How to Make LETOR More Useful and Reliable

Tao Qin, Tie-Yan Liu, Jun Xu, Hang Li
Microsoft Research Asia
No.49 Zhichun Road, Haidian District
Beijing 100190, P.R. China
tsintao@gmail.com,
{tyliu, junxu, hangli}@microsoft.com

ABSTRACT

Learning to rank has attracted great attention recently in both information retrieval and machine learning communities. However, the lack of public dataset had stood in its way until the LETOR benchmark dataset (actually a group of three datasets) was released in the SIGIR 2007 workshop on Learning to Rank for Information Retrieval (LR4IR 2007). Since then, this dataset has been widely used in many learning to rank papers, and has greatly speeded up the corresponding research. In this paper, we discuss how to further improve LETOR to make it more useful and reliable. First, we notice that some low-level information, such as the term frequency in each stream (title, body, url, anchor, etc.) and the stream length, are missing in the current feature set of LETOR. We propose adding the information to LETOR, so as to enable the reproduction or optimization of models like BM25. Second, we find that the sampling of documents associated with each query in LETOR was somehow biased. We therefore propose a new document sampling strategy to reduce the bias. Third, the scale (less than 100 queries) of LETOR is relatively small for real world ranking applications. We propose adding more queries to the current datasets in LETOR, and/or building even larger datasets by leveraging the effort of the entire information retrieval community.

1. INTRODUCTION

Ranking is the central problem for many information retrieval (IR) applications, including document retrieval, collaborative filtering, key term extraction, definition finding, important email routing, sentiment analysis, product rating, and anti web spam. In the task, given a set of objects, a ranking model (function) is used to calculate a score for each object and the objects are sorted in the descending order of the scores.

Learning to rank has attracted great attention recently in both information retrieval and machine learning communities. Many algorithms have been proposed for learning to rank, such as the pointwise approach [12], the pairwise approach [8, 9, 5, 2] and the listwise approach [3, 17]. The lack of public dataset had stood in the way of research on learning to rank until the LETOR benchkmar dataset was released in the SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007). Since then, this dataset has been widely used in many learning to rank papers [16, 21, 4, 10, 25, 23, 7, 6].

Although the release of LETOR has greatly speeded up the research on learning to rank, we also notice several issues in it. To make LETOR more useful and reliable, in this paper, we discuss how to improve it from three aspects:

(1) Some low-level information is missing in the current feature set of LETOR, such as the term frequency in each stream (title, body, url, anchor et. al.) and the stream length. This makes it difficult to reproduce and/or optimize models like BM25. We suggest adding these low-level features to the new version of LETOR to solve the problem.

(2) The sampling of documents associated with each query in LETOR was somehow biased because of the specific document selection strategy. We consider three new document selection strategies in this paper to avoid or reduce the bias, and discuss their feasibility and costs.

(3) The scale (less than 100 queries) of LETOR is not large enough for real-world ranking applications. We propose adding more queries to the current datasets in LETOR. We also suggest leveraging the efforts of the entire IR community to collect larger-scale data for learning to rank research.

The remaining part of this paper is organized as follows: a brief introduction to LETOR is given in Section 2; the details of the proposed ways of improving the existing datasets in LETOR are discussed in Section 3; In Section 4, we propose creating larger-scale training data; and conclusions are given in the last section.

2. ANALYSIS ON THE LETOR DATASET

2.1 Overview of LETOR

LETOR was built based on two widely-used data collections in information retrieval: the OHSUMED collection used in the information filtering task of TREC 2000, and the ".gov" collection used in the topic distillation tasks of TREC 2003 and 2004. Accordingly, there are three sub datasets in LETOR, namely OHSUMED, TD2003, and TD2004.

Appearing in the proceedings of SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR 2008).
Copyright is reserved by the authors of the paper.

Table 1: Statistics of three datasets in LETOR

Dataset	#Query	#Doc	#Doc/#Query	#Feature
OHSUMED	106	16140	152.26	25
TD2003	50	49171	983.42	44
TD2004	75	74170	988.93	44

Both 'low level' and 'high level' features have been extracted for each query-document pair in the OHSUMED collection. Low-level features include term frequency (tf), inverse document frequency (idf), document length (dl) and their combinations [1]. High-level features include the outputs of BM25 [18] and language model for IR [26].

Four kinds of features have been extracted for each query-document pair in the ".gov" collections: low-level content features, high-level content features, hyperlink features, and hybrid features. The low-level and high-level content features are similar to that in the OHSUMED dataset. Hyperlink features include PageRank [14], HITS [11], and their variations (HostRank [24], topical PageRank and topical HITS [13]). Hybrid features refer to those features containing both content and hyperlink information, including "hyperlink-based relevance propagation" [19] and "sitemap-based relevance propagation" [15].

