy to compute that $Z_m = \sum_{j=1}^{4} \frac{1}{log(j+1)}(2^{y_j - 1}) \approx 21.35$.

Then, suppose the outputs of the multi-class classifier are $\hat{y}_1 = 3$, $\hat{y}_2 = 2$, $\hat{y}_3 = 1$, and $\hat{y}_4 = 0$, with 100% confidence in the prediction for each document. It is easy to compute that $1 - \mathrm{NDCG}(f, \mathbf{x}, \Omega_y)$ is 0 and we actually get a perfect ranking based on the classifier. However, in terms of multi-class classification, we made errors in all the four documents, i.e., $\sum_{j=1}^{m} I_{\{y_j \neq \hat{y}_j\}} = 4$. Furthermore, if we compute the bound given by formula (5.3), we obtain

[3] One can get similar results for the regression bound given in inequality (5.2).

$$\frac{15}{Z_m}\sqrt{2\left(\sum_{j=1}^{m}(\frac{1}{log(j+1)})^2 - m\prod_{j=1}^{m}(\frac{1}{log(j+1)})^{\frac{2}{m}}\right)\cdot\sum_{j=1}^{m} I_{\{y_j\neq\hat{y}_j\}}}$$
$$\approx \frac{24.49}{21.35} = 1.15.$$

It is clear the bound is not meaningful since it is even large than one. Actually the loose bound is not difficult to understand. The left-hand side of inequality (5.3) contains the position information, while the right-hand side does not. When the same amount of classification loss occurs in different positions, the ranking error will be quite different. In order to make the inequality always hold, the price one has to pay is that the bound must be very loose.

5.2 The Pairwise Approach

It has been shown in [22] that many of the pairwise loss functions are margin-based upper bounds of a quantity, named the essential loss for ranking. Furthermore, the essential loss is an upper bound of (1−NDCG), and therefore these loss functions are also upper bounds of (1−NDCG).

To better illustrate this result, we first introduce the concept of the essential loss, which is constructed by modeling ranking as a sequence of classifications.

Given a set of documents $\mathbf{x}$ and their ground truth permutation $\pi_y \in \Omega_y$, the ranking problem can be decomposed into several sequential steps. For each step t, one tries to distinguish $\pi_y^{-1}(t)$, the document ranked at the t-th position in π_y, from all the documents ranked below the t-th position in π_y, using a scoring function f. Denote $\mathbf{x}_{(t)} = \{x_{\pi_y^{-1}(t)}, \cdots, x_{\pi_y^{-1}(m)}\}$, one can define a classifier based on f, whose target output is $\pi_y^{-1}(t)$,

$$T \circ f(\mathbf{x}_{(t)}) = \arg\max_{j=\pi_y^{-1}(t),\cdots,\pi_y^{-1}(m)} f(x_j). \tag{5.4}$$

It is clear that there are $m-t$ possible outputs of this classifier, i.e., $\{\pi_y^{-1}(t), \cdots, \pi_y^{-1}(m)\}$. The 0-1 loss for this classification task can be written as follows, where the second equation is based on the definition of $T \circ f$,

$$L_t(f; \mathbf{x}(t), \pi_y^{-1}(t)) = I_{\{T \circ f(\mathbf{x}_{(t)}) \neq \pi_y^{-1}(t)\}} \\ = 1 - \prod_{j=t+1}^{m} I_{\{f(\pi_y^{-1}(t)) > f(\pi_y^{-1}(j))\}}. \tag{5.5}$$

By summing up the losses at all the steps ($t = 1, \cdots, m-1$), one can obtain,

$$\tilde{L}(f; \mathbf{x}, \pi_y) = \sum_{t=1}^{m-1} I_{\{T \circ f(\mathbf{x}_{(t)}) \neq \pi_y^{-1}(t)\}}. \tag{5.6}$$

By further generalizing the indicator function $I_{\{\cdot\}}$ to the label-based cost-sensitive function $a(\cdot)$ ($a(i,j) = 0$ if $l_i = l_j$ and $a(i,j) = 1$ otherwise), we will get the so-called essential loss[4], which takes the same value for any permutation in the permutation set Ω_y.

$$\tilde{L}(f; \mathbf{x}, \Omega_y) = \sum_{t=1}^{m-1} a\left(\pi_y^{-1}(t), T \circ f(x_{(t)})\right), \forall \pi_y \in \Omega_y. \tag{5.7}$$

It has been proved in [22] that the essential loss is an upper bound of (1−NDCG). As a result, the minimization of it will lead to the effective maximization of NDCG:

$$1 - \text{NDCG}(f, \mathbf{x}, \Omega_y) \leq \frac{2^{K-1} - 1}{Z_m} \left(\sum_{t=1}^{m-1} \eta(t)^{\alpha} \right)^{\frac{1}{\alpha}} \left(\tilde{L}(f; \mathbf{x}, \Omega_y) \right)^{\frac{1}{\beta}}, \tag{5.8}$$

where $\frac{1}{\alpha} + \frac{1}{\beta} = 1$.

As compared to the bounds given in the previous section, one can see that the essential loss has a nicer property. When (1−NDCG) is zero, the essential loss is also zero. In other words, the zero value of the essential loss is not only a sufficient condition but also a necessary condition of the zero value of (1−NDCG).

[4] In other words, here one does not require the classifier at each step to output exactly the document $\pi_y^{-1}(t)$, but only requires the label of output is the same as that of $\pi_y^{-1}(t)$.

Furthermore, it has been proved that the widely used pairwise loss functions are actually *Cost-sensitive Pairwise Comparison* (CPC) surrogate functions [142] of the essential loss, which take the following form [22],

$$\begin{aligned} & L_{\phi}(f;\mathbf{x},\Omega_y) \\ & = \sum_{t=1}^{m-1}\sum_{j=t+1}^{n} a(\pi_y^{-1}(j),\pi_y^{-1}(t))\phi\left(f(x_{\pi_y^{-1}(t)}) - f(x_{\pi_y^{-1}(j)})\right) \\ & = \sum_{t=1}^{m-1} \sum_{\substack{j=t+1,\\ l_{\pi_y^{-1}(t)} \neq l_{\pi_y^{-1}(j)}}}^{n} \phi\left(f(x_{\pi_y^{-1}(t)}) - f(x_{\pi_y^{-1}(j)})\right), \forall \pi_y \in \Omega_y. \end{aligned} \tag{5.9}$$

When function ϕ is the hinge function, the exponential function, and the logistic function, the above surrogate loss will become exactly the loss functions of Ranking SVM, RankBoost, and RankNet. According to the properties of margin-based surrogate losses [142], the CPC surrogate loss is an upper bound of the essential loss.

$$\tilde{L}(f;\mathbf{x},\Omega_y) \leq L_{\phi}(f;\mathbf{x},\Omega_y). \tag{5.10}$$

Therefore, the minimization of the loss functions in the aforementioned pairwise ranking algorithms will all lead to the minimization of the essential loss. Further considering the relationship between the essential loss and (1−NDCG), these algorithms can also effectively minimize (1−NDCG).

5.3 The Listwise Approach

5.3.1 Listwise Ranking Loss

One sub-category of the listwise approach minimizes a listwise ranking loss in the training process. Here we take ListMLE as an example to perform the discussion on this sub-category. Actually, the technique used in the discussions on the pairwise loss functions can be used again.

First of all, it has been proved in [22] that the essential loss given in Eqn.(5.7) has the following equivalent form.

$$\tilde{L}(f;\mathbf{x},\Omega_y) = \min_{\pi_y \in \Omega_y} \tilde{L}(f;\mathbf{x},\pi_y). \tag{5.11}$$

Second, one needs to introduce another kind of margin-based surrogate function, called the *Unconstrained Background Discriminative* (UBD) surrogate function [142]. With this surrogate function, one can obtain the following surrogate loss of the essential loss, where ϕ is a decreasing function, α and γ are increasing functions.

$$\begin{aligned} &L_\phi(f;\mathbf{x},\Omega_y) \\ &= \min_{\pi_y \in \Omega_y} \sum_{t=1}^{m-1} \left(\phi(f(x_{\pi_y^{-1}(t)})) + \alpha(\sum_{j=t}^{m} \gamma(f(x_{\pi_y^{-1}(j)}))) \right). \end{aligned} \tag{5.12}$$

It has been proved in [142][22] that in certain conditions, the UBD surrogate loss can also bound the essential loss from above. In particular, by setting $\phi(x) = -x, \alpha(x) = \log x$, and $\gamma(x) = e^x$ in the UBD surrogate loss, one can get the following specific form of it,

$$\begin{aligned} L_\phi(f;\mathbf{x},\Omega_y) &= \min_{\pi_y \in \Omega_y} \sum_{t=1}^{m-1} \left(f(x_{\pi_y^{-1}(t)}) + \log \left(\sum_{j=t}^{m} \exp(f(x_{\pi_y^{-1}(j)})) \right) \right) \\ &\geq \frac{1}{\ln 2} \tilde{L}(f;\mathbf{x},\Omega_y). \end{aligned} \tag{5.13}$$

It is clear that this is exactly the likelihood loss used in ListMLE (see Eqn.(4.14) when the exponential function is used as the transformation function φ). In other words, the loss function of ListMLE is a specific UBD surrogate loss, and is an upper bound of the essential loss. Recalling the connection between the essential loss and (1−NDCG) as discussed in the previous section, the likelihood loss can also upper bound (1−NDCG). Therefore, the minimization of the likelihood loss in the training process will lead to the minimization of (1−NDCG) [22].

5.3.2 Loss Functions in Direct Optimization Methods

The other sub-category of the listwise approach optimizes a loss function derived from the IR evaluation measure. It seems that the discussion on the relationship between such a loss function and the corresponding IR evaluation measure is more straightforward, since they have natural connections. However, in order to make a formal discussion on the issue, a new quantity named "directness" needs to be introduced [61]. Basically, directness measures whether a surrogate measure is a good approximation of the corresponding IR evaluation measure. Note that in the following definition, we assume the use of a linear scoring function, i.e., $f(x) = w^T\mathbf{x}$.

Definition 5.1. (Directness) For any query q, suppose its associated documents and the ground truth are $\mathbf{x}$ and Ω_y respectively. For a ranking model w, denote $\tilde{M}(w, \mathbf{x}, \Omega_y)$ and $M(w, \mathbf{x}, \Omega_y)$ as a surrogate measure and its corresponding IR evaluation measure respectively. The directness of $\tilde{M}$ with respect to M is defined as

$$D(w, \tilde{M}, M) = \frac{1}{\sup_{\mathbf{x},\Omega_y} |M(w, \mathbf{x}, \Omega_y) - \tilde{M}(w, \mathbf{x}, \Omega_y)|}. \tag{5.14}$$

As can be seen, the directness is determined by the maximum difference between the surrogate measure and its corresponding IR evaluation measure with respect to a ranking model w, over the entire input and output spaces. In the extreme case, when the difference becomes zero, the directness will become infinite, and the surrogate measure will become exactly the IR evaluation measure.

As examples, the directness of SoftRank and SVM^{map} have been analyzed in [61]. The corresponding result for SoftRank is listed in the following theorem.

Theorem 5.1. For query q, suppose its associated documents and ground truth labels are $\mathbf{x}$ and Ω_y respectively. Assume $\forall i$ and $j, |f(x_i) - f(x_j)| \geq \delta > 0$ and $\forall q, m \leq M$. If $\sigma_s < \frac{\delta}{2\text{erf}^{-1}\left(\sqrt{\frac{5M-9}{5M-5}}\right)}$, then:

$$D(w, \text{SoftNDCG}, \text{NDCG}) \geq \frac{1}{M \cdot 2^{K-1} \cdot (\varepsilon_1 + \varepsilon_2)}, \tag{5.15}$$

where

$$\varepsilon_1 = \frac{(M-1)\sigma_s}{2\delta\sqrt{\pi}} e^{-\frac{\delta^2}{4\sigma_s^2}}, \quad \varepsilon_2 = \sqrt{\frac{\varepsilon_3(\sigma_s)}{1-5\varepsilon_3(\sigma_s)}} + 5\varepsilon_3(\sigma_s),$$

$$\varepsilon_3 = \frac{M-1}{4}\left[1 - \operatorname{erf}^2\left(\frac{\delta}{2\sigma_s}\right)\right], \quad \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt,$$

and K is the number of relevance degrees in the judgment.

