Noname manuscript No.
(will be inserted by the editor)

LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval

Tao Qin · Tie-Yan Liu · Jun Xu · Hang Li

Received: date / Accepted: date

Abstract LETOR is a benchmark collection for the research on learning to rank for information retrieval, released by Microsoft Research Asia. In this paper, we describe the details of the LETOR collection and show how it can be used in different kinds of researches. Specifically, we describe how the document corpora and query sets in LETOR are selected, how the documents are sampled, how the learning features and meta information are extracted, and how the datasets are partitioned for comprehensive evaluation. We then compare several state-of-the-art learning to rank algorithms on LETOR, report their ranking performances, and make discussions on the results. After that, we discuss possible new research topics that can be supported by LETOR, in addition to algorithm comparison. We hope that this paper can help people to gain deeper understanding of LETOR, and enable more interesting research projects on learning to rank and related topics.

Keywords Learning to rank · information retrieval · benchmark datasets · feature extraction

1 Introduction

Ranking is the central problem for many applications of information retrieval (IR). These include document retrieval [5], collaborative filtering [16], key term extraction

Tao Qin
Microsoft Research Asia
E-mail: taoqin@microsoft.com

Tie-Yan Liu
Microsoft Research Asia
E-mail: tyliu@microsoft.com

Jun Xu
Microsoft Research Asia
E-mail: junxu@microsoft.com

Hang Li
Microsoft Research Asia
E-mail: hangli@microsoft.com

[9], definition finding [46], important email routing [8], sentiment analysis [29], product rating [12], and anti web spam [15]. In the task of ranking, given a set of objects, we utilize a ranking model (function) to create a ranked list of the objects. The relative order of objects in the list may represent their degrees of relevance, preference, or importance, depending on applications. Among the aforementioned applications, document retrieval is by all means the most important one, and therefore we will take it as an example when performing the discussions in this paper.

Learning to rank, when applied to document retrieval, is a task as follows. Assume that there is a corpus of documents. In training, a number of queries are provided; each query is associated with a set of documents with relevance judgments; a ranking function is then created using the training data, such that the model can precisely predict the ranked lists in the training data. In retrieval (i.e., testing), given a new query, the ranking function is used to create a ranked list for the documents associated with the query. Since the learning to rank technology can successfully leverage multiple features for ranking, and can automatically learn the optimal way of combining these features, it has been gaining increasing attention in recent years. Many learning to rank methods have been proposed and applied to different IR applications.

To facilitate the research on learning to rank, an experimental platform is sorely needed, which contains indexed document corpora, selected queries for training and test, feature vectors extracted for each document, implementation of baseline algorithms, and standard evalaution tools. However, there was no such an environment and it largely blocked the advancement of the related research. Researchers had to use their own datasets (i.e., different document corpora, different query sets, different features, and/or different evaluation tools), and thus it was not possible to make meaningful comparison among different methods. This is in sharp contrast with several other fields where research has been significantly enhanced by the availabilities of benchmark collections, such as Reuters 21578 [1] and RCV1 [23] for text categorization, and UCI [1] for general classification. In order to accelerate the research on learning to rank, we decided to build the benchmark collection LETOR. The construction of such a collection is, however, not easy, because it requires rich knowledge on the domain and a lot of engineering efforts. Thanks to the contributions from many people, we were able to release LETOR and upgrade it for several times.

LETOR was constructed based on multiple data corpora and query sets, which have been widely used in IR. The documents in the corpora were sampled according to carefully designed strategies, and then features and metadata were extracted for each query-document pair. Additional information including hyperlink graph, similarity relationship, and sitemap was also included. The data was partitioned into five folds for cross validation, and standard evaluation tools were provided. In addition, the ranking performances of several state-of-the-art ranking methods were also provided, which can serve as baselines for newly developed methods.

LETOR has been widely used in the research community since its release. The first version of LETOR was released in April 2007 and used in the SIGIR 2007 workshop on learning to rank for information retrieval (`http://research.microsoft.com/users/LR4IR-2007/`). At the end of 2007, the second version of LETOR was released, which was later used in the SIGIR 2008 workshop on learning to rank for IR (`http://research.microsoft.com/users/LR4IR-2008/`). Based on the valuable feed-

[1] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

back and suggestions we collected, the third version of LETOR was released in December 2008. The focus of this paper is on the third version [2], LETOR 3.0.

The contributions of LETOR to the research community lie in the following aspects.

(1) *It eases the development of ranking algorithms.*
Researchers can focus on algorithm development, and do not need to worry about experimental setup (e.g., creating datasets and extracting features). In that sense, LETOR has greatly reduced the barrier of the research on learning to rank.
(2) *It makes the comparison among different learning to rank algorithms possible.*
The standard document corpora, query sets, features, and partitions in LETOR enable researchers to conduct comparative experiments. The inclusion of baselines in LETOR also greatly saves researchers' experimental efforts.
(3) *It offers opportunities for new research topics on learning to rank.*
In addition to algorithm comparison, LETOR can also be used to study the problems of ranking model construction, feature creation, feature selection, dependent ranking, and transfer/multitask ranking.

The remaining part of the paper is organized as follows. We introduce the problem of learning to rank for IR in Section 2. Section 3 gives a detailed description about LETOR. Section 4 reports the performances of several state-of-the-art learning to rank algorithms on LETOR. We then show how LETOR can be used to study other research topics beyond algorithm comparison in Section 5. Finally limitations of LETOR are discussed in Section 6 and concluding remarks are given in Section 7.

2 Learning to Rank for IR

There are two major approaches to tackle the ranking problems in IR: the learning to rank approach and the traditional non-learning approach such as BM25 [38] and language model [51].

The main difference between the two approaches lies in that the former can automatically learn the parameters of the ranking function using training data, while the latter usually determines the parameters heuristically. If a ranking model has only several parameters, heuristic tuning can be possible. However, if there are many parameters, it will become very difficult. As more and more evidences have been proved as useful for ranking, the traditional non-learning approach will face challenges in effectively using these evidences.

In contrast, the learning to rank approach can make good use of multiple evidences. Therefore learning to rank has been drawing broad attention in both machine learning and IR communities, and many learning to rank algorithms have been proposed recently. Roughly speaking, there are mainly three kinds of algorithms, namely, the pointwise approach [25,24], the pairwise approach [17,13,4,42,36,26], and the listwise approach [6,35,45,20,47,40,50,33,34,43,31].

The pointwise approach regards a single document as its input in learning and defines its loss function based on individual documents. According to different output spaces of the ranking function, the pointwise approach can be further categorized as regression based algorithms [25], classification based algorithms [25], and ordinal regression based algorithms [24].

[2] The latest version when the paper was submitted.

The pairwise approach takes document pairs as instances in learning, and formalizes the problem of learning to rank as that of pairwise classification. Specifically, in learning it collects or generates document pairs from the training data, with each document pair assigned a label representing the relative order of the two documents. It then trains a ranking model by using classification technologies. The uses of Support Vector Machines (SVM), Boosting, and Neural Network as the classification model lead to the methods of Ranking SVM [17], RankBoost [13], and RankNet [4]. Many other algorithms have also been proposed, such as FRank [42], multiple hyperplane ranker [36] and nested ranker [26].

The listwise approach takes document lists as instances in learning and the loss function is defined on that basis. Representative work includes ListNet [6], RankCosine [35], relational ranking [34], global ranking [33], and StructRank [20]. A sub branch of the listwise approach is usually referred to as the direct optimization of IR measures. Example algorithms include AdaRank [47], SoftRank [40], SVM-MAP [50], PermuRank [48], ApproxRank [31], and BoltzRank [43].

3 Creating LETOR Collection

In this section, we introduce the processes of creating the LETOR collection, including four main steps: selecting document corpora (together with query sets), sampling documents, extracting learning features and meta information, and finalizing datasets.

3.1 Selecting Document Corpora

In the LETOR collection, we selected two document corpora: the "Gov" corpus and the OHSUMED corpus. These two corpora were selected because (1) they were publicly available [44] and (2) they had been widely used by previous work on ranking in the literature of IR [5,11,10,38].