Since there are too many documents in the OHSUMED and ".gov" collections, when building LETOR, documents are sampled for each query to facilitate the experiments on ranking, according to the following strategy. For the OHSUMED dataset, all judged documents were selected, and all un-judged documents were neglected. In this way, about 150 documents per query were eventually selected, as shown in Table 1. The sampling strategy for the ".gov" collection is a little different. First, BM25 was used as the model to rank all the documents with respect to each query, and then the top 1000 documents (if there are such number of documents containing the query) were selected. Considering that some relevant documents may not appear in the top 1000 results, all the relevant documents for each query were also added to the selected document pool. The statistics on the TD2003 and TD2004 datasets can also be found in Table 1.

The value of the LETOR dataset lies in the following two aspects:

(1) LETOR contains a set of standard features, and so researchers can directly use them to test the effectiveness of their ranking algorithms, without the necessity of creating a dataset by their own. Because the dataset creation is very costly (including collection hunting, query selection, feature extraction, etc.), LETOR has greatly reduced the barrier of the research on learning to rank.

(2) LETOR makes the fair comparison among different learning to rank algorithms possible. Before the release of LETOR, different datasets (i.e. different query sets, different document collections, different features, or different evaluation tools) were used in different papers. As a result, it is not easy to get conclusive observations on the effectiveness of the learning to rank algorithms, by comparing the performances reported in different papers. LETOR has made it possible to compare the algorithms, since a standard dataset and a set of standard evaluation tools are used. Furthermore, in the current version of LETOR (version 2.0), several state-of-the-art baselines have been included, which even further ease the algorithm comparison: researchers even do not need to implement some of the baselines to be compared by their own.

2.2 Aspects of LETOR to Be Improved

Although LETOR has achieved great success, there are also problems with it. While using the LETOR dataset, we find that there are several aspects that should be improved, in order to make it more useful and reliable.

First, some "low-level" information is missing in the feature extraction process of LETOR.

One of the goals of LETOR is to provide standard features so that researchers using LETOR do not need to deal with the raw documents. However, the current features provided in LETOR may limit people's research. For example, a BM25 score is provided which was computed with default parameters. If one wants to further tune this model and get a more effective feature, he/she will find that there is no sufficient information to perform the task.

To better understand the problem, let us look at one of the most prominent instantiations of BM25 as follows.

Given a query Q, containing keywords $q_1, ..., q_t$, the BM25 score of a document D is computed as:

$$BM25(D,Q) = \sum_{i=1}^{t} idf(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})},$$

where $f(q_i, D)$ is the occurrences of q_i in the document D, $|D|$ is the length of the document D (i.e., the number of words), and $avgdl$ is the average document length in the entire document collection from which documents are drawn. k_1 and b are free parameters. $idf(q_i)$ is the IDF (inverse document frequency) weight of the query term q_i. It is usually computed as:

$$idf(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

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

As can be seen, to compute the BM25 score, we need to know the term frequency $f(q_i, D)$ in document D and the document frequency $n(q_i)$ in the entire collection containing each query term q_i. Besides, we also need to know the document length $|D|$ and the average document length of the entire collection. In the current version of LETOR, however, only $|D|$ is provided as a feature. $f(q_i, D)$ and $n(q_i)$ are missing[1]. This prevents researchers from tuning the parameters of BM25 (e.g. k_1 and b) and/or creating other strong features based on such low level information.

Second, the document sampling strategy used in LETOR is somehow biased.

For the OHSUMED dataset, only judged query-document pairs were selected. The judgments are "highly relevant", "partially relevant" or "irrelevant". However, this setting is not consistent with real world applications. For a IR system, when a user issues a query, the system does not know which

[1] $\sum_{i=1}^{t} f(q_i, D)$ and $\sum_{i=1}^{t} n(q_i)$ are provides in LETOR2.0 instead. One cannot use the sum of term frequency and document frequency to reproduce the BM25 score.

document is judged or not. That is, we cannot get only judged documents for ranking.

For TD2003 and TD2004 datasets, for each query, both top 1000 documents based on their BM25 scores and the relevant documents which are not ranked in top 1000 positions were selected. It is clear that such a sampling strategy is not consistent with the ranking process in real search scenarios either. Given a query, we can get its top 1000 documents by BM25, but we do not know which documents ranked out of top 1000 are relevant.

Third, the scale of the dataset in LETOR is not large enough.