From the above theorem, we can see that the lower bound of the directness of SoftNDCG is a decreasing function of σ_s. Furthermore, when $\sigma_s \to 0$, the directness of SoftNDCG becomes extremely large. The intuitive explanation of this result is as follows. When the variance of the score distribution is small enough[5], it is highly probable that the introduction of the score distribution will not change the original order of any two documents. Therefore, the value of SoftNDCG can be very close to that of NDCG, even if it has become continuous and differentiable.

Similar analysis on SVM^{map}, however, shows that for any ranking model, there always exists such inputs and outputs that will result in the large difference between its surrogate measure and the corresponding IR evaluation measure [61]. Consequently, it is not guaranteed that these algorithms can lead to the effective optimization of the IR evaluation measures[6].

5.4 Discussions

In this chapter, we have reviewed the relationships between different loss functions in learning to rank and the IR evaluation measure. The discussions well explain why different learning-to-rank algorithms perform reasonably well in practice (see the experimental results in Chapter 6).

While the analyses introduced in this chapter look quite nice, there are still several issues that have not been well solved. First, although

[5] Since σ_s is a pre-defined parameter in the SoftRank algorithm, we are able to make SoftRank as direct as desired by setting σ_s as small as possible regardless the joint probability distribution of the inputs and outputs.

[6] Note that the same conclusion also applies to SVM^{ndcg} [20] and PermuRank [132].

the essential loss can be used to explain the relationship between (1−NDCG) and the loss functions in the pairwise and listwise approaches, it is not clear whether it can also be used to explain the pointwise approach. If it is the case, the essential loss will become really "essential". Second, from the machine learning point of view, being an upper bound of the evaluation measure might not be sufficient for a good loss function. The reason is that what we really care about is the optimal solution with regards to the loss function. Even if a loss can be the upper bound of (1−NDCG) everywhere, its optimum might not correspond to the optimum of (1−NDCG).

The discussions on the "directness" is one step toward solving this problem. A more principled solution should be obtained by investigating the so-called "consistency" of the loss functions, which exactly describes whether the optima with regards to the loss function and the measure can be the same. Consistency of learning methods have been well studied in classification, but not yet for ranking (see discussions in Chapter 7). This should be important future work of learning to rank, from the theoretical point of view.

Another thing to mention is that the discussions on either "directness" or "consistency" are valid only when the number of training examples approaches infinity. However, in practice, the number of training examples is always limited. In this case, the theoretical analysis might not be always in accordance with experimental results. For example, although the directness analysis on SoftRank shows that its surrogate loss SoftNDCG is very direct to NDCG, the empirical results reported in [117] have not demonstrated a significant improvement over previous methods. Therefore, a lot of work needs to be further done in order to predict the performance of a learning-to-rank method when the sample size is small.

6

Benchmarking Learning-to-Rank Algorithms

In this chapter, we introduce a benchmark dataset for learning to rank and investigate the empirical performance of some representative learning-to-rank algorithms on the dataset.

As we know, a standard dataset with standard features and evaluation measures is very helpful for the research on machine learning. For example, there are benchmark datasets such as Reuters[1] and RCV-1[2] for text classification, and UCI[3] for general machine learning. However, there were no such benchmark datasets for ranking until the LETOR collection [79] was released in early 2007. In recent years, the LETOR collection has been widely used in the experiments of learning-to-rank papers, and have helped to greatly move forward the research on learning to rank.

[1] http://www.daviddlewis.com/resources/testcollections/reuters21578/
[2] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm
[3] http://archive.ics.uci.edu/ml/

6.1 The LETOR Collection

In this section, we introduce the LETOR collection, including five aspects: document corpora (together with query sets), document sampling, feature extraction, meta information, and cross validation.

6.1.1 Document Corpora

Two document corpora together with seven query sets were used in the LETOR collection.

6.1.1.1 The "Gov" Corpus and Six Query Sets

In TREC 2003 and 2004, a special track for web IR, named the Web track[4], was organized. The track used the "Gov" corpus, which is based on a January, 2002 crawl of the "Gov" domain. There are in total 1,053,110 html documents in this corpus.

Three search tasks were designed in the Web track: topic distillation (TD), homepage finding (HP), and named page finding (NP). *Topic distillation* aims to find a list of entry points for good websites principally devoted to the topic. *Homepage finding* aims at returning the homepage of the query. *Named page finding* aims to return the page whose name is exactly identical to the query. Generally speaking, there is only one answer for homepage finding and named page finding. The numbers of queries in these three tasks are shown in Table. 6.1.

Due to the large scale of the corpus, it is not feasible to check every document and judge whether it is relevant to a given query. Therefore, the pooling strategy as introduced in Chapter 1 was used [35].

Many research papers [97, 131, 101, 133] have been published using the three tasks on the "Gov" corpus as their experimental platform.

6.1.1.2 The OHSUMED Corpus

The OHSUMED corpus [64] is a subset of MEDLINE, a database on medical publications. It consists of 348,566 records (out of over 7 million) from 270 medical journals during the period of 1987-1991. The

[4] http://trec.nist.gov/tracks.html

Table 6.1 Number of queries in TREC Web track

Task	TREC2003	TREC2004
Topic distillation	50	75
Homepage finding	150	75
Named page finding	150	75

fields of a record include title, abstract, MeSH indexing terms, author, source, and publication type.

A query set with 106 queries on the OHSUMED corpus was used in many previous works [97, 131], with each query describing a medical search need (associated with patient information and topic information). The relevance degrees of the documents with respect to the queries are judged by humans, on three levels: definitely relevant, partially relevant, and irrelevant. There are a total of 16,140 query-document pairs with relevance judgments.

6.1.2 Documents Sampling

Due to the similar reason for selecting documents for labeling, it is not feasible to extract feature vectors of all the documents in the corpora either. A reasonable strategy is to sample some "possibly" relevant documents, and then extract feature vectors for the corresponding query-document pairs.

For the "Gov" corpus, following the suggestions in [88] and [100], the documents were sampled in the following way. First, the BM25 model was used to rank all the documents with respect to each query, and then the top 1000 documents for each query were selected for feature extraction. Please note that this sampling strategy is to ease the experimental investigation, and it is by no means to say that learning to rank can only be applicable in such a re-ranking scenario.

Different from the "Gov" corpus where unjudged documents are regarded as irrelevant, in OHSUMED, the judgments explicitly contain the category of "irrelevant" and the unjudged documents are ignored in the evaluation [64]. Following this practice, in LETOR, only judged documents were used for feature extraction in OHSUMED and all the unjudged documents were ignored. On average, a query has about 152 documents sampled for feature extraction.

6.1.3 Features Extraction

In this section, we introduce the feature representation of each document in LETOR.

For the "Gov" corpus, 64 features were extracted for each query document pair, as shown in Table. 6.2.

Table 6.2: Learning Features of TREC

ID	**Feature Description**
1	Term frequency (TF) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of whole document
6	Inverse document frequency (IDF) of body
7	IDF of anchor
8	IDF of title
9	IDF of URL
10	IDF of whole document
11	TF*IDF of body
12	TF*IDF of anchor
13	TF*IDF of title
14	TF*IDF of URL
15	TF*IDF of whole document
16	Document length (DL) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	BM25 of anchor
23	BM25 of title
24	BM25 of URL
25	BM25 of whole document
26	LMIR.ABS of body
27	LMIR.ABS of anchor
28	LMIR.ABS of title
29	LMIR.ABS of URL
30	LMIR.ABS of whole document

31	LMIR.DIR of body
32	LMIR.DIR of anchor
33	LMIR.DIR of title
34	LMIR.DIR of URL
35	LMIR.DIR of whole document
36	LMIR.JM of body
37	LMIR.JM of anchor
38	LMIR.JM of title
39	LMIR.JM of URL
40	LMIR.JM of whole document
41	Sitemap based term propagation
42	Sitemap based score propagation
43	Hyperlink base score propagation: weighted in-link
44	Hyperlink base score propagation: weighted out-link
45	Hyperlink base score propagation: uniform out-link
46	Hyperlink base feature propagation: weighted in-link
47	Hyperlink base feature propagation: weighted out-link
48	Hyperlink base feature propagation: uniform out-link
49	HITS authority
50	HITS hub
51	PageRank
52	HostRank
53	Topical PageRank
54	Topical HITS authority
55	Topical HITS hub
56	Inlink number
57	Outlink number
58	Number of slash in URL
59	Length of URL
60	Number of child page
61	BM25 of extracted title
62	LMIR.ABS of extracted title
63	LMIR.DIR of extracted title
64	LMIR.JM of extracted title

For the OHSUMED corpus, 40 features were extracted in total, as shown in Table 6.3.

Table 6.3: Learning Features of OHSUMED

ID	**Feature Description**
1	$\sum_{q_i \in q \cap d} TF(q_i, d)$ in title
2	$\sum_{q_i \in q \cap d} \log(TF(q_i, d) + 1)$ in title

3	$\sum_{q_i \in q \cap d} \frac{TF(q_i,d)}{LEN(d)}$ in title
4	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} + 1\right)$ in title
5	$\sum_{q_i \in q \cap d} \log\left(IDF(q_i)\right)$ in title
6	$\sum_{q_i \in q \cap d} \log\left(\log\left(IDF(q_i)\right)\right)$ in title
7	$\sum_{q_i \in q \cap d} \log\left(\frac{N}{TF(q_i,C)} + 1\right)$ in title
8	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} \cdot \log\left(IDF(q_i)\right) + 1\right)$ in title
9	$\sum_{q_i \in q \cap d} TF(q_i,d) \cdot \log\left(IDF(q_i)\right)$ in title
10	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} \cdot \frac{N}{TF(q_i,C)} + 1\right)$ in title
11	BM25 of title
12	log(BM25) of title
13	LMIR.DIR of title
14	LMIR.JM of title
15	LMIR.ABS of title
16	$\sum_{q_i \in q \cap d} TF(q_i,d)$ in abstract
17	$\sum_{q_i \in q \cap d} \log\left(TF(q_i,d) + 1\right)$ in abstract
18	$\sum_{q_i \in q \cap d} \frac{TF(q_i,d)}{LEN(d)}$ in abstract
19	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} + 1\right)$ in abstract
20	$\sum_{q_i \in q \cap d} \log\left(IDF(q_i)\right)$ in abstract
21	$\sum_{q_i \in q \cap d} \log\left(\log\left(IDF(q_i)\right)\right)$ in abstract
22	$\sum_{q_i \in q \cap d} \log\left(\frac{N}{TF(q_i,C)} + 1\right)$ in abstract
23	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} \cdot \log\left(IDF(q_i)\right) + 1\right)$ in abstract
24	$\sum_{q_i \in q \cap d} TF(q_i,d) \cdot \log\left(IDF(q_i)\right)$ in abstract
25	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} \cdot \frac{N}{TF(q_i,C)} + 1\right)$ in abstract
26	BM25 of abstract
27	log(BM25) of abstract
28	LMIR.DIR of abstract
29	LMIR.JM of abstract
30	LMIR.ABS of abstract
31	$\sum_{q_i \in q \cap d} TF(q_i,d)$ in title + abstract
32	$\sum_{q_i \in q \cap d} \log\left(TF(q_i,d) + 1\right)$ in title + abstract
33	$\sum_{q_i \in q \cap d} \frac{TF(q_i,d)}{LEN(d)}$ in title + abstract
34	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} + 1\right)$ in title + abstract
35	$\sum_{q_i \in q \cap d} \log\left(IDF(q_i)\right)$ in title + abstract
36	$\sum_{q_i \in q \cap d} \log\left(\log\left(IDF(q_i)\right)\right)$ in title + abstract
37	$\sum_{q_i \in q \cap d} \log\left(\frac{N}{TF(q_i,C)} + 1\right)$ in title + abstract
38	$\sum_{q_i \in q \cap d} \log\left(\frac{TF(q_i,d)}{LEN(d)} \cdot \log\left(IDF(q_i)\right) + 1\right)$ in title + abstract
39	$\sum_{q_i \in q \cap d} TF(q_i,d) \cdot \log\left(IDF(q_i)\right)$ in title + abstract

40	$\sum_{q_i \in q \cap d} \log \left(\frac{TF(q_i,d)}{LEN(d)} \cdot \frac{N}{TF(q_i,C)} + 1 \right)$ in title + abstract
41	BM25 of title + abstract
42	log(BM25) of title + abstract
43	LMIR.DIR of title + abstract
44	LMIR.JM of title + abstract
45	LMIR.ABS of title + abstract

6.1.4 Meta Information

In addition to the features, meta information that can be used to reproduce these features and even other new features have also been provided in LETOR. There are three kinds of meta information.