3.1.1 The "Gov" Corpus and Six Query Sets

In TREC 2003 and 2004, there is a special track for web IR, named the Web track [11, 10]. The tracks used the "Gov" corpus, which is based on a January, 2002 crawl of the "Gov" domain. There are about one million html documents in this corpus.

Three search tasks are defined 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 of good websites principally devoted to the topic. The focus is to return entry pages of good websites rather than the web pages containing relevant information, because entry pages provide a better overview of the websites. *Homepage finding* aims at returning the homepage of the query. *Named page finding* is about finding the page whose name is identical to the query. In principle, there is only one answer for homepage finding and named page finding. Many papers [36,47,32,49] have been published using the three tasks on the "Gov" corpus as the basis for evaluation.

The following example illustrates the differences among these three tasks [41]. Consider the query 'USGS', which is the acronym for the US Geological Survey.

Table 1 Number of queries in TREC Web track

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

- For topic distillation, the query means "find for me homepages of sites describing the US Geological Survey". Possible answers include `http://www.usgs.gov`, `http://water.usgs.gov`, `http://geography.usgs.gov`, `http://earthquake.usgs.gov`.
- For homepage finding, the query means "find for me the URL of the USGS homepage (`http://www.usgs.gov`). I have forgotten or do not know that URL, or I prefer typing 'usgs' to typing the full URL". The right answer is exactly `http://www.usgs.gov`.
- For named page finding , the query may mean find me the URL of a non-homepage e.g. searching for `http://www.usgs.gov/aboutusgs.html` with the query 'introduction to usgs'. The query is the name of the page in question (rather than e.g. words describing its topic).

The numbers of queries in these three tasks are shown in Table. 1. For simplicity, we use acronyms in following sections: TD2003 for the topic distillation query set in TREC2003, TD2004 for the topic distillation query set in TREC2004, NP2003 for the named page finding query set in TREC2003, NP2004 for the named page finding query set in TREC2004, HP2003 for the homepage finding query set in TREC2003, and HP2004 for the homepage finding query set in TREC2004.

3.1.2 The OHSUMED Corpus

The OHSUMED corpus [18] is a subset of MEDLINE, a database on medical publications. It consists of about 0.3 million records (out of over 7 million) from 270 medical journals during the period of 1987-1991. The 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 has been widely used in previous work [36, 47], in which each query describes 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 human annotators, on three levels: definitely relevant, partially relevant, and irrelevant. There are in total 16,140 query-document pairs with relevance judgments.

3.2 Sampling Documents

Due to the large scale of the corpora, it is not feasible to judge the relevance of all the documents to a given query. As a common practice in IR, given a query, only some "possibly" relevant documents are selected for judgment. For similar reasons, it is not necessary to extract feature vectors from all the documents in the corpora. A reasonable approach is to sample some "possibly" relevant documents, and then extract feature vectors from the corresponding query-document pairs. In this section, we introduce the sampling strategy used in the construction of LETOR.

For the "Gov" corpus, given a query, the annotators organized by the TREC committee labeled some relevant documents. For the remaining unlabeled documents, the TREC committee treated them as irrelevant in the evaluation process [11]. Following this practice, in LETOR, we also treated the unlabeled documents for a query as irrelevant. Specifically, following the suggestions in [30] and [27], we performed the document sampling in the following way. We first used the BM25 model to rank all the documents with respect to each query, and then selected the top 1000 documents for each query for feature extraction. Note that for some rare queries, less than 1000 documents can be retrieved. As a result, some queries have less than 1000 associated documents in LETOR.

Different from the "Gov" corpus in which unjudged documents are regarded as irrelevant, in OHSUMED, the judgments explicitly contain the category of "irrelevant" and the unjudged documents are ignored in evaluation [18]. Following this practice, we only sampled judged documents for feature extraction and ignored the unjudged documents. As a result, on average, a query has about 152 documents associated for feature extraction.

3.3 Extracting Learning Features

In IR, given a query, we would like to rank a set of documents according to their relevance and importance to the query. In learning to rank, each query-document pair is represented by a multi-dimensional feature vector, and each dimension of the vector is a feature indicating how relevant or important the document is with respect to the query. For example, the first element of the vector could be the BM25 score of the document with regards to the query; the second element could be the frequency of the query terms appearing in the document; and the third one could be the PageRank score of the document. In this section, we introduce how the features in LETOR were extracted.

The following principles were used in the feature extraction process of LETOR.

(1) To cover as many classical features in IR as possible.
(2) To reproduce as many features proposed in recent SIGIR papers as possible, which were used in the experiments on the OHSUMED corpus or the "Gov" corpus.
(3) To conform to the settings in the original documents or papers. If the authors suggested parameter tuning on a feature, parameter tuning was also conducted in the feature extraction of LETOR. Otherwise, the default parameters reported in the documents or papers were directly applied.

The feature file in LETOR is formatted in a matrix style. A sample file is shown in Figure 1. Each row represents a query-document pair. The first column is the relevance judgment of the query-document pair; the second column is the query ID, followed with the feature ID and feature values.

3.3.1 Extracting Features for the "Gov" Corpus

For the "Gov" corpus, we extracted 64 features in total for each query-document pair, as shown in Table 2. Note that the id of each feature in this table is the same as in the LETOR datasets. We categorized the features into three classes: Q-D means that the

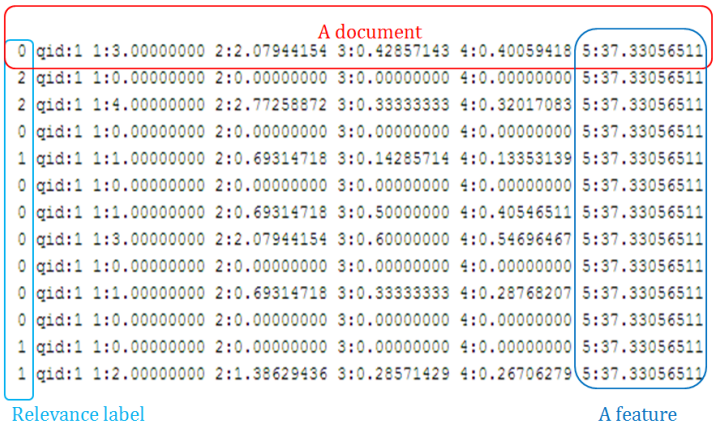

Fig. 1 Sample data from LETOR

feature is dependent on both the query and the document, D means that the feature only depends on the document, and Q means that the feature only depends on the query. Here we would like to point out that a linear ranking function cannot make use of the class-Q features, since these features are the same for all the documents under a query.

Table 2: Learning Features for the "Gov" Corpus

ID	Feature Description	Category
1	$\sum_{q_i \in q \cap d} c(q_i, d)$ in body	Q-D
2	$\sum_{q_i \in q \cap d} c(q_i, d)$ in anchor	Q-D
3	$\sum_{q_i \in q \cap d} c(q_i, d)$ in title	Q-D
4	$\sum_{q_i \in q \cap d} c(q_i, d)$ in URL	Q-D
5	$\sum_{q_i \in q \cap d} c(q_i, d)$ in whole document	Q-D
6	$\sum_{q_i \in q} idf(q_i)$ in body	Q
7	$\sum_{q_i \in q} idf(q_i)$ in anchor	Q
8	$\sum_{q_i \in q} idf(q_i)$ in title	Q
9	$\sum_{q_i \in q} idf(q_i)$ in URL	Q
10	$\sum_{q_i \in q} idf(q_i)$ in whole document	Q
11	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot idf(q_i)$ in body	Q-D
12	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot idf(q_i)$ in anchor	Q-D
13	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot idf(q_i)$ in title	Q-D
14	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot idf(q_i)$ in URL	Q-D
15	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot idf(q_i)$ in whole document	Q-D
16	$\|d\|$ of body	D
17	$\|d\|$ of anchor	D
18	$\|d\|$ of title	D
19	$\|d\|$ of URL	D
20	$\|d\|$ of whole document	D