Some statistics of the three datasets are listed in Table 1. As can be seen, there are only 50 and 75 queries in TD2003 and TD2004 separately, and about 100 queries in OHSUMED. After the 5-fold partitioning, each validation (testing) set only contains 10/15/21 queries in TD2003/TD2004 /OHSUMED. The small number of queries may make the experimental results got on these datasets not reliable. Failing in one single query may lead to significant performance drop.

From the view of machine learning, a larger-scale dataset will be better for a learning algorithm. From the view of real world ranking applications (e.g. web search), a ranking algorithm should also handle large-scale dataset. Considering these, it is important and necessary to add large datasets to LETOR.

3. IMPROVING EXISTING DATASETS IN LETOR

In this section, we discuss how to improve the existing datasets in LETOR. We mainly focus on the first and second aspects mentioned in the previous section.

3.1 Adding More Raw Features

For the OHSUMED dataset, besides the 25 features in the current version of LETOR, we suggest adding more features for learning. Specifically, we suggest extracting 20 new features from the fields of 'title', 'abstract', and 'title + abstract': 10 'low level features' from 'title + abstract', 5 'high level' features from 'title', and 5 'high level' features from 'abstract'. The entire feature set can be found in Table 3.

we also suggest providing 'meta features' on the OHSUMED dataset. Term frequency and document frequency of each query term, and document lengths etc. are important meta features. Some statistics on the corpus, such as the average document length, the total number of documents in the corpus etc., are also important and should be included. These meta features may not be directly used by a learning algorithm. However, based on these meta features, people can construct their own features, reproduce and tune BM25 and other strong features. Table 3 lists all of the proposed meta features.

For TD2003 and TD2004, besides the 44 features in the current version of LETOR, we suggest adding in-link number, out-link number, length of URL, number of slashes in the URL, etc. as new features. Also, we suggest extracting those existing features in all streams (URL, title, anchor and body), while features in some streams are missing in the current version of LETOR. Overall, there will be 64 features (Table. 4) which can be directly used by learning algorithms.

We also propose adding the term frequency in each stream

Table 2: Features for OHSUMED

ID	Feature Description
1	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'title'
2	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'title'
3	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'title'
4	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'title'
5	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title'
6	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'title'
7	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title'
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'
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'
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'
11	BM25 score in 'title'
12	log(BM25 score) in 'title'
13	LMIR with DIR smoothing in 'title'
14	LMIR with JM smoothing in 'title'
15	LMIR with ABS smoothing in 'title'
16	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'abstract'
17	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'abstract'
18	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'abstract'
19	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'abstract'
20	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'abstract'
21	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'abstract'
22	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'abstract'
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'
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'
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'
26	BM25 score in 'abstract'
27	log(BM25 score) 'abstract'
28	LMIR with DIR smoothing in 'abstract'
29	LMIR with JM smoothing in 'abstract'
30	LMIR with ABS smoothing in 'abstract'
31	$\sum_{q_i \in q \cap d} c(q_i, d)$ in 'title + abstract'
32	$\sum_{q_i \in q \cap d} \log\left(c(q_i, d) + 1\right)$ in 'title + abstract'
33	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$ in 'title + abstract'
34	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$ in 'title + abstract'
35	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$ in 'title + abstract'
36	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$ in 'title + abstract'
37	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$ in 'title + abstract'
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'
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'
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'
41	BM25 score in 'title + abstract'
42	log(BM25 score) 'title + abstract'
43	LMIR with DIR smoothing 'title + abstract'
44	LMIR with JM smoothing in 'title + abstract'
45	LMIR with ABS smoothing in 'title + abstract'

Table 3: Meta Features for OHSUMED

ID	Feature Description
1	Term frequency (TF) of a single query term in 'title'
2	Term frequency (TF) of a single query term in 'abstract'
3	Term frequency (TF) of a single query term in 'title + abstract'
4	Document frequency (DF) of a single query term in terms of 'title'
5	Document frequency (DF) of a single query term in terms of 'abstract'
6	Document frequency (DF) of a single query term in terms of 'title + abstract'
7	Document length in terms of 'title'
8	Document length in terms of 'abstract'
9	Document length in terms of 'title + abstract'
10	Average document length in terms of 'title'
11	Average document length in terms of 'abstract'
12	Average document length in terms of 'title + abstract'
13	Corpus size

(body, anchor, title and url) and document frequency of each query term as meta features. Note that different from OHSUMED, the stream length of each document is already included in LETOR 2.0. The meta features are listed in Table. 5. They can be used to derive new strong features or tuning the parameters in existing features like BM25.