- Statistical information about the corpus, such as the total number of documents, the number of streams, and the number of (unique) terms in each stream.
- Raw information of the documents associated with each query, such as the term frequency, and the document length.
- Relational information, such as the hyperlink graph, the sitemap information, and the similarity relationship matrix of the corpus.

With the meta information, one can reproduce existing features, tune their parameters, investigate new features, and perform some advanced researches such as relational ranking [102] [103].

6.1.5 Cross Validation

In total, there are seven datasets in the LETOR collection, i.e., TD2003, TD2004, NP2003, NP2004, HP2003, HP2004 and OHSUMED. Each of these datasets was partitioned into five parts with about the same number of queries, denoted as S1, S2, S3, S4, and S5, in order to conduct five-fold cross validation. For each fold, three parts are used for training the ranking model, one part for tuning the hyper parameters of the ranking algorithm (e.g., the number of iterations in RankBoost, and the combination coefficient in the objective function of Ranking SVM), and the remaining part to evaluate the ranking performance of the

Table 6.4 Data Partitioning for 5-fold Cross Validation

Folds	Training set	Validation set	Test set
Fold1	{S1,S2,S3}	S4	S5
Fold2	{S2,S3,S4}	S5	S1
Fold3	{S3,S4,S5}	S1	S2
Fold4	{S4,S5,S1}	S2	S3
Fold5	{S5,S1,S2}	S3	S4

learned model (See Table 6.4). The average performance over the five folds is used to measure the overall performance of a learning-to-rank algorithm.

The LETOR collection, containing the aforementioned feature representations of documents, their relevance judgments with respective to queries, and the partitioned training, validation and test sets can be downloaded from http://research.microsoft.com/~LETOR/[5].

6.2 Experimental Results on LETOR

In this section, we introduce the experiments on LETOR to evaluate several representative learning-to-rank algorithms, and make discussions on the experimental results[6]. Three widely used measures are adopted for the evaluation: precision at position k (P@k) [8], mean average precision (MAP) [8], and normalized discounted cumulative gain at position k (N@k) [66]. The official evaluation tool provided with LETOR was used in the evaluation process.

We have evaluated seven representative learning-to-rank algorithms in total. For the pointwise approach, we tested the regression based method. For the pairwise approach, we tested Ranking SVM, RankBoost, and FRank. For the listwise approach, we tested ListNet, AdaRank, and SVM^{map}. To make fair comparisons, we tried to use the same setting for all the algorithms. Firstly, most algorithms use the linear ranking function, except RankBoost, which uses binary weak

[5] Note that the LETOR collection is being frequently updated. It is expected that more datasets will be added in the future.
[6] These results are also published at the website of LETOR.

Table 6.5 Results on the TD2003 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.320	0.307	0.326	0.320	0.260	0.178	0.241
RankSVM	0.320	0.344	0.346	0.320	0.293	0.188	0.263
RankBoost	0.280	0.325	0.312	0.280	0.280	0.170	0.227
FRank	0.300	0.267	0.269	0.300	0.233	0.152	0.203
ListNet	0.400	0.337	0.348	0.400	0.293	0.200	0.275
AdaRank	0.260	0.307	0.306	0.260	0.260	0.158	0.228
SVM^{map}	0.320	0.320	0.328	0.320	0.253	0.170	0.245

rankers. Secondly, all the algorithms used MAP on the validation set for model selection. Some detailed experimental settings are listed as follows.

As for the linear regression based algorithm, the validation set was used to select a good mapping from the ground truth labels to real values. For Ranking SVM, the public tool of SVMlight was employed and the validation set was used to tune the parameter C in its loss function. For RankBoost, the weak ranker was defined on the basis of a single feature with 255 possible thresholds. The validation set was used to determine the best number of iterations. For FRank, to efficiently minimize the fidelity loss, a generalized additive model was adopted. The validation set was used to determine the number of weak learners in the additive model. For ListNet, the validation set was used to determine the best mapping from the ground truth label to scores in order to use the Luce model, and to determine the optimal number of iterations in the gradient descent process. For AdaRank, MAP was set as the IR evaluation measure to be optimized, and the validation set was used to determine the number of iterations. For SVM^{map} [136], the publicly available tool SVM^{map} was employed, and the validation set was used to determine the parameter C in its loss function.

The ranking performances of the aforementioned algorithms are listed in Table 6.5 - 6.11. According to these experimental results, we find that listwise ranking algorithms perform very well on most datasets. Among the three listwise ranking algorithms, ListNet seems to be better than the others. AdaRank and SVM^{map} obtain similar per-

Table 6.6 Results on the TD2004 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.360	0.335	0.303	0.360	0.333	0.249	0.208
RankSVM	0.413	0.347	0.307	0.413	0.347	0.252	0.224
RankBoost	0.507	0.430	0.350	0.507	0.427	0.275	0.261
FRank	0.493	0.388	0.333	0.493	0.378	0.262	0.239
ListNet	0.360	0.357	0.317	0.360	0.360	0.256	0.223
AdaRank	0.413	0.376	0.328	0.413	0.369	0.249	0.219
SVM^{map}	0.293	0.304	0.291	0.293	0.302	0.247	0.205

Table 6.7 Results on the NP2003 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.447	0.614	0.665	0.447	0.220	0.081	0.564
RankSVM	0.580	0.765	0.800	0.580	0.271	0.092	0.696
RankBoost	0.600	0.764	0.807	0.600	0.269	0.094	0.707
FRank	0.540	0.726	0.776	0.540	0.253	0.090	0.664
ListNet	0.567	0.758	0.801	0.567	0.267	0.092	0.690
AdaRank	0.580	0.729	0.764	0.580	0.251	0.086	0.678
SVM^{map}	0.560	0.767	0.798	0.560	0.269	0.089	0.687

formances. Pairwise ranking algorithms obtain good ranking accuracy on some (although not all) datasets. For example, RankBoost offers the best performance on TD2004 and NP2003; Ranking SVM shows very promising results on NP2003 and NP2004; and FRank achieves very

Table 6.8 Results on the NP2004 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.373	0.555	0.653	0.373	0.200	0.082	0.514
RankSVM	0.507	0.750	0.806	0.507	0.262	0.093	0.659
RankBoost	0.427	0.627	0.691	0.427	0.231	0.088	0.564
FRank	0.480	0.643	0.729	0.480	0.236	0.093	0.601
ListNet	0.533	0.759	0.812	0.533	0.267	0.094	0.672
AdaRank	0.480	0.698	0.749	0.480	0.244	0.088	0.622
SVM^{map}	0.520	0.749	0.808	0.520	0.267	0.096	0.662

Table 6.9 Results on the HP2003 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.420	0.510	0.594	0.420	0.211	0.088	0.497
RankSVM	0.693	0.775	0.807	0.693	0.309	0.104	0.741
RankBoost	0.667	0.792	0.817	0.667	0.311	0.105	0.733
FRank	0.653	0.743	0.797	0.653	0.289	0.106	0.710
ListNet	0.720	0.813	0.837	0.720	0.320	0.106	0.766
AdaRank	0.733	0.805	0.838	0.733	0.309	0.106	0.771
SVM^{map}	0.713	0.779	0.799	0.713	0.309	0.100	0.742

Table 6.10 Results on the HP2004 dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.387	0.575	0.646	0.387	0.213	0.08	0.526
RankSVM	0.573	0.715	0.768	0.573	0.267	0.096	0.668
RankBoost	0.507	0.699	0.743	0.507	0.253	0.092	0.625
FRank	0.600	0.729	0.761	0.600	0.262	0.089	0.682
ListNet	0.600	0.721	0.784	0.600	0.271	0.098	0.690
AdaRank	0.613	0.816	0.832	0.613	0.298	0.094	0.722
SVM^{map}	0.627	0.754	0.806	0.627	0.280	0.096	0.718

good results on TD2004 and NP2004. Comparatively speaking, simple linear regression performs worse than the pairwise and listwise ranking algorithms. Its results are not so good on most datasets.

We have also observed that most ranking algorithms perform dif-

Table 6.11 Results on the OHSUMED dataset

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	0.446	0.443	0.411	0.597	0.577	0.466	0.422
RankSVM	0.496	0.421	0.414	0.597	0.543	0.486	0.433
RankBoost	0.463	0.456	0.430	0.558	0.561	0.497	0.441
FRank	0.530	0.481	0.443	0.643	0.593	0.501	0.444
ListNet	0.533	0.473	0.441	0.652	0.602	0.497	0.446
AdaRank	0.539	0.468	0.442	0.634	0.590	0.497	0.449
SVM^{map}	0.523	0.466	0.432	0.643	0.580	0.491	0.445

Table 6.12 Winner Number of Each Algorithm

Algorithm	N@1	N@3	N@10	P@1	P@3	P@10	MAP
Regression	4	4	4	5	5	5	4
RankSVM	21	22	22	21	22	22	24
RankBoost	18	22	22	17	22	23	19
FRank	18	19	18	18	17	23	15
ListNet	29	31	33	30	32	35	33
AdaRank	26	25	26	23	22	16	27
SVM^{map}	23	24	22	25	20	17	25

ferently on different datasets. They may perform very well on some datasets but not so well on other datasets. To evaluate the overall ranking performances of an algorithm, we use the number of other algorithms that it can beat over all the seven datasets as a measure. That is,

$$S_i(M) = \sum_{j=1}^{7} \sum_{k=1}^{7} I_{\{M_i(j) > M_k(j)\}}$$

where j is the index of a dataset, i and k are the indexes of algorithms, $M_i(j)$ is the performance of i-th algorithm on j-th dataset, and $I_{\{\cdot\}}$ is the indicator function.

It is clear that the larger $S_i(M)$, the better the i-th algorithm performs. For ease of reference, we call this measure the *winner number*. Table 6.12 shows the winner number for all the algorithms under investigation. From this table, we have the following observations.

(1) In terms of NDCG@1, the listwise ranking algorithms perform the best, followed by the pairwise ranking algorithms, while the pointwise algorithm performs the worse. Among the three listwise ranking algorithms, ListNet is better than AdaRank, while SVM^{map} performs a little worse than the others. The three pairwise ranking algorithms achieve comparable results, among which Ranking SVM seems to be slightly better than the other two.

(2) In terms of NDCG@3 and NDCG@10, ListNet and AdaRank perform much better than the pairwise and pointwise ranking

algorithms, while the performance of SVM^{map} is very similar to the pairwise ranking algorithms.

(3) In terms of P@1 and P@3, the listwise ranking algorithms perform the best, followed by the pairwise ranking algorithms, while the pointwise ranking algorithm performs the worse. Among the three listwise ranking algorithms, ListNet is better than AdaRank and SVM^{map}. The three pairwise ranking algorithms achieve comparable results, among which Ranking SVM seems to be slightly better than the other two algorithms.

(4) In terms of P@10, ListNet performs much better than the pairwise and pointwise ranking algorithms, while the performances of AdaRank and SVM^{map} are not as good as those of the pairwise ranking algorithms.

(5) In terms of MAP, the listwise ranking algorithms are in general better than the pairwise ranking algorithms. Furthermore, the variance among the three pairwise ranking algorithms in terms of MAP is much larger than the variance among the three algorithms in terms of other measures (e.g., P@1,3 and 10). The possible explanation is that since MAP involves all the documents associated with a query in the evaluation process, it can better differentiate algorithms.

To summarize, the experimental results show that the listwise algorithms have certain advantages over other algorithms, especially for top positions of the ranking result.