21	BM25 of body	Q-D
22	BM25 of anchor	Q-D
23	BM25 of title	Q-D
24	BM25 of URL	Q-D
25	BM25 of whole document	Q-D
26	LMIR.ABS of body	Q-D
27	LMIR.ABS of anchor	Q-D
28	LMIR.ABS of title	Q-D
29	LMIR.ABS of URL	Q-D
30	LMIR.ABS of whole document	Q-D
31	LMIR.DIR of body	Q-D
32	LMIR.DIR of anchor	Q-D
33	LMIR.DIR of title	Q-D
34	LMIR.DIR of URL	Q-D
35	LMIR.DIR of whole document	Q-D
36	LMIR.JM of body	Q-D
37	LMIR.JM of anchor	Q-D
38	LMIR.JM of title	Q-D
39	LMIR.JM of URL	Q-D
40	LMIR.JM of whole document	Q-D
41	Sitemap based term propagation	Q-D
42	Sitemap based score propagation	Q-D
43	Hyperlink based score propagation: weighted in-link	Q-D
44	Hyperlink based score propagation: weighted out-link	Q-D
45	Hyperlink based score propagation: uniform out-link	Q-D
46	Hyperlink based propagation: weighted in-link	Q-D
47	Hyperlink based feature propagation: weighted out-link	Q-D
48	Hyperlink based feature propagation: uniform out-link	Q-D
49	HITS authority	Q-D
50	HITS hub	Q-D
51	PageRank	D
52	HostRank	D
53	Topical PageRank	Q-D
54	Topical HITS authority	Q-D
55	Topical HITS hub	Q-D
56	Inlink number	D
57	Outlink number	D
58	Number of slash in URL	D
59	Length of URL	D
60	Number of child page	D

61	BM25 of extracted title	Q-D
62	LMIR.ABS of extracted title	Q-D
63	LMIR.DIR of extracted title	Q-D
64	LMIR.JM of extracted title	Q-D

Some details about these features are listed as below.

(1) We considered five streams/fields [37] of a document: body, anchor, title, URL, and their union.
(2) q represents a query, which contains terms $q_1, ..., q_t$. In the two corpora, a query is expressed by two parts: the title of the query and the description of the query. Here we only considered the words appearing in the title of the query as query terms for feature extraction.
(3) $c(q_i, d)$ denotes the number of occurrences of query term q_i in document d. Note that while talking about a stream (e.g. title), $c(q_i, d)$ means the number of occurrences of q_i in the specific stream (e.g., title) of document d.
(4) Inverse document frequency (IDF) of query term q_i was computed as follows,

$$idf(q_i) = \log \frac{|C| - df(q_i) + 0.5}{df(q_i) + 0.5}, \tag{1}$$

where document frequency $df(q_i)$ is the number of documents containing q_i in a stream, and $|C|$ is the total number of documents in the document collection. Note that IDF is document-independent, and all the documents associated with the same query has the same IDF value.
(5) $|d|$ denotes the length (i.e., the number of words) of document d. When considering a specific stream, $|d|$ means the length of the stream. For example, $|d|$ of body means the length of the body of document d.
(6) The BM25 score of a document d was computed as follows,

$$BM25(d, q) = \sum_{q_i \in q} \frac{idf(q_i) \cdot c(q_i, d) \cdot (k_1 + 1)}{c(q_i, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})} \cdot \frac{(k_3 + 1)c(q_i, q)}{k_3 + c(q_i, q)}, \tag{2}$$

where $avgdl$ denotes the average document length in the entire document corpus. k_1, k_3 and b are free parameters. In feature 21-25 and 61, we set $k_1 = 2.5$, $k_3 = 0$ and $b = 0.8$. The $avgdl$ for each stream can be obtained from meta information, as described in the next sub section.
(7) For the language model features (26-40, 62-64), the implementation and the suggested parameters given in [51] were used. For the LMIR.ABS features, we set parameter $\delta = 0.7$; for LMIR.DIR features, we set parameter $\mu = 2000$; for LMIR.JM features, we set parameter $\lambda = 0.1$.
(8) For sitemap-based propagation features (41-42), we set the propagation rate $\alpha = 0.9$ according to [32]. For hyperlink based propagation features (43-48), we set the propagation rate $\alpha = 0.85$ according to [39].
(9) Because the original PageRank score of a document is usually very small, we amplified it by 10^5. We also scaled up the original HostRank score by 10^3.
(10) For topical PageRank and topical HITS, the same twelve categories as in [28] were used.

(11) To count the number of children of a web page (for feature 60), we simply used the URLs of documents to recover parent-child relationship between any two web pages.
(12) Extracted title was extracted from the content of a html document [19], and it is sometimes more meaningful than the original title of the html document.

3.3.2 Extracting Features for the OHSUMED Corpus

For the OHSUMED corpus, there are in total 45 features extracted from the steams/fields of title, abstract, and 'title + abstract', as shown in Table 3. Note that the id of each feature in this table is the same as in the LETOR datasets. We categorized the features into two classes: Q-D means that the feature depends on both the query and the document, and Q means that the feature only depends on the query.

Some details about these features are listed as below.

(1) For the language model features (13-15, 28-30, 43-45), the implementation and suggested parameters given in [51] were used: for LMIR.ABS features, we set parameter $\delta = 0.5$; for LMIR.DIR features, we set parameter $\mu = 50$; for LMIR.JM features, we set parameter $\lambda = 0.5$.
(2) For BM25 feature, we set its parameters as $k_1 = 1.2$, $k_3 = 7$ and $b = 0.75$.

3.4 Extracting Meta Information

In addition to the learning features, meta information that can be used to reproduce these learning features and even to create new features is also provided in LETOR. With the help of the meta information, one can conduct research on feature engineering, which is very important to learning to rank. For example, one can tune the parameters in existing learning features such as BM25 and LMIR, or investigate new features.

There are three kinds of meta information. The first is about the statistical information of a corpus; the second is about the raw information of the documents associated with a query; and the third is about the relationship among documents in a corpus.

Corpus meta information

An xml file was created to contain the information about the corpus, as shown in Figure. 2. As can be seen from the figure, this file contains the number of documents, the number of streams, the number of (unique) terms in each stream, and so on. One can easily obtain quantities like *avgdl* based on such information.

Query meta information

An xml file was created for each individual query, as shown in Figure. 3, containing the raw information about the query and its associated documents. Line 3 in the figure indicates the ID of the query, which comes from the original corpus. Other information includes streaminfo, terminfo, and docinfo.

(1) The node "streaminfo" (line 5-10 in Figure 3) describes the statistical information of query terms with respect to each stream. If there are n streams in the corpus, there will be n "stream" nodes under the "streaminfo" node. The node "streamid" is consistent with the XML file containing the corpus information. We say that "a

Table 3 Learning Features for the OHSUMED Corpus

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

```
<?xml version="1.0" encoding="utf-8"?>
<collectioninfo>
  <name>OHSUMED</name>
  <docnum>348566</docnum>
  <streamnum>3</streamnum>
  <stream>
    <streamid>1</streamid>
    <streamdescription>Title</streamdescription>
    <uniquetermnum>53613</uniquetermnum>
    <totaltermnum>2770138</totaltermnum>
  </stream>
  <stream>
    <streamid>2</streamid>
    <streamdescription>Abstract</streamdescription>
    <uniquetermnum>120208</uniquetermnum>
    <totaltermnum>23680877</totaltermnum>
  </stream>
  <stream>
    <streamid>3</streamid>
    <streamdescription>Title and Abstract</streamdescription>
    <uniquetermnum>126335</uniquetermnum>
    <totaltermnum>26451015</totaltermnum>
  </stream>
</collectioninfo>
```

Fig. 2 XML file to describe a corpus

term appears in a stream (e.g., title) of a corpus" if this term is included in the stream (e.g., title) of at least one document in the corpus. The element "streamspecificquerylength" refers to the number of query terms that appear in this specific stream. It is a building block for language model features.