3.2 Adopting New Sampling Strategy

Here we discuss how to better sample documents for a query from the original document collection to make LETOR more consistent with real world applications.

As we have pointed out, for the OHSUMED dataset, it might not be reasonable to only look at the small number of judged documents. A possible solution is that for a query, we include all the documents containing at least one query term. By doing so, we can signicantly increase the number of documents to be ranked. However, there are also some concerns with this method because we have introduced a number of unjudgment document into the dataset. First, in training, it is not clear how to use these un-judged documents. Simply regarding them as irrelevant may or may not be reasonable. In fact, this is a research problem by itself: how to make good use of unlabeled documents in learning. Second, in testing, it is difficult to evaluate the accuracy of a ranking algorithm. At least the current evaluation measures used in LETOR, MAP and NDCG, cannot be computed in this case. Considering the two issues, we suggest keeping the sampling of the current version of OHSUMED dataset unchanged.

For the TD2003 and TD2004 datasets, it is a different story. The judgments in these datasets only include the "relevant" documents. In other words, all unjudged documents are regarded as irrelevant [22, 17, 3, 20]. In this case, if we introduce more documents, there are no such confusions on the labels as for the OHSUMED dataset. Note that, in TD2003 and TD2004 datasets, all the relevant documents were added to the datasets, while no enough unjudged documents were added at the same time. As a result, the document distribution is changed significantly as compared to the original collection. To solve the problem, we suggest making one of the following changes.

Table 4: Features for TD2003 and TD2004

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	TD*IDF of body
12	TD*IDF of anchor
13	TD*IDF of title
14	TD*IDF of URL
15	TD*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 feature 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

Table 5: Meta Features for TREC

ID	Feature Description
1	Term frequency (TF) of a single query term in body
2	TF of a single query term in anchor
3	TF of a single query term in title
4	TF of a single query term in URL
5	TF of a single query term in whole document
6	Inverse document frequency (IDF) of a single query term in body
7	IDF of a single query term in anchor
8	IDF of a single query term in title
9	IDF of a single query term in URL
10	IDF of a single query term in whole document
11	TD*IDF of a single query term in body
12	TD*IDF of a single query term in anchor
13	TD*IDF of a single query term in title
14	TD*IDF of a single query term in URL
15	TD*IDF of a single query term in whole document
16	Average length of body
17	Average length of anchor
18	Average length of title
19	Average length of URL
20	Average length of whole document
21	Corpus size

(1) For each query, we can create a document pool with all documents containing at least one query term in it. The problem is that, however, there are too many documents in the pool for a query to rank. Table 6 shows the number of documents containing at least one query term for each query ($|D(q)|$). As can be seen, there are about 5 millon documents for TD2003[2] which need to be considered for feature extraction. Note that the size (with zip compression) of LETOR 2.0 with about 50000 documents in TD2003 dataset and 75000 documents in TD2004 dataset is about 44M bytes. The number of documents w.r.t. this new sampling strategy is about 100 times the number in the current version of TD2003 and TD2004. Then the size of the new TD2003 and TD2004 datasets will be about 4G bytes without considering the new features added. Such a dataset would be too large to download and to use[3].

(2) For a query, we first create a pool with all the documents containing at least one query term. Then we rank the documents in this pool in the descending order of their BM25 scores. Let $p = \max(pos(rd))$ be the rank position of the last relevant document. We select $p + 1000$ documents for this query for feature extraction. In this way, we will get about 0.6 million documents for TD2003[4] as shown in Table 6. This is about 10 times the current version of TD2003 and TD2004.

(3) We only change the current version of LETOR slightly. We remove all the relevant documents that are ranked out of the top 1000 positions according to their BM25 scores. In this way, the data distribution will not be as biased as before either.

[2]There are about 10 millon documents for TD2004, which are not listed here.

[3]Someone may argue that 4.4G is not very large for download. However, it would take a lot of training time for a learning algorithm.

[4]And about 0.8 million documents for TD2004.

4. CREATING LARGER SCALE DATASETS

There are only about (or less than) 100 queries in each dataset of LETOR. These datasets are relatively small. Large scale datasets are desired for the research on learning to rank. Here we suggest three possible solutions.

(1) Note that there are three tasks in TREC web track 2003 and 2004: topic distillation (TD), homepage finding (HP) and named page finding (NP). In LETOR 2.0, only topic distillation task was used for data creation. The first solution we suggest is to add the queries of the other two tasks into LETOR. There are two benefits to include all the three tasks. First, by doing so, we can enlarge the data to hundreds of queries, as shown in Table 7. Second, since there are three different types of queries, some other research topics can be conducted based on LETOR, such as query classification and query dependent ranking.