Here, we would like to point out that the above experimental results are still primal, since the result of almost every algorithm can be further improved. For example, for regression, we can add regularization item to make it more robust; for Ranking SVM, we can run more steps of iteration so as to guarantee a better convergence of the optimization; for ListNet, we can also add regularization item to its loss function and make it more generalizable to the test set.

7

Statistical Ranking Theory

As a new machine learning problem, ranking is not only about effective algorithms but also about the theory behind these algorithms. In this chapter, we will introduce the so-called statistical ranking theory [76] [75], and will focus on the generalization analysis of learning-to-rank methods.

Actually, theoretical analysis on an algorithm always plays an important role in machine learning. This is because in practice, one can only observe experimental results on small-scale datasets (e.g., the experimental results on the LETOR collection as introduced in Chapter 6). To some extent, such empirical results might not be fully trustworthy, because a small training set sometimes cannot fully realize the potential of a learning algorithm, and a small test set sometimes cannot reflect the true performance of an algorithm due to the fact that the input and output spaces are too large to be well represented by a small number of samples. In this regard, theories are solely needed in order to analyze the performance of a learning algorithm when the training data is infinite and the test data is perfectly sampled from the input and output spaces.

For example, the generalization analysis on an algorithm is concerned with whether and at what rate its empirical risk computed from the training data will converge to the expected risk on any test data in the input and output spaces, when the number of training samples approaches infinity. Sometimes, we alternatively represent the generalization ability of an algorithm using the bound of the difference between its expected risk and empirical risk, and see whether and at what rate the bound will converge to zero when the number of training samples approaches infinity. In the context of learning to rank, the expected risk measures the average error that a ranking model would make on a randomly sampled input instance (document, document pairs, or all documents associated with a query), while the empirical risk is an estimate of the expected risk based on the training data. In general, an algorithm is regarded as better than the other algorithm if its empirical risk can converge to the expected risk but that of the other cannot. Furthermore, an algorithm is regarded as better than the other if its corresponding convergence rate is faster than that of the other.

To facilitate the discussions on (but not limited to) the generalization ability, a theoretical framework is needed. We will first review some conventional generalization analyses on ranking and their corresponding theoretical frameworks. After that, a recently-proposed query-level ranking framework and its use in analyzing the generalization ability of ranking methods will be introduced.

Note that it is unavoidable that a lot of mathematics will be used in this chapter. It is safe, however, to skip this whole chapter, if one only wants to know the algorithmic development of learning to rank.

7.1 Conventional Generalization Analyses on Ranking

Most of the conventional generalization analyses on ranking are for the pairwise approach to learning to rank. The theoretical framework behind these analyses is given in the following subsection.

7.1.1 Theoretical Framework for Pairwise Ranking

The basic assumption in the theoretical framework is that the documents and their relevance degrees are i.i.d. random variables. In this case, the document pairs, as random variables, are not independent of each other. To handle the challenge, U-statistics is used as the tool to perform generalization analysis.

Assume that (X_u, Y_u) and (X_v, Y_v) are i.i.d. random variables according to distribution P_{XY}, where X stands for the document and Y stands for the ground truth label of the document. (X_u, Y_u) and (X_v, Y_v) construct a pair. Given the scoring function f, loss comes up if the documents are not ranked according to their ground truth labels. Suppose the loss function is $L(f, x_u, x_v, y_{u,v})$, where $y_{u,v} = 2 \cdot I_{\{y_u \succ y_v\}} - 1$. Then the *expected risk* for ranking is defined as,

$$R(f) = \int_{(\mathcal{X}\times\mathcal{Y})^2} L(f, x_u, x_v, y_{u,v})\, P_{XY}(dx_u, dy_u)\, P_{XY}(dx_v, dy_v). \quad (7.1)$$

Intuitively, the expected risk means the average loss that a ranking model f would make for a random document pair. Since it is almost impossible to compute the expected risk, in practice, the empirical risk on the training set is used as an estimate of the expected risk. In particular, given the training data $\{(x_j, y_j)\}_{j=1}^m$, the *empirical risk* can be defined with the following U-statistics,

$$\hat{R}(f) = \frac{1}{C_m^2} \sum_{u=1}^{m} \sum_{v=u+1}^{m} L(f, x_u, x_v, y_{u,v}), \quad (7.2)$$

where $C_m^2 = m(m-1)/2$.

Specifically, when the ground truth is given as binary relevance degree, the positive example is denoted as X^+ according to P^+ and the negative example is denoted as X^- according to P^-. Given the training data $\mathbf{x}^+ = \{x_j^+\}_{j=1}^{m^+}$ and $\mathbf{x}^- = \{x_j^-\}_{j=1}^{m^-}$ (where m^+ and m^- are the numbers of positive and negative examples in the training data respectively), the *expected risk* and the *empirical risk* are refined as follows,

$$R(f) = \int_{\mathcal{X}^2} L(f, x^+, x^-) P^+(dx^+) P^-(dx^-), \quad (7.3)$$

$$\hat{R}(f) = \frac{1}{m^+ m^-} \sum_{u=1}^{m^+} \sum_{v=1}^{m^-} L(f, x^+, x^-). \quad (7.4)$$

7.1.2 Generalization Analysis on the Pairwise Approach

With the above theoretical framework, several tools have been used to conduct generalization analyses for the pairwise approach to learning to rank. Here we give three examples. VC dimension and rank shatter coefficient measure the complexity of the function class, while stability measures the robustness of a ranking algorithm. Basically, simpler ranking models generalize better, and more robust ranking algorithms generalize better.

Based on VC Dimension

In [47], VC dimension [126] [125] is used to obtain the generalization bound for RankBoost. It is clear the bound converges to zero at a rate of $O(\max\{\sqrt{\frac{\log(m^+)}{m^+}}, \sqrt{\frac{\log(m^-)}{m^-}}\})$.

Theorem 7.1. Assume that all the weak learners belong to the function class $\mathcal{F}'$, which has a finite VC dimension V, the scoring function belong to function class $\mathcal{F}$. Let S^+ and S^- be positive and negative samples of size m^+ and m^- respectively. Then with probability at least $1-\delta$, the following inequality holds,

$$\forall f \in \mathcal{F}, |R(f) - \hat{R}(f)| \leq$$
$$2\sqrt{\frac{V'(\log\frac{2m^+}{V'}+1)+\log\frac{18}{\delta}}{m^+}} + 2\sqrt{\frac{V'(\log\frac{2m^-}{V'}+1)+\log\frac{18}{\delta}}{m^-}}, \quad (7.5)$$

where $V' = 2(V+1)(T+1)\log_2(e(T+1))$ and T is the number of weak rankers in RankBoost.

The above theorem is only applicable to the bipartite case. Clemencon, et al. proposed another theorem using the properties of U-Statistics [27], which can handle the general case.

Based on Rank Shatter Coefficient

The notion of the bipartite rank-shatter coefficient, denoted as $r(\mathcal{F}, m^+, m^-)$, was proposed by Agarwal et al. [1]. Here $\mathcal{F}$ is the class of scoring functions, m^+ is the number of relevant documents and m^- is the number of irrelevant documents. This new notion has a similar meaning to the shattering coefficient (growth function) in the VC theory [126] [125].

Using the bipartite rank-shatter coefficient as a tool, the following generalization theorem has been proved. As shown in [1], for the class of linear ranking functions in the one-dimensional feature space, $r(\mathcal{F}, m^+, m^-)$ is a constant, regardless of the values of m^+ and m^-. In this case, the bound converges to zero at a rate of $O(\max\{\frac{1}{\sqrt{m^+}}, \frac{1}{\sqrt{m^-}}\})$, and is therefore tighter than the bound for RankBoost given in inequality (7.5). For the class of linear ranking functions in the d-dimensional feature space ($d > 1$), $r(\mathcal{F}, m^+, m^-)$ is of the order $O((m^+m^-)^d)$, and in this case the bound has a similar convergence rate to that base on VC dimension, i.e., $O(\max\{\sqrt{\frac{\log(m^+)}{m^+}}, \sqrt{\frac{\log(m^-)}{m^-}}\})$.

Theorem 7.2. Let $\mathcal{F}$ be the class of real-valued functions on $\mathcal{X}$, then $\forall 0 < \delta < 1$, with probability at least $1 - \delta$,

$$\forall f \in \mathcal{F}, |R(f) - \hat{R}(f)| \leq \sqrt{\frac{8(m^+ + m^-)(\log \frac{4}{\delta} + \log r(\mathcal{F}, 2m^+, 2m^-))}{m^+m^-}} \quad (7.6)$$

Furthermore, Rajaram and Agarwal [108] generalized the above theory to the k-partite case using the *k-partite rank-shatter coefficient*.

Based on the Stability Theory

Agarwal and Niyogi [2] used the stability as a tool and derived the generalization bound for some bipartite ranking algorithms.

Let $\mathcal{A}$ be a ranking algorithm, and let L be the loss function that is minimized in $\mathcal{A}$. Suppose we have learned a ranking model f_1 from the training data using algorithm $\mathcal{A}$. Then we replace a relevant training document with another document and learn a new model f_2 from the new training data. Similarly, we replace an irrelevant training document with another document and learn a new model f_3. We say that $\mathcal{A}$ has uniform loss stability (α, β) with respect to L, if the difference between the losses with respect to f_1 and f_2 on any unseen document pair $(x+, x-)$ is smaller than $\alpha(m^+, m^-)$, and that with respect to f_1 and f_3 is smaller than $\beta(m^+, m^-)$. In other words, if the model learned from the training data is robust to the small change in the training data, the algorithm is regarded as having certain stability.

The generalization bound they obtained is as shown in Theorem 7.3. As shown in [12], for many algorithms, $\alpha(m^+, m^-)$ and $\beta(m^+, m^-)$ are of the order $O(\frac{1}{m^+})$ and $O(\frac{1}{m^-})$ respectively. Therefore, the bound given in Theorem 7.3 will converge to zero at a rate of $O(\max\{\frac{1}{\sqrt{m^+}}, \frac{1}{\sqrt{m^-}}\})$. In this regard, this bound is tighter than the bound based on the Rank Shatter Coefficient (see inequality (7.6)) except for the case of using linear ranking functions in the one-dimensional feature space. This is understandable since the use of stability can lead to a data dependent bound [12], which is usually tighter than data-independent bounds.

Theorem 7.3. Let l be the loss function with $L(f; x^+, x^-) \in [0, B]$, and $\mathcal{A}$ has uniform loss stability (α, β). Then $\forall 0 < \delta < 1$, with probability at least $1 - \delta$, the following inequality holds,

$$R_L(f_{S^+,S^-}) \leq \hat{R}_L(f_{S^+,S^-}) + \alpha(m^+, m^-) + \beta(m^+, m^-) + \sqrt{\frac{\{m^-(2m^+\alpha(m^+, m^-) + B)2 + m^+(2m^-\beta(m^+, m^-) + B)^2\} \log \frac{1}{\delta}}{2m^+m^-}}. \tag{7.7}$$

Due to the basic assumption of the theoretical framework, the aforementioned theorems can only be used when the ground truth labels are given as relevance degrees of all the documents. However, as we mentioned before, the ground truth labels can also be given in terms the pairwise preference and even total order of the documents. In this case, new framework needs to be developed.

When the ground truth is given as pairwise preference, it is more reasonable to assume document pairs and their ground truth labels are i.i.d. random variables. In this way, ranking is actually formalized as a standard classification problem on document pairs. By using the theoretical results of classification, one can also get a generalization bound. For ease of reference, we refer to this way as taking the "average view" on the generalization analysis, and refer to the work introduced in this section as taking the "U-statistics view" on the generalization analysis.

Note that the "average view" is also technically sound. The intuition is not right that two document pairs cannot be independent of each other when they share a common document. The reason is that the dependence (or independence) is actually defined with regards to random variables but not their values. Therefore, as long as two document pairs are sampled and labeled in an independent manner, they are i.i.d. random variables no matter whether their values (the specific documents in the pair) have overlap or not. More discussions about the average view and the U-statistics view will be given in the next section.