(2) The node "terminfo" (line 12-23 in Figure 3) contains the information of each query term with respect to each individual stream. We processed queries by stemming and removing stop words, and as a result the queries here may be a little different from the original ones. The node "termnum" indicates the total number of terms in this query. The node "streamfrequency" indicates the number of documents in the corpus that contain the query term in stream "streamid". The node "streamtermfrequency" indicates the times that a query term appears in stream "streamid" of all the documents in the corpus. A node "term" may contain multiple "streams" as its child nodes, depending on the number of streams in the corpus; a node "terminfo" may have multiple "terms" as its child nodes, depending on the number of terms in the query.

(3) The node "docinfo" (line 25-40 in Figure 3) plays a role similar to the forward index [3], but is limited to the query terms and the selected documents. The node "docnum" indicates the number of documents selected for a query, followed by the information of these selected documents. Here "docid" is consistent with the files containing the learning features. The node "label" refers to the judgment of the document with regards to a query. The node "length" under the node "stream" means the total number of words (not limited to the query terms) in the stream of a document, and the node "uniquelength" means the number of unique words contained in the stream of a document. The node "termfrequency" indicates the times that a query term appears in the stream of a document. Similarly, a node "docinfo" may contain multiple child nodes of "doc"; a node "doc" may contain multiple nodes of "stream"; and a node"stream" may contain multiple child nodes of "term".

```
<?xml version="1.0" encoding="utf-8"?>
<query>
  <id>xxx</id>

  <streaminfo>
    <stream>
      <streamid>1</streamid>
      <streamspecificquerylength>4</streamspecificquerylength>
    </stream>
  </streaminfo>

  <terminfo>
    <termnum>xxx</termnum>
    <term>
      <termid>xxx</termid>
      <word>xxx</word>
      <stream>
        <streamid>xxx</streamid>
        <streamfrequency>xxx</streamfrequency>
        <streamtermfrequency>xxx</streamtermfrequency>
      </stream>
    </term>
  </terminfo>

  <docinfo>
    <docnum>xxx</docnum>
    <doc>
      <docid>xxx</docid>
      <label>xxx</label>
      <stream>
        <streamid>xxx</streamid>
        <length>xxx</length>
        <uniquelength>xxx</uniquelength>
        <term>
          <termid>xxx</termid>
          <termfrequency>xxx</termfrequency>
        </term>
      </stream>
    </doc>
  </docinfo>

</query>
```

Fig. 3 XML file to describe a query

Additional meta information

As requested by researchers working on learning to rank, we have created several additional files that contain the hyperlink graph, the sitemap information, and the similarity relationship matrix of the corpora. The hyperlink graph and the sitemap information (built by analyzing URLs of all the documents according to some heuristic rules) are specific for the "Gov" corpus. With the data, one can study link analysis algorithms and relevance propagation algorithms [32]. The similarity matrix is provided for the OHSUMED corpus, which describes the similarity between all the sampled documents associated with a query. With such kind of information, one can study the problem of dependent ranking [52,34,33].

Table 4 Data Partitioning for five-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

3.5 Finalizing Datasets

As described in Section 3.1, there are seven datasets in the LETOR collection: TD2003, TD2004, NP2003, NP2004, HP2003, HP2004, and OHSUMED. There are three versions for each dataset: Feature_NULL, Feature_MIN, and QueryLevelNorm.

(1) *Feature_NULL*: Since some documents may not contain query terms in certain streams (such as URL), we used "NULL" to indicate that the corresponding feature is absent for a query-document pair. One may need to preprocess these "NULL" values before applying a learning to rank algorithm. One can also study how to handle such absent features [7] with this version of data.
(2) *Feature_MIN*: In this version, the "NULL" values have been replaced by the minimal value of a feature in all the documents associated with a given query. This version of dataset can be directly used for learning.
(3) *QueryLevelNorm*: Taking into consideration that the values of different features or different queries may vary largely, this version further conducted query-level normalization of the feature values based on the Feature_MIN version.

We partitioned each dataset into five parts with about the same number of queries, denoted as S1, S2, S3, S4, and S5, for five-fold cross validation. In each fold, we propose using three parts for training, one part for validation, and the remaining part for test (See Table 4). The training set is used to learn ranking models. The validation set is used to tune the hyper parameters of the learning algorithms, such as the number of iterations in RankBoost and the combination coefficient in the objective function of Ranking SVM. The test set is used to evaluate the performance of the learned ranking models. Note that since we conducted five-fold cross validation, the reported performance of a ranking method is actually the average performance over the five trials.

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/`.

4 Benchmarking the Collection

On top of the data collections introduced above, standard evaluation tools and the evaluation results for several represenatative learning to rank algorithms are also provided in LETOR. The availability of these baseline results can ease the comparison between existing algorithms and newly designed algorithms, and ensure the objectiveness and correctness of the comparison.

In this section, we will introduce the experimental settings for the evaluation of the baselines algorithms, and make discussions on the experimental results.

4.1 Evaluation Measures and Tools

Three widely used measures were adopted for evaluation: precision at position k [2], average precision (AP) [2], and normalized discounted cumulative gain (NDCG) [21].

Precision at Position k (P@k)

P@k is a measure for evaluating top k positions of a ranked list using two levels (relevant and irrelevant) of relevance judgment:

$$\text{P@}k = \frac{1}{k}\sum_{j=1}^{k} r_j, \tag{3}$$

where k denotes the truncation position and

$$r_j = \begin{cases} 1 & \text{if document in } j\text{-th position is relevant,} \\ 0 & \text{otherwise.} \end{cases}$$

For a set of queries, we averaged the P@k values of all the queries to get the mean P@k value. Since P@k requires binary judgments while there are three levels of relevance judgments in the OHSUMED corpus, we simply propose treating "definitely relevant" as relevant and the other two levels as irrelevant when computing P@k.

Average Precision (AP)

Average precision (AP), another measure based on two levels of relevance judgment, is defined on the basis of P@k:

$$\text{AP} = \frac{1}{|D_+|}\sum_{j=1}^{N} r_j \times \text{P@}j, \tag{4}$$

where N indicates the number of retrieved documents and $|D_+|$ denotes the number of relevant documents with respect to the query. Given a ranked list for a query, we can compute AP for this query. Then MAP is defined as the mean of AP over a set of queries.

Normalized Discount Cumulative Gain (NDCG)

NDCG@k is a measure for evaluating top k positions of a ranked list using multiple levels (labels) of relevance judgment. It is defined as follows,

$$\text{NDCG@}k = N_k^{-1}\sum_{j=1}^{k} g(r_j)d(j), \tag{5}$$

where k has the same meaning as in Eq (3); N_k denotes the maximum[3] of $\sum_{j=1}^{k} g(r_j)d(j)$; r_j denotes the relevance level of the document ranked at the j-th position; $g(r_j)$ denotes a gain function:

$$g(r_j) = 2^{r_j} - 1;$$

[3] The maximum is obtained when the documents are ranked in the perfect order.

and $d(j)$ denotes a discount function:

$$d(j) = \begin{cases} 1 & \text{for } j = 1, 2 \\ \frac{1}{\log_2(j)} & \text{otherwise.} \end{cases}$$

Note that in order to calculate NDCG scores, we need to define the rating of each document. For the OHSUMED dataset, we defined three ratings 0, 1, 2, corresponding to "irrelevant", "partially relevant", and "definitely relevant" respectively; and for the other six datasets, we defined two ratings 0, 1 corresponding to "irrelevant" and "relevant".

To avoid differences in the evaluation results due to different implementations of these measures, a set of standard evaluation tools are provided in LETOR. The tools are written in perl, and can output Precision, MAP and NDCG for the ranking results of a given ranking algorithm, and the significance test results between two given ranking algorithms. We encourage all the users to use the tools. By using a single set of evaluation tools, the experimental results of different methods can be easily and impartially compared. The input to the evaluation tools should be one of the original datasets in the collection, because the evaluation tools (such as Eval-Score-3.0.pl) sort the documents according to their input order when the documents have the same ranking scores. That is, the evaluation tools are sensitive to the order of documents in the input file.

4.2 Baseline Ranking Algorithms

We tested a number of learning to rank algorithms on LETOR and provided the corresponding results as beaslines. To make fair comparisons, we tried to use the same setting for all the algorithms. Firstly, most algorithms adopted linear ranking functions, except RankBoost and FRank, which adopted non-linear ranking functions by combining multiple binary weak rankers. Secondly, all the algorithms utilized MAP on validation set for model selection.

4.2.1 Pointwise Approach

As for the pointwise approach, we tested the linear regression based algorithm on LETOR. This algorithm aims to learn a linear ranking function which maps a feature vector to a real value. Since only relevance label is provided for each query-document pair, one needs to map the label to a real value for training. The general rule is that after mapping the real value of a more relevant document should be larger that that of a less relevant document. In our experiments, we used the validation set to select a good mapping from labels to real values.

4.2.2 Pairwise Approach

Several pairwise ranking algorithms were tested on LETOR, including Ranking SVM, RankBoost, and FRank.

The basic idea of Ranking SVM [17, 22] is to formalize learning to rank as a problem of binary classification on document pairs, and then to solve the classification problem using Support Vector Machines. The public tool of SVMlight was used in our

experiment. We chose the linear ranking function, and tuned the parameter c using the validation set. To reduce the training time, we set -# 5000 when running SVMlight.

RankBoost [13] also formalizes learning to rank as a problem of binary classification, but solves the classification problem by means of boosting. Like all the boosting algorithms, RankBoost trains one weak ranker at each round of iteration, and combines these weak rankers to get the final ranking function. After each round, the document pairs are re-weighted by decreasing the weights of correctly ranked pairs and increasing the weights of incorrectly ranked pairs. In our implementation, we defined each weak ranker on the basis of a single feature. With a proper threshold, the weak ranker has binary output {0, 1}. For each round of iteration, we selected the best weak ranker from (# of features) x (255 thresholds) candidates. The validation set was used to determine the best number of iterations.

FRank [42] is another pairwise ranking algorithm that utilizes a novel loss function called the fidelity loss within the probabilistic ranking framework [4]. The fidelity loss has several nice properties for ranking. To efficiently minimize the fidelity loss, a generalized additive model is adopted. In our experiments, the validation set was employed to determine the number of weak learners in the additive model.

4.2.3 Listwise Approach

Four listwise ranking algorithms were tested on LETOR, i.e., ListNet, AdaRank-MAP, AdaRank-NDCG, and SVMMAP.

ListNet [6] is based on a probability distribution on permutations. Specifically, for a document list, ListNet first uses the Luce model to define a permutation probability distribution based on the scores outputted by the ranking function, and then defines another distribution based on the ground truth labels. After that, the cross entropy between the two distributions is used to define the loss function. To define the distribution based on the ground truth, ListNet needs to map the relevance label of a query-document pair to a real-valued score. In our experiments, the validation set was used to determine the mapping.

AdaRank-MAP [47] manages to directly optimize MAP. The basic idea of AdaRank-MAP is to repeatedly construct 'weak rankers' on the basis of re-weighted training queries and finally linearly combine the weak rankers for making ranking predictions. AdaRank-MAP utilizes MAP to measure the goodness of a weak ranker. In our experiments, the validation set was employed to determine the number of weak rankers.

AdaRank-NDCG [47] manages to directly optimize NDCG. The basic idea of AdaRank-NDCG is Similar to that of AdaRank-MAP. the difference lies in that AdaRank-NDCG utilizes NDCG to measure the goodness of a weak ranker. In our experiments, the validation set was employed to determine the number of weak rankers.

SVMMAP [50] is a structured Support Vector Machine (SVM) method that optimizes an upper bound of (1-AP) in the predicted rankings. There is a hyper parameter in the loss function of SVMMAP. In our experiments, the validation set was employed to determine the value of the hyper parameter.

4.3 Results

The ranking performances of the aforementioned algorithms are listed in Table 5 - 18. From the experimental results, we find that listwise ranking algorithms perform very

Table 5 NDCG on TD2003 dataset

Algorithm	@1	@3	@5	@10
Regression	0.32	0.3071	0.2984	0.326
Ranking SVM	0.32	0.3441	0.3621	0.346
RankBoost	0.28	0.3246	0.3149	0.3122
FRank	0.3	0.2671	0.2468	0.269
ListNet	0.4	0.3365	0.3393	0.348
AdaRank-MAP	0.26	0.3067	0.3029	0.306
AdaRank-NDCG	0.36	0.2908	0.2939	0.303
SVMMAP	0.32	0.3199	0.3318	0.3282

Table 6 Precision and MAP on TD2003 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.32	0.26	0.216	0.178	0.2409
Ranking SVM	0.32	0.2933	0.276	0.188	0.2628
RankBoost	0.28	0.28	0.232	0.1700	0.2274
FRank	0.3	0.2333	0.172	0.152	0.2031
ListNet	0.4	0.2933	0.252	0.200	0.2753
AdaRank-MAP	0.26	0.26	0.208	0.158	0.2283
AdaRank-NDCG	0.36	0.24	0.204	0.164	0.2368
SVMMAP	0.32	0.2533	0.232	0.1700	0.2445

Table 7 NDCG on TD2004 dataset

Algorithm	@1	@3	@5	@10
Regression	0.36	0.3352	0.3257	0.303
Ranking SVM	0.4133	0.3467	0.324	0.307
RankBoost	0.5067	0.4295	0.3878	0.3504
FRank	0.4933	0.3875	0.3629	0.333
ListNet	0.36	0.3573	0.3325	0.317
AdaRank-MAP	0.4133	0.3757	0.3602	0.328
AdaRank-NDCG	0.4267	0.3688	0.3514	0.316
SVMMAP	0.2933	0.3035	0.3007	0.2907

Table 8 Precision and MAP on TD2004 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.36	0.3333	0.312	0.249	0.2078
Ranking SVM	0.4133	0.3467	0.3013	0.252	0.2237
RankBoost	0.5067	0.4267	0.352	0.2747	0.2614
FRank	0.4933	0.3778	0.3333	0.262	0.2388
ListNet	0.36	0.36	0.3067	0.256	0.2231
AdaRank-MAP	0.4133	0.3689	0.328	0.249	0.2189
AdaRank-NDCG	0.4267	0.3644	0.328	0.248	0.1936
SVMMAP	0.2933	0.3022	0.2907	0.2467	0.2049

Table 9 NDCG on NP2003 dataset

Algorithm	@1	@3	@5	@10
Regression	0.4467	0.6135	0.6423	0.665
Ranking SVM	0.58	0.7654	0.7823	0.800
RankBoost	0.6	0.7636	0.7818	0.8068
FRank	0.54	0.7261	0.7595	0.776
ListNet	0.5667	0.7579	0.7843	0.801
AdaRank-MAP	0.58	0.7286	0.7482	0.764
AdaRank-NDCG	0.56	0.7161	0.7447	0.767
SVMMAP	0.56	0.7673	0.7881	0.7975

Table 10 Precision and MAP on NP2003 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.4467	0.22	0.1453	0.081	0.5644
Ranking SVM	0.58	0.2711	0.1707	0.092	0.6957
RankBoost	0.6	0.2689	0.1693	0.0940	0.7074
FRank	0.54	0.2533	0.168	0.090	0.6640
ListNet	0.5667	0.2667	0.172	0.092	0.6895
AdaRank-MAP	0.58	0.2511	0.16	0.086	0.6783
AdaRank-NDCG	0.56	0.2467	0.1613	0.089	0.6678
SVMMAP	0.56	0.2689	0.1707	0.0893	0.6869

Table 11 NDCG on NP2004 dataset

Algorithm	@1	@3	@5	@10
Regression	0.3733	0.5554	0.6135	0.653
Ranking SVM	0.5067	0.7503	0.7957	0.806
RankBoost	0.4267	0.6274	0.6512	0.6914
FRank	0.48	0.6431	0.687	0.729
ListNet	0.5333	0.7587	0.7965	0.812
AdaRank-MAP	0.48	0.6979	0.731	0.749
AdaRank-NDCG	0.5067	0.6722	0.7122	0.738
SVMMAP	0.52	0.7489	0.7869	0.8079