7.2 A Query-level Ranking Framework

In [76], Lan, et al. argued the limitation of the aforementioned generalization analyses, from the IR point of view.

As we know, the generalization ability is concerned with the difference between the empirical risk and the expected risk of learning, and these risks are highly related to how one evaluates the performance of a learning algorithm. As mentioned earlier in the tutorial, all the IR evaluation measures are defined at the query level. Therefore, the definition of the risks in ranking, and the generalization analysis for ranking should also be at the query level.

In contrast, the previous generalization analyses are either performed at the document level or at the document pair level. Therefore, Lan, et al. argued that a novel theoretical framework needs to be developed for learning to rank, to facilitate (but not limited to) the query-level generalization analysis. They call such a framework the query-level ranking framework.

Let $\mathcal{Q}$ be the query space, $\mathcal{D}$ be the document space, and $\mathcal{X}$ be the d-dimensional space containing feature representation of each document. Let q be a random variable defined on the query space with an unknown probability distribution $P_{\mathcal{Q}}$. Denote f as the scoring function, and use a loss function $L(f;q)$ to measure the loss for each query. Then the goal of ranking is to minimize the following expected *query-level* risk,

$$R(f) = \int_{\mathcal{Q}} L(f;q) P_{\mathcal{Q}}(dq). \tag{7.8}$$

Intuitively, the expected query-level risk means the average loss that the ranking model f would make on a randomly sampled query. Since the distribution $P_{\mathcal{Q}}$ is unknown, the following empirical query-level risk on the training set is used as the estimate of the expected query-level risk,

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} L(f;q_i), \tag{7.9}$$

where $q_1, \cdots, q_n$ are i.i.d observations with the same distribution as that of q.

The forms of $L(f;q)$ corresponding to the pointwise, pairwise and listwise approaches are given as follows.

The Pointwise Approach

In the pointwise approach, each document x is given a ground truth label[1] $y \in \mathcal{Y}$, which stands for the relevance degree of x with regards to a specific query q.

For a fixed query q, (x, y) is assumed to be a random variable sampled according to probability distribution $\mathcal{D}_q^{(1)}$ (which is dependent on query q). Then, $L(f;q)$ is defined as below,

$$L(f;q) = \int_{\mathcal{X}\times\mathcal{Y}} L(f;x,y)\, \mathcal{D}_q^{(1)}(dx, dy). \tag{7.10}$$

Where $L(f;x,y)$ is the loss[2] of the scoring function f on sample (x, y).

[1] For example, $\mathcal{Y} = \{0, 1\}$ for binary classification based algorithms, and $\mathcal{Y} = \mathbb{R}$ (where $\mathbb{R}$ stands for the space of real values) for regression-based algorithms.

[2] For example, $L(f;x,y)$ can be defined as $I_{\{\mathrm{sgn}(f(x)) \neq y\}}$ (indicator function) in binary classification and $(f(x) - y)^2$ in regression.

As the distribution $\mathcal{D}_q^{(1)}$ is unknown, the average of the loss over a set of training samples is used to estimate $L(f;q)$,

$$\hat{L}(f;q) = \frac{1}{m}\sum_{j=1}^{m} L(f;x_j,y_j), \tag{7.11}$$

where $\{x_j, y_j\}_{j=1}^m$ stands for m i.i.d observations with distribution $\mathcal{D}_q^{(1)}$.

The Pairwise Approach

As introduced in the previous section, there are two views on the generalization analysis of the pairwise approach: the *U-statistics* view and the *average* view. For both views, it is not difficult to extend them to be fitted into the query-level ranking framework.

(1) The U-statistics View

With the U-statistics view, one assumes i.i.d. distribution of the documents and their ground truth labels with respect to a query. Given two documents associated with query q and their ground truth labels, (x_1, y_1) and (x_2, y_2), (x_1, x_2, y_1, y_2) is regarded as a random variable with probabilistic distribution $\mathcal{D}_q^{(2)}$. Then $L(f;q)$ can be defined as follows,

$$L(f;q) = \int_{(\mathcal{X}\times\mathcal{Y})^2} L(f;x_1,x_2,y_1,y_2)\, \mathcal{D}_q^{(2)}(dx_1,dx_2,dy_1,dy_2), \tag{7.12}$$

where $L(f;x_1,x_2,y_1,y_2)$ is the loss of f on sample (x_1,x_2,y_1,y_2).

As the distribution $\mathcal{D}_q^{(2)}$ is unknown, the following U-statistics is used to estimate $L(f;q)$,

$$\hat{L}(f;q) = \frac{1}{C_m^2}\sum_{u=1}^{m}\sum_{v=u+1}^{m} L(f;x_u,x_v,y_u,y_v), \tag{7.13}$$

where $(x_u,x_v,y_u,y_v), u,v = 1,\cdots,m$ can be viewed as C_m^2 observations with distribution $\mathcal{D}_q^{(2)}$, which might not be independent of each other.

Since two document pairs are not independent under the aforementioned assumptions, to perform the generalization analysis, one needs to use some advanced statistical tools such as [27] and [124].

(2) The Average View

The average view assumes the i.i.d. distribution of document pairs. More specifically, with the average view, each document pair (x_1, x_2) is given a ground truth label $y \in \mathcal{Y} = \{-1, +1\}$, where $y = +1$ indicates that document x_1 is more relevant than x_2 and $y = -1$ otherwise. Then (x_1, x_2, y) is assumed to be a random variable with probabilistic distribution $\mathcal{D}_q^{(3)}$, and $L(f; q)$ can be defined as follows,

$$L(f;q) = \int_{\mathcal{X}\times\mathcal{X}\times\mathcal{Y}} L(f; x_1, x_2, y)\, \mathcal{D}_q^{(3)}(dx_1, dx_2, dy), \tag{7.14}$$

where $L(f; x_1, x_2, y)$ is the loss of f on sample (x_1, x_2, y).

As the distribution $\mathcal{D}_q^{(3)}$ is unknown, the average of the loss over training document pairs is used to estimate $L(f; q)$,

$$\hat{L}(f;q) = \frac{1}{\tilde{m}} \sum_{j=1}^{\tilde{m}} L(f; x_{j_1}, x_{j_2}, y_j), \tag{7.15}$$

where $\{x_{j_1}, x_{j_2}, y_j\}_{j=1}^{\tilde{m}}$ stands for $\tilde{m}$ i.i.d observations with distribution $\mathcal{D}_q^{(3)}$, and $\tilde{m}$ is the number of document pairs associated with query q in the training data.

The Listwise Approach

In the listwise approach, each query is represented by a group of documents (denoted by $\mathbf{x}$) and their ground truth labels, and queries are assumed to be i.i.d. random variables. As mentioned earlier, there are two types of output spaces used in the listwise approach. For simplicity and clarity, in this section, our discussion focuses on the case where π_y is used as the ground truth label. And furthermore, we will take re-ranking as our target application[3], where there are the same number of documents (i.e., m) for every query under investigation.

Let $\mathcal{X}^m$ be the input space, whose elements are m feature vectors corresponding to m documents associated with a query[4]. Let $\mathcal{Y}$ be the output space, whose elements are permutations of m documents. Then $(\mathbf{x}, \pi_y)$ can be regarded as a random variable sampled from the space $\mathcal{X}^m \times \mathcal{Y}$ according to an unknown probability distribution $P(\cdot,\cdot)$.

Define

$$L(f;q) = L(f;\mathbf{x},\pi_y), \tag{7.16}$$

where $L(f;\mathbf{x},\pi_y)$ denotes a listwise ranking loss (e.g., the likelihood loss in [129] and the K-L divergence loss in [17]) defined on the random variable $(\mathbf{x},\pi_y)$ and a scoring function f.

7.3 Query-level Generalization Analysis

In this section, we review the works on query-level generalization analysis by using the aforementioned query-level ranking framework. Here the query-level generalization analysis is concerned with whether and at what convergence rate, the empirical query-level risk converges to the expected query-level risk, when the number of training queries approaches infinity. Please note the differences between the query-level generalization analysis and the previous works on generalization analysis as reviewed in Section 7.1.1. As for the query-level generalization analysis, one only cares about the number of training queries, but not the number of training documents. In previous works, however, it is the number of training documents that really matters for the generalization bound.

7.3.1 On the Pairwise Approach

In [76], the stability theory [12] was extended to perform the query-level generalization analysis on the pairwise approach. The *average view* is

[3] In the re-ranking application, "all the documents" may mean all the documents under investigation, e.g., the top 1000 documents with respect to a query returned by a conventional ranking model like BM25.

[4] We call m the list length in this tutorial, and suppose $m \geq 3$. Otherwise the listwise approach will be reduced to the pointwise or pairwise approaches.

taken in the analysis.

To assist the analysis, the definition of uniform leave-one-query-out pairwise loss stability (also referred to as *query-level stability* for short) was given [76]. Suppose we have learned a ranking model f_1 from a training set with n queries, using an algorithm $\mathcal{A}$. Suppose L is the loss function that is minimized in algorithm $\mathcal{A}$. Then we randomly remove a query and all its associated document pairs from the training data, and learn a ranking model f_2 from the new training set. If the difference between the losses with respect to f_1 and f_2 on any unseen document pair (x_1, x_2) is smaller than $\tau(n)$, we say the algorithm $\mathcal{A}$ has uniform leave-one-query-out pairwise loss stability with coefficient τ with respect to L.

Based on the concept of the query-level stability, a query-level generalization bound has been derived in [76], as shown in Theorem 7.4. The theorem states that if a pairwise ranking algorithm has the query-level stability, then with a large probability, the expected query-level risk can be bounded by the empirical query-level risk and a term that depends on the query number and the stability of the algorithm.

Theorem 7.4. Let $\mathcal{A}$ be a learning-to-rank algorithm, $(q_1, S^{(1)}), \cdots, (q_n, S^{(n)})$ be n training queries, and let L be the pairwise loss function. If

(1) $\forall (q_1, S^{(1)}), \cdots, (q_n, S^{(n)}), \forall q \in \mathcal{Q}, (x_1, x_2, y) \in \mathcal{X} \times \mathcal{X} \times \mathcal{Y}$, $\left| L\left(f_{(q_i, S^{(i)})_{i=1}^n}, x_1, x_2, y\right)\right| \leq B$,
(2) $\mathcal{A}$ has query-level stability with coefficient τ,

then $\forall \delta \in (0,1)$ with probability at least $1-\delta$ over the samples of $\left\{(q_i, S^{(i)})\right\}_{i=1}^n$ in the product space $\prod_{i=1}^n \{\mathcal{Q} \times (\mathcal{X} \times \mathcal{X} \times \mathcal{Y})^\infty\}$, the following inequality holds,

$$R_L\left(f_{\{(q_i,S^{(i)})\}_{i=1}^n}\right) \leq \widehat{R_L}\left(f_{\{(q_i,S^{(i)})\}_{i=1}^n}\right) + 2\tau(n) + (4n\tau(n) + B)\sqrt{\frac{\log\frac{1}{\delta}}{2n}}. \tag{7.17}$$

When using this theorem to perform the query-level generalization analysis on pairwise ranking algorithms, what one needs to do is to compute the query-level stability coefficient $\tau(n)$ of the algorithms.

Query-level Generalization Bound for Ranking SVM

As proved in [76], Ranking SVM has query-level stability with coefficient $\tau(n) = \frac{4\kappa^2}{\lambda n} \times \max \frac{\tilde{m}^{(i)}}{\frac{1}{n}\sum_{i=1}^{n} \tilde{m}^{(i)}}$, where $\tilde{m}^{(i)}$ is the number of document pairs associated with query q_i, and $\forall x \in \mathcal{X}, K(x,x) \leq \kappa^2 < \infty$.

On this basis, one can have the following discussions regarding the query-level generalization ability of Ranking SVM.

- When the number of training queries approaches infinity, with a large probability the empirical query-level risk of Ranking SVM will converge to its expected query-level risk, at a rate of $O(\frac{1}{\sqrt{n}})$.
- When the number of training queries is finite, the expected query-level risk and the empirical query-level risk are not necessarily close to each other.

Query-level Generalization Bound for IR-SVM

As proved in [76], IR-SVM [16] has a query-level stability with coefficient $\tau(n) = \frac{4\kappa^2}{\lambda n}$.