Table 12 Precision and MAP on NP2004 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.3733	0.2	0.144	0.082	0.5142
Ranking SVM	0.5067	0.2622	0.1787	0.093	0.6588
RankBoost	0.4267	0.2311	0.152	0.0880	0.5640
FRank	0.48	0.2356	0.16	0.093	0.6008
ListNet	0.5333	0.2667	0.1787	0.094	0.6720
AdaRank-MAP	0.48	0.2444	0.1627	0.088	0.6220
AdaRank-NDCG	0.5067	0.2489	0.1653	0.090	0.6269
SVMMAP	0.52	0.2667	0.1787	0.0960	0.6620

Table 13 NDCG on HP2003 dataset

Algorithm	@1	@3	@5	@10
Regression	0.42	0.5097	0.5463	0.594
Ranking SVM	0.6933	0.7749	0.7954	0.807
RankBoost	0.6667	0.792	0.8034	0.8171
FRank	0.6533	0.7432	0.778	0.797
ListNet	0.72	0.8128	0.8298	0.837
AdaRank-MAP	0.7333	0.8051	0.8252	0.838
AdaRank-NDCG	0.7133	0.7902	0.8006	0.806
SVMMAP	0.7133	0.7791	0.7922	0.7994

Table 14 Precision and MAP on HP2003 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.42	0.2111	0.1453	0.088	0.4968
Ranking SVM	0.6933	0.3089	0.1987	0.104	0.7408
RankBoost	0.6667	0.3111	0.1987	0.1053	0.7330
FRank	0.6533	0.2889	0.1987	0.106	0.7095
ListNet	0.72	0.32	0.204	0.106	0.7659
AdaRank-MAP	0.7333	0.3089	0.1987	0.106	0.7710
AdaRank-NDCG	0.7133	0.3111	0.1947	0.100	0.7480
SVMMAP	0.7133	0.3089	0.1947	0.1000	0.7421

Table 15 NDCG on HP2004 dataset

Algorithm	@1	@3	@5	@10
Regression	0.3867	0.5752	0.613	0.646
Ranking SVM	0.5733	0.7147	0.7512	0.768
RankBoost	0.5067	0.6989	0.7211	0.7428
FRank	0.6	0.7287	0.7486	0.761
ListNet	0.6	0.7213	0.7694	0.784
AdaRank-MAP	0.6133	0.8164	0.8277	0.832
AdaRank-NDCG	0.5867	0.7512	0.792	0.805
SVMMAP	0.6267	0.7536	0.8011	0.8062

Table 16 Precision and MAP on HP2004 dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.3867	0.2133	0.144	0.084	0.5256
Ranking SVM	0.5733	0.2667	0.1787	0.096	0.6675
RankBoost	0.5067	0.2533	0.1653	0.0920	0.6251
FRank	0.6	0.2622	0.168	0.089	0.6817
ListNet	0.6	0.2711	0.1893	0.098	0.6899
AdaRank-MAP	0.6133	0.2978	0.1867	0.094	0.7219
AdaRank-NDCG	0.5867	0.2711	0.1813	0.096	0.6914
SVMMAP	0.6267	0.28	0.1893	0.0960	0.7176

Table 17 NDCG on OHSUMED dataset

Algorithm	@1	@3	@5	@10
Regression	0.4456	0.4426	0.4278	0.411
Ranking SVM	0.4958	0.4207	0.4164	0.414
RankBoost	0.4632	0.4555	0.4494	0.4302
FRank	0.53	0.4812	0.4588	0.443
ListNet	0.5326	0.4732	0.4432	0.441
AdaRank-MAP	0.5388	0.4682	0.4613	0.442
AdaRank-NDCG	0.533	0.479	0.4673	0.449
SVMMAP	0.5229	0.4663	0.4516	0.4319

Table 18 Precision and MAP on OHSUMED dataset

Algorithm	@1	@3	@5	@10	MAP
Regression	0.5965	0.5768	0.5337	0.466	0.4220
Ranking SVM	0.5974	0.5427	0.5319	0.486	0.4334
RankBoost	0.5576	0.5609	0.5447	0.4966	0.4411
FRank	0.6429	0.5925	0.5638	0.501	0.4439
ListNet	0.6524	0.6016	0.5502	0.497	0.4457
AdaRank-MAP	0.6338	0.5895	0.5674	0.497	0.4487
AdaRank-NDCG	0.6719	0.5984	0.5767	0.508	0.4498
SVMMAP	0.6433	0.5802	0.5523	0.4910	0.4453

well on most datasets. Among the four listwise ranking algorithms, ListNet seems to be better than the others. AdaRank-MAP, AdaRank-NDCG and SVMMAP obtain similar performances. Pairwise ranking algorithms achieve good ranking accuracy on some (but 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 good results on TD2004 and NP2004. In contrast, simple linear regression performs worse than the pairwise and listwise ranking algorithms. Its results are not so good on most datasets.

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

$$S_i(M) = \sum_{j=1}^{7} \sum_{k=1}^{8} \mathbf{1}_{\{M_i(j) > M_k(j)\}}$$

where j is the index of a dataset, i and k are the indices of an algorithm, $M_i(j)$ is the performance of i-th algorithm on j-th dataset in terms of measure M (such as MAP), and $\mathbf{1}_{\{M_i(j) > M_k(j)\}}$ is the indicator function

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

It is clear that the larger $S_i(M)$ is, the better the i-th algorithm performs. For ease of reference, we call this measure *winning number*. Figure 4 shows the winning number in terms of NDCG for all the algorithms under investigation. From this figure, we have the following observations [4].

(1) In terms of NDCG@1, among the four listwise ranking algorithms, ListNet is better than AdaRank-MAP and AdaRank-NDCG, while SVMMAP 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. Overall, the listwise algorithms seem to perform better than the pointwise and pairwise algorithms.
(2) In terms of NDCG@3, ListNet and AdaRank-MAP perform much better than the other algorithms, while the performances of Ranking SVM, RankBoost, AdaRank-NDCG, and SVMMAP are very similar to each other.
(3) For NDCG@10, one can get similar conclusions to those for NDCG@3.

Comparing NDCG@1, NDCG@3, and NDCG@10, it seems that the listwise ranking algorithms have certain advantages over other algorithms at top positions (position 1) of the ranking results. Here we give a possible explanation. Because the loss functions of listwise algorithms are defined on all the documents of a query, it can consider all the documents and make use of the position information of them. In contrast, the loss functions of the pointwise and pairwise algorithms are defined on a single document or a document pair. It cannot access the scores of all the documents at the same time and cannot see the position information of each document. Since most IR measures (such as MAP and NDCG) are position based, listwise algorithms which can see the position information in their loss functions should perform better than pointwise and pairwise algorithms which cannot see such information in their loss functions.

Figure 5 shows the winning number in terms of Precision and MAP. We have the following observations from the figure.

[4] These observations are based on the results on the LETOR website when the paper was submitted. The website is continuously updating to incorporate the active contributions from the IR community. For the latest status of the algorithms, please refer to the LETOR website: `http://research.microsoft.com/~letor`.

(1) In terms of P@1, among the four listwise ranking algorithms, ListNet is better than AdaRank-NDCG, while AdaRank-MAP and SVMMAP perform worse than AdaRank-NDCG. The three pairwise ranking algorithms achieve comparable results, among which Ranking SVM seems to be slightly better. Overall, the listwise algorithms seem to perform than the pointwise and pairwise algorithms.
(2) For P@3, one can get similar conclusions to those for P@1.
(3) In terms of P@10, ListNet performs much better than all the other algorithms; the performances of Ranking SVM, RankBoost and FRank are better than AdaRank-MAP, AdaRank-NDCG, and SVMMAP.
(4) In terms of MAP, ListNet is the best one; Ranking SVM, AdaRank-MAP, and SVMMAP achieve similar results, and are better than the remaining algorithms. Furthermore, the variance among the three pairwise ranking algorithms is much larger than the variance in terms of other measures (P@1, 3 and 10). The possible explanation is as follows. 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 (ListNet, AdaRank-MAP, AdaRank-NDCG, and SVMMAP) have certain advantages over other algorithms, especially for the top positions of the ranking results.