On this basis, one can find that when the number of training queries approaches infinity, with a large probability the empirical query-level risk of IR-SVM will converge to its expected query-level risk, at a convergence rate of $O(\frac{1}{\sqrt{n}})$. When the number of queries is finite, the query-level generalization bound is a decreasing function of the number of training queries.

By comparing the query-level generalization abilities of Ranking SVM and IR-SVM, we can find that the convergence rates of the empirical query-level risks to the expected query-level risks for these two algorithms are both $O(\frac{1}{\sqrt{n}})$. However, by comparing the case with a finite number of training queries, the bound for IR-SVM is much tighter than that for Ranking SVM.

7.3.2 On the Listwise Approach

In [75], the theory of Rademacher average [10] [11] was extended to perform the query-level generalization analysis on the listwise approach. Specifically, the algorithms that minimize listwise ranking losses introduced in Section 4.2 were taken as examples.

The Rademacher average measures how much the function class can fit random noise, which is defined below.

Definition 7.1. For a function class $\mathcal{G}$, the empirical Rademacher average is defined as,

$$\widehat{\mathcal{R}}(\mathcal{G}) = E_\sigma \sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i g(z_i), \tag{7.18}$$

where $z_i, i = 1,\cdots,n$ are i.i.d. random variables, and $\sigma_i, i = 1,\cdots,n$ are i.i.d. random variables, with probability $\frac{1}{2}$ of taking a value of $+1$ or -1.

By fitting the theory of Rademacher average [10] [11] to the query-level ranking framework, the query-level generalization bound for listwise ranking algorithms have been derived [75], as shown in the following theorem. Here it is assumed that $\forall x \in \mathcal{X}, \|x\| \leq M$, and the ranking function f is learned from the linear function class $\mathcal{F} = \{x \to w^T x : \|w\| \leq B.\}$, for simplicity. In this case, one has $\forall x \in \mathcal{X}, |f(x)| \leq BM$.

Theorem 7.5. Let $\mathcal{A}$ denote a listwise ranking algorithm, and let $L_{\mathcal{A}}(f;\mathbf{x},\pi_y)$ be its listwise loss, given the training data $S = \{(\mathbf{x}^{(i)},\pi_y^{(i)}), i = 1,\cdots,n\}$, if $\forall f \in \mathcal{F}, (\mathbf{x},\pi_y) \in \mathcal{X}^m \times \mathcal{Y}, L_{\mathcal{A}}(f;\mathbf{x},\pi_y) \in [0,1]$, then with probability at least $1-\delta$, the following inequality holds,

$$\sup_{f\in\mathcal{F}}(R_{L_{\mathcal{A}}}(f) - \widehat{R}_{L_{\mathcal{A}}}(f;S)) \leq 2C_{\mathcal{A}}(\varphi)N(\varphi)\widehat{\mathcal{R}}(\mathcal{F}) + \sqrt{\frac{2\log\frac{2}{\delta}}{n}}, \tag{7.19}$$

where $\widehat{\mathcal{R}}(\mathcal{F})$ is the Rademacher average of the scoring function class (for the linear scoring function, we have $\widehat{\mathcal{R}}(\mathcal{F}) \leq \frac{2BM}{\sqrt{n}}$); $N(\varphi) = \sup_{x\in[-BM,BM]} \varphi'(x)$ measures the smoothness of the transformation function φ; $C_{\mathcal{A}}(\varphi)$ is an algorithm-dependent factor.

Table 7.1 $N(\varphi)$ and $C_{\mathcal{A}}(\varphi)$ for Listwise Ranking Algorithms

φ	$N(\varphi)$	$C_{ListMLE}(\varphi)$	$C_{ListNet}(\varphi)$
φ_L	a	$\frac{2}{(b-aBM)(\log m+\log\frac{b+aBM}{b-aBM})}$	$\frac{2m!}{(b-aBM)(\log m+\log\frac{b+aBM}{b-aBM})}$
φ_E	ae^{aBM}	$\frac{2e^{aBM}}{\log m+2aBM}$	$\frac{2m!e^{aBM}}{\log m+2aBM}$
φ_S	$\frac{ae^{aBM}}{(1+e^{-aBM})^2}$	$\frac{2(1+e^{aBM})}{\log m+aBM}$	$\frac{2m!(1+e^{aBM})}{\log m+aBM}$

The expressions of $N(\varphi)$ and $C_{\mathcal{A}}(\varphi)$ for ListNet and ListMLE, with respect to three representative transformation functions[5] are listed in Table 7.1.

From Theorem 7.5, one can see that when the number of training queries n approaches infinity, the query-level generalization bound will converge to zero at a rate of $O(\frac{1}{\sqrt{n}})$. Furthermore, by comparing the query-level generalization bound for different listwise ranking algorithms, and with regards to different transformation functions, one can have the following observations.

- The query-level generalization bound for ListMLE is much tighter than that for ListNet, especially when m, the length of the list, is large.
- The query-level generalization bound for ListMLE decreases monotonously, while that of ListNet increases monotonously, with respect to m.
- The linear transformation function is the best choice in terms of the query-level generalization bound in most cases.

[5] The three transformation functions are

◇ Linear Functions: $\varphi_L(x) = ax + b, x \in [-BM, BM]$.
◇ Exponential Functions: $\varphi_E(x) = e^{ax}, x \in [-BM, BM]$.
◇ Sigmoid Functions: $\varphi_S(x) = \frac{1}{1+e^{-ax}}, x \in [-BM, BM]$.

7.4 Discussions

Since learning to rank is still a new research area, many theoretical issues are still left open and significant efforts are still needed to make it a legitimate branch of machine learning.

Actually, the full story of the statistical ranking theory should consist of two parts: the statistical consistency and the generalization ability of learning-to-rank methods. We have briefly reviewed the recent advances on generalization analysis in terms of a given surrogate loss function L, i.e., when the number of training queries approaches infinity, whether the empirical risk defined with the surrogate loss L can converge to the expected risk, which is also defined with the same surrogate loss L. The statistical consistency further discusses whether the minimization of the expected risk defined with the surrogate loss L can lead to the minimization of the expected risk defined with the true loss (which is not clearly defined for ranking yet).

Although there have been some works discussing the consistency for ranking in the literature, the problem has not been well solved yet. For example, the consistency for bipartite ranking was studied in [28], which is, however, more like a classification and much simpler than real ranking problems. For another example, in [27], the consistency for pairwise ranking was discussed, but no necessary discussions were provided on how to extend this result to listwise ranking. In [33], the consistency issue with respect to DCG was studied. However, the authors only discussed the regression-based loss function and thus is insufficient to explain the majority of learning-to-rank algorithms that are based on pairwise or listwise loss functions. In [129], the consistency of listwise ranking was investigated. However, the true loss is defined as a 0-1 loss at the permutation level, which is clearly not in accordance with our understanding of the loss in ranking (e.g., errors at different positions should lead to different losses). Therefore, all the aforementioned works cannot be used to well explain the consistency issue in learning to rank for IR.

To move forward, it is very important to define a reasonable true loss for ranking. One choice is to directly use the IR evaluation measures to define the true loss. However, there are still several problems with it. For example, there are many different IR evaluation measures, and it is not clear which one should be regarded as the true loss. Since these measures are not necessarily consistent with each other, no matter which one we choose as the true loss, it looks not that "true" after all. Second, according to the practice of classification, the true loss is usually not what we use to evaluate the performance of a learning algorithm. For example, while the 0-1 loss is used as the true loss of classification, precision, recall and F-scores are widely used as evaluation measures. Based on these discussions, it is not clear yet how to define the true loss for ranking. But it is relatively easier to get some principles of defining such a true loss. For example, the true loss for ranking should be listwise (since many IR evaluation measures are listwise), should consider the position information, and should reflect the behaviors of the users when they check the search result page from top to bottom. This could be future work of the theoretical study on learning to rank.

8

Summary and Outlook

In this tutorial, we have mainly introduced three approaches to learning to rank. The first is called the pointwise approach, which reduces ranking to regression, classification, or ordinal regression on each single document. The second is called the pairwise approach, which basically formulates ranking as a pairwise classification problem. The third is called the listwise approach, which regards ranking as a new problem, and tries either to directly optimize the non-smooth IR evaluation measures, or to minimize a listwise ranking loss. We have introduced the representative algorithms of these three approaches, discussed their advantages and problems, and validated their empirical effectiveness on the LETOR benchmark dataset. In addition, we have also introduced the statistical ranking theory, and analyzed the query-level generalization ability of several learning-to-rank methods.

As a summary, we plot the representative algorithms introduced in this tutorial in Figure 8.1. From the figure, one can find that learning to rank for IR has become hotter and hotter in recent years, and more and more attention has been paid to the listwise approach.

Note that this tutorial is by no means a complete review of the area of learning to rank, especially considering that this is still a de-

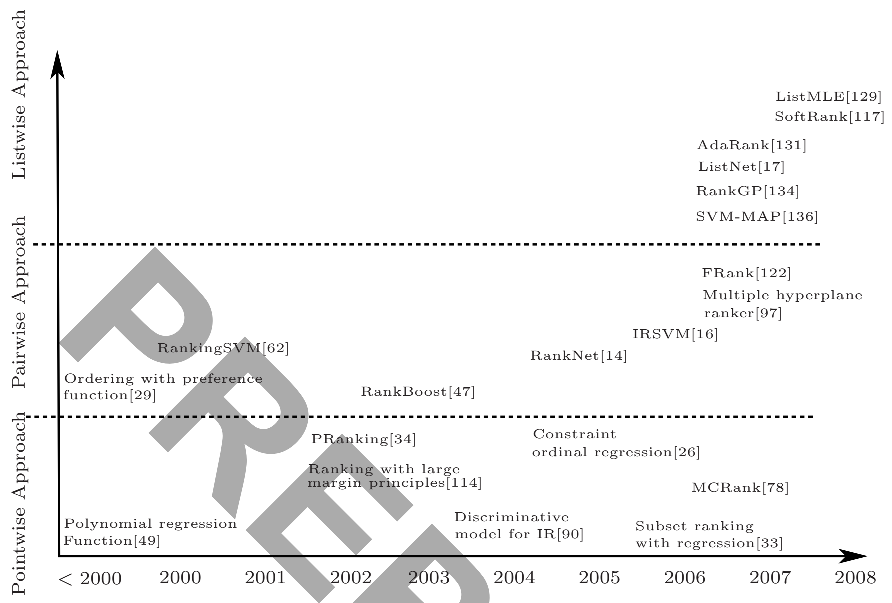

Fig. 8.1 Learning to Rank Algorithms

veloping research area. There are still many other works that have not been covered, some of which cannot be easily categorized into the three approaches. These works have covered the following sub-topics.

- *Ground truth mining* [68] [105] [3], which targets automatically mining ground truth labels for learning to rank, mainly from click-through logs of search engines.
- *Feature engineering* [51], which includes feature selection, dimensionality reduction, and effective feature learning.
- *Query-dependent ranking* [52] [71], which adopts different ranking models for different types of queries, based on either hard query type classification or soft nearest neighbor based approach.
- *Supervised rank aggregation* [80], which learns the ranking model not to combine features, but to aggregate candidate ranked lists.
- *Semi-supervised / active ranking* [135] [106] [40] [5] [67], which leverages the large number of unlabeled queries and

documents to improve the performance of ranking model learning.

- *Relational / global ranking* [38] [102] [103] [107] [137], which does not only make use of the scoring function for ranking, but also considers the inter-relationship between documents to define more complex ranking models.
- *Others learning-to-rank algorithms* [127] [144] [32] [15] [93] [109] [143] [19] [145] that are based on association rules, decision systems, and other technologies; *other theoretical analysis on ranking* [50]; and *applications of learning-to-rank methods* [128] [87].

At the end of this tutorial, let us come back to the questions we raised in the introduction, and give their possible answers.

(1) *To what respect are these learning-to-rank algorithms similar and in which aspects do they differ? What are the strengths and weaknesses of each algorithm?*

The answer can be found by the algorithm description and categorization. Basically all the algorithms belonging to the pointwise approach reduce ranking to either regression, classification, or ordinal regression. Almost all algorithms belonging to the pairwise approach reduce ranking to a pairwise classification. The advantage of these two approaches is that many existing theories and tools in machine learning can be directly applied. However, distinct properties of ranking have not been considered in such formulations. For example, most IR evaluation measures are query-level and position-based. However, neither the query information nor the position information is visible to the loss functions of these two approaches. The listwise approach instead treats ranking as a new problem, and defines specific algorithms for it. It can better leverage the concept of *query*, and consider the position information in the ranking results when training the ranking model. The problem with the listwise approach is that it is in general more complex than the pointwise and pairwise approaches. Furthermore, a new theoretical foundation is needed to explain the behaviors of listwise ranking algorithms.

According the analysis in Chapter 5, the loss functions of most learning-to-rank methods, no matter pointwise, pairwise, or listwise, are upper bounds of (1−NDCG). Therefore, the minimization of these loss functions can lead to the minimization of (1−NDCG), or the maximization of NDCG.

(2) *Empirically speaking, which of those many learning-to-rank algorithms perform the best? What kind of datasets can be used to make fair comparison among different learning-to-rank algorithms?*

According to the discussions in Chapter 6, the LETOR benchmark dataset has been widely used in the learning-to-rank works recently. Due to the standard data collection, feature representation, dataset partitioning, and evaluation tools in LETOR, it is possible to perform fair comparisons among different learning-to-rank methods. Empirical studies on LETOR have shown that the listwise ranking algorithms seem to have certain advantages over other algorithms, especially for top positions of the ranking result, and the pairwise ranking algorithms seem to outperform the pointwise algorithms. These results are in accordance with the discussions in this tutorial. However, as pointed out in Chapter 6, these experimental results are still primal and by carefully tuning the optimization process, the performance of every algorithm can be further improved.

(3) *Theoretically speaking, is ranking a new machine learning problem, or can it be simply reduced to existing machine learning problems? What are the unique theoretical issues for ranking that should be investigated?*

According to the discussions in Chapter 7, we can clearly see that it is better to regard ranking as a new machine learning problem, rather than reducing it to existing problems. Unique properties of ranking for IR as compared to classification and regression lie in that the evaluation of a ranking model is performed at the query level and is position based. Therefore, the risks should be defined at the query level as well, and a query-level theoretical framework is desired for conducting analyses on the learning-to-rank methods. Furthermore, the "true loss" for ranking should consider the position information in the ranking result, but not as simple as the $0-1$ loss in classification.

(4) *Are there many remaining issues regarding learning to rank we should study in the future? What are they?*

As mentioned previously, there are still many open problems. We list some of them as follows, as the future work on learning to rank.

Learning from Logs

- The existing benchmark datasets are all of relatively small scales. From a machine learning point of view, hundreds of queries cannot reasonably guarantee the effectiveness of a learning-to-rank algorithm. Developing realistically sized datasets is very important. Click-through log mining is one of the possible approaches to construct large-scale training data. Several works have been done along this direction [68] [105] [3], however they also have certain limitations. Basically these works have tried to map the click-through logs to pairwise preferences or multiple ordered categories. However, this process is not always necessary (and sometimes even not reasonable). As we know, the multiple ordered categories are designed for human labelers, which cannot cover all the rich information contained in the click-through logs, e.g., the user sessions, the frequency of clicking a certain document, the frequency of a certain click pattern, and the diversity in the intentions of different users. If converting the log data to multiple ordered categories or pairwise preferences, the information will be missing. Therefore, it is meaningful to reconsider the problem, and probably change the learning algorithms to adapt to the log data. For example, one can regard the click-through logs (without mining) as the ground truth, and define the loss function based on its likelihood.

Feature Engineering

- After one extracts a set of features for each document, it seems the learning-to-rank problem becomes a standard prediction task. However, one should notice that ranking is deeply rooted in IR, so the eventual goal of learning to rank is not only to develop a set of new algorithms and theories, but also to substantially improve the ranking performance. For this purpose, feature engineering cannot be overlooked. It is a killer aspect whether we can encode the knowledge on IR accumulated in the past half a century in the extracted features. Currently, these kinds of works have not been given enough attention to. In the future, we should even study the possibility of learning effective features.

Advanced Ranking Methods

- In most existing learning-to-rank algorithms, a scoring function is used for sake of simplicity and efficiency. However, sometimes such a simplification cannot handle complex ranking problems. People have made some attempts on leveraging the inter-relationships between objects and some relational (global) ranking algorithms [102] [103] [107] have been proposed. However, this is not yet the most straightforward way of defining the hypothesis for ranking, especially for the listwise approach. Since the output space of the listwise approach is composed of permutations of documents, the ranking hypothesis should better directly output permutations of the documents, rather than output scores for each of the individual documents. In this regard, defining the ranking hypothesis as a multi-variate function directly on permutations could be a future research topic. Note that the task is challenging because permutation-based ranking function can be very complex due to the extremely large number of permutations, but we think it is worthy and also possible to find efficient algorithms to deal with this situation.

- Motivated by the recent advances in machine learning, one should expect corresponding progresses in learning to rank. For example, transfer ranking, multi-task ranking, semi-supervised ranking, and active ranking can all be promising future research topics. However, when performing such research, one should pay attention to the unique properties of ranking, and make necessary adaption when introducing these concepts and technologies. For example, we have mentioned some previous works on semi-supervised ranking [40] [5] [67]. Basically they simply borrow some concepts and algorithms in semi-supervised classification. However, the validity of doing so needs further checking. For instance, since similarity is essential to classification ("similar documents should have the same class label"), it looks very natural and reasonable to propagate labels cross similar documents. However, in ranking, similarity does not play the same central role. It seems that preference is more fundamental than similarity. Then it is questionable to still conduct similarity based label propagation for semi-supervised ranking. Furthermore, in classification, if we do not have class labels, we know nothing about the objects. However, in ranking, even if we do not have ground truth labels, we still have several very strong rankers, such as the BM25 and language models for IR, which can give us a relatively reasonable guess on which document should be ranked higher. Therefore, it is reasonable to assume that we have some knowledge about the unlabeled data. If we can incorporate such knowledge into the semi-supervised ranking process, we may have the chance to do a better job.
- As introduced in this tutorial, most efforts on learning to rank have been given to discriminative learning. However, as we notice, generative learning is also a very important branch of machine learning. There is no reason that generative learning cannot be used in ranking. This could be a

promising research direction of learning to rank, from both algorithmic and theoretical points of view.

Ranking Theory

- For existing algorithms directly optimizing IR evaluation measures, although a theory has been proposed to quantify the “directness” of some algorithms in the large sample limit, it is not clear how such algorithm will perform on the test data when the training data are limited. This could also be a future research direction.
- As compared to the efforts on algorithms, the theoretical work on ranking is not yet sufficient. For example, it is still unclear about the “true loss” in ranking, and in addition to the generalization analysis, statistical consistency and fast convergence rate have not been well studied. Furthermore, some fundamental questions have not been well answered with regards to the ranking theory, such as the complexity of the function class in ranking. This could also be an important research direction for learning to rank.

Overall, this tutorial is just a stage-wise summary of this hot research field. Given the fast development of learning to rank, we can foresee that many new algorithms and theories will appear in the future. We hope that this tutorial can motivate more people to work on learning to rank, so as to make this research direction more impactful in both the information retrieval and machine learning communities.

References

[1] S. Agarwal, T. Graepel, R. Herbrich, S. Har-Peled, and D. Roth. Generalization bounds for the area under the roc curve. *Journal of Machine Learning*, 6:393–425, 2005.

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

[3] E. Agichtein, E. Brill, S. T. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR 2006*, pages 3–10, 2006.

[4] H. Almeida, M. Goncalves, M. Cristo, and P. Calado. A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *SIGIR 2007*, pages 399–406, 2007.

[5] M.-R. Amini, T.-V. Truong, and C. Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *SIGIR 2008*, pages 99–106, 2008.

[6] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *NIPS 2003*, pages 561–568, 2003.

[7] J. A. Aslam and M. Montague. Models for metasearch. In *SIGIR 2001*, pages 276–284, 2001.

[8] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.

[9] B. Bartell, G. W. Cottrell, and R. Belew. Learning to retrieve information. In *SCC 1995*, 1995.

[10] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities risk bounds and structural results. *Journal of Machine Learning*, pages 463–482, 2002.

[11] O. Bousquet, S. Boucheron, and G. Lugosi. Introduction to statistical learning theory. In *Advanced Lectures on Machine Learning*, pages 169–207. Springer Berlin / Heidelberg, 2004.

[12] O. Bousquet and A. Elisseeff. Stability and generalization. *The Journal of Machine Learning Research*, 2:449–526, 2002.

[13] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS 2006*, pages 395–402, 2006.

[14] C. J. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML 2005*, pages 89–96, 2005.

[15] G. Cao, J. Nie, L. Si, and J. Bai. Learning to rank documents for ad-hoc retrieval with regularized models. In *SIGIR 2007 workshop on Learning to Rank for Information Retrieval*, 2007.

[16] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *SIGIR 2006*, pages 186–193, 2006.

[17] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML 2007*, pages 129–136, 2007.

[18] B. Carterette, V. Pavlu, E. Kanoulas, J. A. Aslam, and J. Allan. Evaluation over thousands of queries. In *SIGIR 2008*, pages 651–658, 2008.

[19] V. R. Carvalho, J. L. Elsas, W. W. Cohen, and J. G. Carbonell. A meta-learning approach for robust rank learning. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.

[20] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *SIGKDD 2008*, pages 88–96, 2008.

[21] O. Chapelle, Q. Le, and A. Smola. Large margin optimization of ranking measures. In *NIPS workshop on Machine Learning for Web Search 2007*, 2007.

[22] W. Chen, Y. Lan, T.-Y. Liu, and H. Li. A unified view on loss functions in learning to rank. Technical report, Microsoft Research, MSR-TR-2009-39, 2009.

[23] P. Chirita, J. Diederich, and W. Nejdl. MailRank: using ranking for spam detection. In *CIKM 2005*, pages 373–380. ACM New York, NY, USA, 2005.

[24] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:1019–1041, 2005.

[25] W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *ICML 2005*, pages 137–144, 2005.

[26] W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *ICML 2005*, pages 145–152, 2005.

[27] S. Clemencon, G. Lugosi, and N. Vayatis. Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, 36(2):844–874, 2008.

[28] S. Clemencon and N. Vayatis. Ranking the best instances. *Journal of Machine Learning Research*, 8(Dec):2671–2699, 2007.

[29] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. In *NIPS 1998*, volume 10, pages 243–270, 1998.

[30] M. Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *ACL 2002*, pages 07–12, 2002.