Note that the above experimental results are in some sense still preliminary, since the result of almost every algorithm can be further improved. For example, for regression, we can add a regularization item to make it more robust; for Ranking SVM, if the time complexity is not an issue, we can remove the constraint of -# 5000 to achieve a better convergence of the algorithm; for ListNet, we can also add a regularization item to its loss function and make it more generalizable to the test set. Considering these issues, we would like to call for contributions from the research community. Researchers are encouraged to submit the results of their newly developed algorithms as well as their carefully tuned existing algorithms to LETOR. In order to let others re-produce the submitted results, the contributors are respectfully asked to prepare a package for the algorithm, including

(1) a brief document introducing the algorithm;
(2) an executable file of the algorithm;
(3) a script to run the algorithm on the seven datasets of LETOR.

We believe with the collaborative efforts of the entire community, we can have more versatile and reliable baselines on LETOR, and better facilitate the research on learning to rank.

5 Supporting New Research Directions

So far LETOR has mainly been used as an experimental platform to compare different algorithms. In this section, we show that LETOR can also be used to support many new research directions.

5.1 Ranking Models

Most of the previous work (as reviewed in Section 2) focuses on developing better loss functions, and simply uses a scoring function as the ranking model. It may be one of

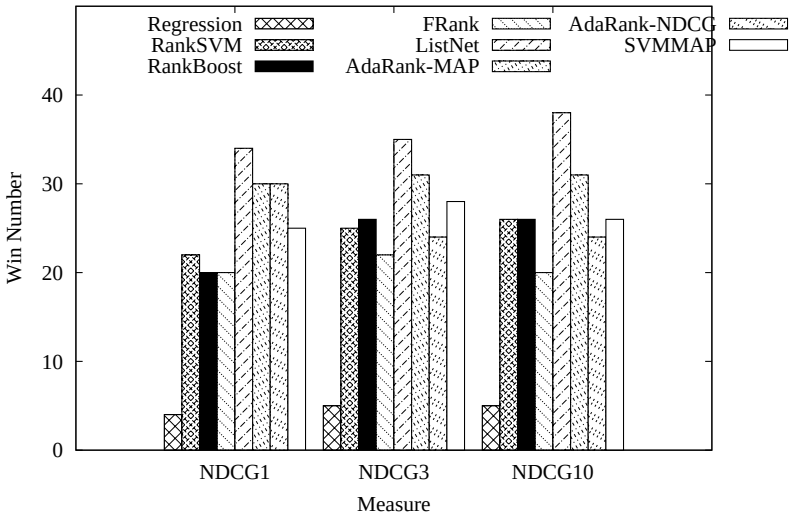

Fig. 4 Comparison cross seven datasets by NDCG

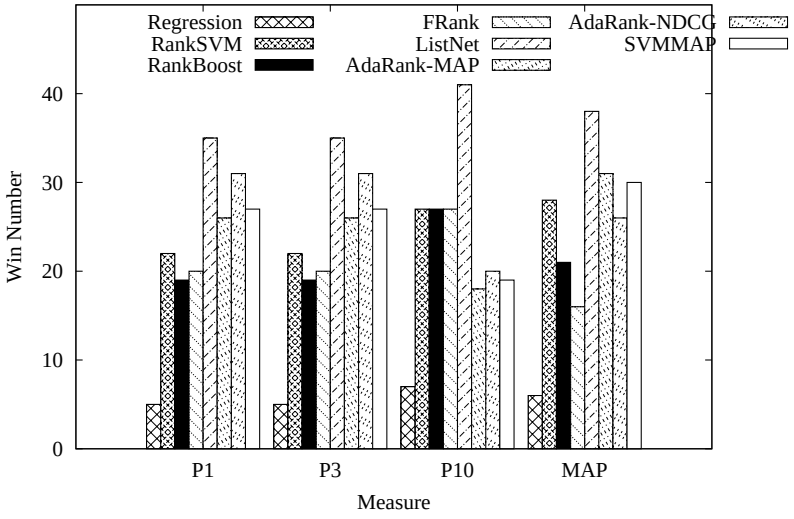

Fig. 5 Comparison cross seven datasets by Precision and MAP

the major topics at the next step to investigate new ranking models. For example, one can study new algorithms using pairwise and listwise ranking functions. Please note the challenge of using a pairwise/listwise ranking function. That is, the test complexity will be much higher than that of using a scoring function. One should pay attention to the issue if he/she performs research on pairwise and listwise ranking functions.

5.2 Feature Engineering

Feature is, by all means, very important for learning to rank algorithms. Since LETOR contains rich meta information, it can be used to study feature related problems.

Feature Extraction

The performance of a ranking algorithm greatly depends on the effectiveness of the features used. LETOR contains the low-level information such as term frequency and document length. It also contains rich meta information about the corpora and the documents. These can be used to derive new features, and study their contributions to ranking.

Feature Selection

Feature selection has been extensively studied for classification. However, as far as we know, the discussions on feature selection for ranking are still very limited. LETOR contains tens of standard features, and it is feasible to use LETOR to study the selection of most effective features for ranking.

Dimensionality Reduction

Different from feature selection, dimensionality reduction tries to reduce the number of features by transforming/combining the original features. Dimensionality reduction has been shown very effective in many applications, such as face detection and signal processing. Similarly to feature selection, little work has been done on dimensionality reduction for ranking. It is an important research topic, and LETOR can be used to support such research.

5.3 New Ranking Scenarios

LETOR also offers the opportunities to investigate some new ranking scenarios. Here we give several examples as follows.

Query Classification and Query Dependent Ranking

In most previous work, a single ranking function is used to handle all kinds of queries. This may not be appropriate, particularly for web search. Queries in web search may vary largely in semantics and user intentions. Using a single model alone would make compromises among queries and result in lower accuracy in relevance ranking. Instead, it would be better to exploit different ranking models for different queries. Since LETOR contains several different kinds of query sets (such as topic distillation, homepage finding, and named page finding) and rich information about queries, it is possible to study the problems of query classification and query dependent ranking [14].

Beyond Independent Ranking

Existing technologies on learning to rank assume that the relevance of a document is independent of the relevance of other documents. The assumption makes it possible to score each document independently first and sort the documents according to their scores after that. In reality, the assumption may not always hold. There are many

retrieval applications in which documents are not independent and relation information among documents can be or must be exploited. For example, Web pages from the same site form a sitemap hierarchy. If both a page and its parent page are about the topic of the query, then it would be better to rank higher the parent page for this query. As another example, similarities between documents are available, and we can leverage the information to enhance relevance ranking. Other problems like Subtopic Retrieval [36] also need utilize relation information. LETOR contains rich relation information, including hyperlink graph, similarity matrix, and sitemap hierarchy, and therefore can well support the research on dependent ranking.

Multitask Ranking and Transfer Ranking

Multitask learning aims at learning several related task at the same time, and the learning of the tasks can benefit from each other. In other words, the information provided by the training signal for each task serves as a domain-specific inductive bias for the other tasks. Transfer learning uses the data in one or more auxiliary domains to help the learning task in the main domain. Because LETOR contains seven query sets and three different retrieval tasks, it is a good test bed for multitask ranking and transfer ranking.

To summarize, although the current use of LETOR is mostly about algorithm comparison, LETOR actually can be used to support much richer research agenda. We hope that more and more interesting researches can be carried out with the help of LETOR, and the state of the art of learning to rnak can be significantly advanced.

6 Limitations

Although LETOR has been widely used, it has certain limitations as listed below.

Document Sampling Strategy

For the "Gov" datasets, the retrieval problem is essentially cast as a re-ranking task (for top 1000 documents) in LETOR. On one hand, this is a common practice for real-world Web search engines. Usually two rankers are used by a search engine for sake of efficiency: firstly a simple ranker (e.g., BM25) is used to select some candidate documents, and then a more complex ranker (e.g., the learning to rank algorithms as mentioned in the paper) is used to produce the final ranking result. On the other hand, however, there are also some retrieval applications that should not be cast as a re-ranking task. We will add datasets beyond re-ranking settings to LETOR in the future.