[31] W. S. Cooper, F. C. Gey, and D. P. Dabney. Probabilistic retrieval based on staged logistic regression. In *SIGIR 1992*, pages 198–210, 1992.
[32] C. Cortes, M. Mohri, and etc. Magnitude-preserving ranking algorithms. In *ICML 2007*, pages 169–176, 2007.
[33] D. Cossock and T. Zhang. Subset ranking using regression. In *COLT 2006*, pages 605–619, 2006.
[34] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS 2002*, pages 641–647, 2002.
[35] N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the trec 2003 web track. In *TREC 2003*, pages 78–92, 2003.
[36] K. Dave, S. Lawrence, and D. Pennock. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *WWW 2003*, pages 519–528. ACM Press New York, NY, USA, 2003.
[37] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.
[38] F. Diaz. Regularizing query-based retrieval scores. *Information Retrieval*, 10(6):531–562, 2007.
[39] H. Drucker, B. Shahrary, and D. C. Gibbon. Support vector machines: Relevance feedback and information retrieval. *Information Processing & Management*, 38(3):305–323, 2002.
[40] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR 2008*, pages 251–258, 2008.
[41] W. Fan, E. A. Fox, P. Pathak, and H. Wu. The effects of fitness functions on genetic programming based ranking discovery for web search. *Journal of American Society for Information Science and Technology*, 55(7):628–636, 2004.
[42] W. Fan, M. Gordon, and P. Pathak. Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):523–527, 2004.
[43] W. Fan, M. Gordon, and P. Pathak. A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing and Management*, 40(4):587–602, 2004.
[44] W. Fan, M. Gordon, and P. Pathak. Genetic programming-based discovery of ranking functions for effective web search. *Journal of Management of Information Systems*, 21(4):37–56, 2005.
[45] W. Fan, M. Gordon, and P. Pathak. On linear mixture of expert approaches to information retrieval. *Decision Support System*, 42(2):975–987, 2006.
[46] W. Fan, M. D. Gordon, W. Xi, and E. A. Fox. Ranking function optimization for effective web search by genetic programming: an empirical study. In *HICSS 2004*, page 40105, 2004.
[47] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[48] Y. Freund and R. E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1995.
[49] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems*, 7(3):183–204, 1989.
[50] G. Fung, R. Rosales, and B. Krishnapuram. Learning rankings via convex hull separation. In *NIPS 2005 workshop on Learning to Rank*, 2005.
[51] X.-B. Geng, T.-Y. Liu, and T. Qin. Feature selection for ranking. In *SIGIR 2007*, pages 407–414, 2007.
[52] X.-B. Geng, T.-Y. Liu, T. Qin, H. Li, and H.-Y. Shum. Query-dependent ranking using k-nearest neighbor. In *SIGIR 2008*, pages 115–122, 2008.
[53] F. C. Gey. Inferring probability of relevance using the method of logistic regression. In *SIGIR 1994*, pages 222–231, 1994.
[54] S. Guiasu and A. Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1), 1985.
[55] J. Guiver and E. Snelson. Learning to rank with softrank and Gaussian processes. In *SIGIR 2008*, pages 259–266, 2008.
[56] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB 2004*, pages 576–587. VLDB Endowment, 2004.
[57] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB 2004*, pages 576–587, 2004.
[58] E. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *ICML 2003*, volume 20, pages 250–257, 2003.
[59] E. F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *ICML 2003*, pages 250–257, 2003.
[60] B. He and I. Ounis. A study of parameter tuning for term frequency normalization. In *CIKM 2003*, pages 10–16, 2003.
[61] Y. He and T.-Y. Liu. Are algorithms directly optimizing ir measures really direct? Technical report, Microsoft Research, MSR-TR-2008-154, 2008.
[62] R. Herbrich, K. Obermayer, and T. Graepel. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132, 2000.
[63] R. Herbrich, T. G. T., and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, 2001.
[64] W. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *SIGIR 1994*, pages 192–201, 1994.
[65] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR 2000*, pages 41–48, 2000.
[66] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.
[67] R. Jin, H. Valizadegan, and H. Li. Ranking refinement and its application to information retrieval. In *WWW 2008*, pages 397–406, 2008.
[68] T. Joachims. Optimizing search engines using clickthrough data. In *KDD 2002*, pages 133–142, 2002.

[69] T. Joachims. Evaluating retrieval performance using clickthrough data. *Text Mining*, pages 79–96, 2003.
[70] T. Joachims. A support vector method for multivariate performance measures. In *ICML 2005*, pages 377–384, 2005.
[71] I. Kang and G. Kim. Query type classification for web document retrieval. In *SIGIR 2003*, pages 64–71, 2003.
[72] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of ACM*, 46(5):604–632, 1999.
[73] S. Kramer, G. Widmer, B. Pfahringer, and M. D. Groeve. Prediction of ordinal classes using regression trees. *Funfamenta Informaticae*, 34:1–15, 2000.
[74] J. Lafferty and C. Zhai. Document language models, query models and risk minimization for information retrieval. In *SIGIR 2001*, pages 111–119, 2001.
[75] Y. Lan and T.-Y. Liu. Generalization analysis of listwise learning-to-rank algorithms. In *ICML 2009*, 2009.
[76] Y. Lan, T.-Y. Liu, T. Qin, Z. Ma, and H. Li. Query-level stability and generalization in learning to rank. In *ICML 2008*, pages 512–519, 2008.
[77] F. Li and Y. Yang. A loss function analysis for classification methods in text categorization. In *ICML 2003*, pages 472–479, 2003.
[78] P. Li, C. Burges, and Q. Wu. McRank: Learning to rank using multiple classification and gradient boosting. In *NIPS 2007*, pages 845–852, 2007.
[79] T.-Y. Liu, J. Xu, T. Qin, W.-Y. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR '07 Workshop on learning to rank for information retrieval*, 2007.
[80] Y. Liu, T.-Y. Liu, T. Qin, Z. Ma, and H. Li. Supervised rank aggregation. In *WWW 2007*, pages 481–490, 2007.
[81] R. D. Luce. *Individual Choice Behavior*. Wiley, New York, 1959.
[82] C. L. Mallows. Non-null ranking models. *Biometrika*, 44:114–130, 1975.
[83] M. E. Maron and J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the ACM*, 7(3):216–244, 1960.
[84] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR 2006*, pages 437–444, 2006.
[85] D. A. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *SIGIR 2005*, pages 472–479, 2005.
[86] D. A. Metzler, W. B. Croft, and A. McCallum. Direct maximization of rank-based metrics for information retrieval. In *CIIR Technical Report*, 2005.
[87] D. A. Metzler and T. Kanungo. Machine learned sentence selection strategies for query-biased summarization. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.
[88] T. Minka and S. Robertson. Selection bias in the LETOR datasets. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.
[89] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
[90] R. Nallapati. Discriminative models for information retrieval. In *SIGIR 2004*, pages 64–71, 2004.
[91] L. Nie, B. D. Davison, and X. Qi. Topical link analysis for web search. In *SIGIR 2006*, pages 91–98, 2006.

[92] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[93] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and T. Salakoski. Learning to rank with pairwise regularized least-squares. In *SIGIR 2007 workshop on Learning to Rank for Information Retrieval*, 2007.

[94] B. Pang and L. Lee. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL 2005*, pages 115–124. Association for Computational Linguistics Morristown, NJ, USA, 2005.

[95] R. L. Plackett. The analysis of permutations. *Applied Statistics*, 24(2):193–202, 1975.

[96] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR 1998*, pages 275–281, 1998.

[97] T. Qin, T.-Y. Liu, W. Lai, X.-D. Zhang, D.-S. Wang, and H. Li. Ranking with multiple hyperplanes. In *SIGIR 2007*, pages 279–286, 2007.

[98] T. Qin, T.-Y. Liu, and H. Li. A general approximation framework for direct optimization of information retrieval measures. Technical report, Microsoft Research, MSR-TR-2008-164, 2008.

[99] T. Qin, T.-Y. Liu, M.-F. Tsai, X.-D. Zhang, and H. Li. Learning to search web pages with query-level loss functions. Technical report, Microsoft Research, MSR-TR-2006-156, 2006.

[100] T. Qin, T.-Y. Liu, J. Xu, and H. Li. How to make LETOR more useful and reliable. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.

[101] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. In *SIGIR 2005*, pages 408–415, 2005.

[102] T. Qin, T.-Y. Liu, X.-D. Zhang, D. Wang, and H. Li. Learning to rank relational objects and its application to web search. In *WWW 2008*, pages 407–416, 2008.

[103] T. Qin, T.-Y. Liu, X.-D. Zhang, D.-S. Wang, and H. Li. Global ranking using continuous conditional random fields. In *NIPS 2008*, pages 1281–1288, 2008.

[104] T. Qin, T.-Y. L. X.-D. Zhang, M.-F. Tsai, D.-S. Wang, and H. Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2007.

[105] F. Radlinski and T. Joachims. Query chain: Learning to rank from implicit feedback. In *KDD 2005*, pages 239–248, 2005.

[106] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *KDD 2007*, 2007.

[107] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML 2008*, pages 784–791, 2008.

[108] S. Rajaram and S. Agarwal. Generalization bounds for k-partite ranking. In *NIPS 2005 WorkShop on Learning to Rank*, 2005.

[109] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli. Learning to rank by a neural-based sorting algorithm. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.

[110] S. Robertson and H. Zaragoza. On rank-based effectiveness measures and optimization. *Information Retrieval*, 10(3):321–339, 2007.

[111] S. E. Robertson. Overview of the okapi projects. *Journal of Documentation*, 53(1):3–7, 1997.

[112] J. J. Rochhio. Relevance feedback in information retrieval. *The SMART Retrieval System - Experiments in Automatic Document Processing*, pages 313–323, 1971.

[113] A. Shakery and C. Zhai. A probabilistic relevance propagation model for hypertext retrieval. In *CIKM 2006*, pages 550–558, 2006.

[114] A. Shashua and A. Levin. Ranking with large margin principles: Two approaches. In *NIPS 2002*, pages 937–944, 2002.

[115] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[116] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[117] M. Talyor, J. Guiver, and etc. Softrank: Optimising non-smooth rank metrics. In *WSDM 2008*, pages 77–86, 2008.

[118] T. Tao and C. Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *SIGIR 2006*, pages 162–169, 2006.

[119] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In *SIGIR 2007*, pages 295–302, 2007.

[120] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. J. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM 2006*, pages 585–593, 2006.

[121] A. Trotman. Learning to rank. *Information Retrieval*, 8(3):359–381, 2005.

[122] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *SIGIR 2007*, pages 383–390, 2007.

[123] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output space. In *ICML 2004*, pages 104–111, 2004.

[124] N. Usunier, M.-R. Amini, and P. Gallinari. Generalization error bounds for classifiers trained with interdependent data. In *NIPS 2005*, pages 1369–1376, 2005.

[125] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[126] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[127] A. Veloso, H. M. de Almeida, M. A. Gon?alves, and W. M. Jr. Learning to rank at query-time using association rules. In *SIGIR 2008*, pages 267–274, 2008.

[128] W. Xi, J. Lind, and E. Brill. Learning effective ranking functions for newsgroup search. In *SIGIR 2004*, pages 394–401, 2004.

[129] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank - theorem and algorithm. In *ICML 2008*, pages 1192–1199, 2008.

[130] J. Xu, Y. Cao, H. Li, and M. Zhao. Ranking definitions with supervised learning methods. In *WWW 2005*, pages 811–819. ACM Press New York, NY, USA, 2005.

[131] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR 2007*, pages 391–398, 2007.

[132] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing IR evaluation measures in learning to rank. In *SIGIR 2008*, pages 107–114, 2008.

[133] G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *SIGIR 2005*, pages 186–193, 2005.

[134] J.-Y. Yeh, J.-Y. Lin, and etc. Learning to rank for information retrieval using genetic programming. In *SIGIR 2007 Workshop in Learning to Rank for Information Retrieval*, 2007.

[135] H. Yu. SVM selective sampling for ranking with application to data retrieval. In *KDD 2005*, pages 354–363, 2005.

[136] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR 2007*, pages 271–278, 2007.

[137] Y. Yue and T. Joachims. Predicting diverse subsets using structural SVM. In *ICML 2008*, pages 1224–1231, 2008.

[138] C. Zhai. Language models. *Foundations and Trends in Information Retrieval*, 2008.

[139] C. Zhai, W. W. Cohen, and J. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *SIGIR 2003*, pages 10–17, 2003.

[140] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM 2001*, pages 403–410, 2001.

[141] C. Zhai and J. Lafferty. A risk minimization framework for information retrieval. *Information Processing and Management*, 42(1):31–55, 2006.

[142] T. Zhang. Statistical analysis of some multi-category large margin classification methods. *Journal of Machine Learning Research*, 5:1225–1251, 2004.

[143] Z. Zheng, H. Zha, and G. Sun. Query-level learning to rank using isotonic regression. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.

[144] K. Zhou, G.-R. Xue, H. Zha, and Y. Yu. Learning to rank with ties. In *SIGIR 2008*, pages 275–282, 2008.

[145] O. Zoeter, M. Taylor, E. Snelson, J. Guiver, N. Craswell, and M. Szummer. A decision theoretic framework for ranking using implicit feedback. In *SIGIR 2008 workshop on Learning to Rank for Information Retrieval*, 2008.

Acknowledgements

The author would like to thank Yanyan Lan, Tao Qin, Jun Xu, and Hang Li for their helpful suggestions and comments on this tutorial. The author would also like to thank anonymous reviewers.