For the "Gov" datasets, we sampled documents for each query using a cutoff number of 1000. We will study the impact of the cutoff number on the performances of the ranking algorithms. It is possible that the dataset should be refined using a better cutoff number.

Features

In both academic and industrial communities, more and more features have been studied and applied to improve ranking accuracy. The feature list provided in LETOR is far away from comprehensive. For example, document features (such as document length) are not included in the OHSUMED dataset, and proximity features are not included in all the seven datasets. We will add more features into the LETOR datasets in the future.

Scale and Diversity of Datasets

As compared with the real-web search, the scale (number of queries) of the datasets in LETOR is not yet very large. To verify the performances of learning to rank techniques for real-web search, large scale datasets are needed. We are working on some large scale datasets and plan to release them in the future versions of LETOR.

Although there are seven query sets in LETOR3.0, there are only two document corpora involved. We will create new datasets using more document corpora in the future.

Baselines

Most baseline algorithms in LETOR use linear ranking functions. From Section 4.3, we can see that the performances of these algorithms are not good enough, since the perfect ranking should achieved the accuracy of 1 in terms of all the measures (P@k, MAP and NDCG). As pointed out in Section 3.3, class-Q features cannot be effectively used by linear ranking functions. We will add more algorithms with nonlinear ranking functions as baselines of LETOR. We also encourage researchers in the community to test more non-linear ranking models.

7 Conclusions

By explaining the data creation process and the results of the state-of-the-arts learning to rank algorithms in this paper, we have provided the information for people to better understand the nature of LETOR and to more effectively utilize the datasets in their research work.

We have received a lot of comments and feedback about LETOR after its first release. We hope we can get more suggestions from the research community. We also encourage researchers to contribute to LETOR by submitting their results.

Finally, we expect that LETOR can be just a start for the research community to build benchmark datasets for learning to rank. With more and more such efforts, the research on learning to rank for IR can be significantly advanced.

Acknowledgements We would like to thank Chao-Liang Zhong, Kang Ji and Wenying Xiong for their work on the creation of the LETOR datasets, and thank Da Kuang, Chao-Liang Zhong, Yong-Deok Kim, Ming-Feng Tsai, Yi-Song Yue, Olivier Chapelle and Thorsten Joachims for their work on the baseline algorithms. We would like to thank Yunhua Hu for providing the extracted titles of the "Gov" corpus and thank Guomao Xin, Shuming Shi, Ruihua Song, Zhicheng Dou and Jirong Wen for their help on corpus processing and indexing. We would also like to thank Lan Nie,Brian D. Davison, and Xiaoguang Qi for providing the web page classification models for the feature extraction of the "Gov" corpus.

References

1. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.
3. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

4. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, New York, NY, USA, 2005. ACM Press.
5. Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193, New York, NY, USA, 2006. ACM Press.
6. Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 129–136, New York, NY, USA, 2007. ACM Press.
7. G. Chechik, G. Heitz, G. Elidan, P. Abbeel, and D. Koller. Max-margin classification of data with absent features. *J. Mach. Learn. Res.*, 9:1–21, 2008.
8. P. Chirita, J. Diederich, and W. Nejdl. MailRank: using ranking for spam detection. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 373–380. ACM New York, NY, USA, 2005.
9. M. Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, July*, pages 07–12, 2002.
10. N. Craswell and D. Hawking. Overview of the TREC-2004 Web track. 2004.
11. N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the TREC 2003 web track. In *Proceedings of TREC 2003*, 2003.
12. K. Dave, S. Lawrence, and D. Pennock. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM Press New York, NY, USA, 2003.
13. Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.
14. X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122, New York, NY, USA, 2008. ACM.
15. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 576–587. VLDB Endowment, 2004.
16. E. F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *Proceedings of the 20th International Conference on Machine Learning*, pages 250–257, 2003.
17. R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN1999*, pages 97–102, 1999.
18. W. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *SIGIR '94*, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
19. Y. Hu, G. Xin, R. Song, G. Hu, S. Shi, Y. Cao, and H. Li. Title extraction from bodies of html documents and its application to web page retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 250–257, New York, NY, USA, 2005. ACM.
20. J. C. Huang and B. J. Frey. Structured ranking learning using cumulative distribution networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 697–704. 2009.
21. K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
22. T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM Press.
23. D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A New Benchmark Collection for Text Categorization Research. *The Journal of Machine Learning Research*, 5:361–397, 2004.
24. L. Li and H.-T. Lin. Ordinal regression by extended binary classification. In *NIPS*, pages 865–872, 2006.
25. P. Li, C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems 20*, pages 897–904. MIT Press, Cambridge, MA, 2008.

26. I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 437–444, New York, NY, USA, 2006. ACM.
27. T. Minka and S. Robertson. Selection bias in the LETOR datasets. In *Proceedings of SIGIR 2008 Workshop on Learning to Rank for Information Retrieval*, 2008.
28. L. Nie, B. D. Davison, and X. Qi. Topical link analysis for web search. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–98, New York, NY, USA, 2006. ACM.
29. B. Pang and L. Lee. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115–124. Association for Computational Linguistics Morristown, NJ, USA, 2005.
30. T. Qin, T. Liu, J. Xu, and H. Li. How to Make LETOR More Useful and Reliable. In *Proceedings of SIGIR 2008 Workshop on Learning to Rank for Information Retrieval*, 2008.
31. T. Qin, T.-Y. Liu, and H. Li. A general approximation framework for direct optimization of information retrieval measures. Technical Report MSR-TR-2008-164, Microsoft Corporation, 2008.
32. T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. In *SIGIR '05*, pages 408–415, New York, NY, USA, 2005. ACM Press.
33. T. Qin, T.-Y. Liu, X.-D. Zhang, D.-S. Wang, and H. Li. Global ranking using continuous conditional random fields. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS*. MIT Press, 2008.
34. T. Qin, T.-Y. Liu, X.-D. Zhang, D.-S. Wang, W.-Y. Xiong, and H. Li. Learning to rank relational objects and its application to web search. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 407–416, New York, NY, USA, 2008. ACM.
35. T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2008.
36. T. Qin, X.-D. Zhang, D.-S. Wang, T.-Y. Liu, W. Lai, and H. Li. Ranking with multiple hyperplanes. In *SIGIR '07*, pages 279–286, New York, NY, USA, 2007. ACM Press.
37. S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, New York, NY, USA, 2004. ACM.
38. S. E. Robertson and D. A. Hull. The TREC-9 filtering track final report. In *TREC*, pages 25–40, 2000.
39. A. Shakery and C. Zhai. Relevance Propagation for Topic Distillation UIUC TREC-2003 Web Track Experiments. In *Proceedings of TREC*, pages 673–677, 2003.
40. M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 77–86, New York, NY, USA, 2008. ACM.
41. TREC2004 Web track guideline. `http://research.microsoft.com/en-us/um/people/nickcr/guidelines_2004.html`.
42. M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *SIGIR '07*, pages 383–390, New York, NY, USA, 2007. ACM Press.
43. M. N. Volkovs and R. S. Zemel. Boltzrank: learning to maximize expected ranking gain. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1089–1096, New York, NY, USA, 2009. ACM.
44. E. Voorhees and D. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.
45. F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank - theory and algorithm. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, New York, NY, USA, 2008. ACM Press.
46. J. Xu, Y. Cao, H. Li, and M. Zhao. Ranking definitions with supervised learning methods. In *International World Wide Web Conference*, pages 811–819. ACM Press New York, NY, USA, 2005.
47. J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, New York, NY, USA. ACM Press.

48. J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in learning to rank. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 107–114, New York, NY, USA, 2008. ACM.
49. G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193, New York, NY, USA, 2005. ACM.
50. Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, New York, NY, USA, 2007. ACM Press.
51. C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to Ad Hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM Press New York, NY, USA, 2001.
52. C. X. Zhai, W. W. Cohen, and J. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 10–17, New York, NY, USA, 2003. ACM Press.