(2) Considering that hundreds of queries may be still not large enough, another possible data source is the Million Query Track at TREC 2007[5], which contains about 10000 queries (about 1700 queries are labeled in the track last year, and more queries are expected to be labeled this year.). The documents associated with these queries are retrieved from the ".gov2" collection. It would be great to create a large dataset based on this source. The creation process can be very similar to that of the current version of LETOR: indexing the ".gov2" dataset, extracting standard features, and making the data files publicly available.

(3) In addition, we make a even wilder proposal on leveraging the efforts of the entire IR community (and even web users) for data collection. The basic idea is to develop a meta search engine, which mixes the results from multiple search engines. We will collect the search log of this meta search engine to mine the ground truth. Of course, the one who uses this engine should be aware that we are recording the search log for research purpose. After mining the ground truth from the search log, we can extract features for query-document pairs and release the dataset.

The advantages of using this solution are as follows. First, it is easy for data collection. One only need to use the engine like some other search engines without any other effort. Second, the number of available queries can increase along with time, and thus can be extremely large eventually. Third, the documents are up-to-date webpages indexed by the search engines, while the documents in the ".gov" and ".gov2" collections are pretty old.

There are also concerns with this solution. First, we need to filter the logs to delete some privacy related queries. Second, the ground truth mining from log

[5]We would like to thank the reviewer for the recommendation on this data source.

Table 6: Statistics on sampled documents of TD2003 dataset

QueryID	$\|D(q)\|$	$max(pos(rd))$
1	47360	16435
2	11263	1247
3	38852	22
4	196651	1917
5	220930	3685
6	81253	193
7	115902	66641
8	117876	12
9	301335	286648
10	101552	512
11	45090	255
12	68896	166
13	1010	2
14	87993	560
15	139505	560
16	219321	1896
17	525	3
18	533	5
19	109707	316
20	79393	44112
21	1619	737
22	106215	532
23	7536	268
24	181882	7
25	46414	116
26	141552	1676
27	112917	783
28	182231	1798
29	3149	45
30	41912	409
31	68912	526
32	61599	20485
33	148548	1078
34	6694	444
35	63765	47773
36	72125	2657
37	125920	1250
38	134569	19881
39	536840	37821
40	111526	253
41	52267	1351
42	6035	250
43	64461	5340
44	28141	3004
45	19445	517
46	19856	1058
47	132448	53441
48	7609	529
49	179907	69
50	2885	80
total	4653926	629365

Table 7: Number of queries in TREC web track

Task	TREC2003	TREC2004
TD	50	75
HP+NP	300	150

data may not be very easy because it is still a research topic by its own. Alternatively, we can also release the log data directly for research.

5. CONCLUSIONS

LETOR is a public dataset for learning to rank, which eases the research in this direction. Since benchmark dataset is always very important for a research direction, continuous discussions and efforts should be given to the construction of such datasets. In this paper, we discussed how to make LETOR more useful and reliable from three aspects, i.e., adding more raw features, adopting new document sampling strategies, and creating larger scale datasets. We hope with the joint efforts of the learning to rank research community, a better benchmark dataset can be built shortly and our research work can benefit from it.

6. REFERENCES

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

[2] 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.

[3] 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.

[4] J. Elsas, V. Carvalho, and J. Carbonell. Fast learning of document ranking functions with the committee perceptron. *Proceedings of the international conference on Web search and web data mining*, pages 55–64, 2008.

[5] 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.

[6] 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 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM.

[7] J. Guiver and E. Snelson. Learning to rank with softrank and gaussian processes. In *SIGIR '08: Proceedings of the 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM Press.

[8] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN1999*, pages 97–102, 1999.

[9] R. Herbrich, T. Graepel, and K. Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.

[10] R. Jin, H. Valizadegan, and H. Li. Ranking refinement and its application to information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 397–406, New York, NY, USA, 2008. ACM.

[11] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[12] P. Li, C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS2007*, 2007.

[13] 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.

[14] 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.

[15] 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: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 408–415, New York, NY, USA, 2005. ACM.

[16] 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.

[17] 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*, 2007.

[18] S. J. M. M. H.-B. M. G. S. E. Robertson, S. Walker. Okapi at TREC-3. In *Text REtrieval Conference*, pages 109–126, 1994.

[19] A. Shakery and C. Zhai. Relevance propagation for topic distillation uiuc trec 2003 web track experiments. In *TREC*, pages 673–677, 2003.

[20] 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: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, New York, NY, USA, 2007. ACM.

[21] 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.

[22] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval.

[23] 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 31th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2008. ACM Press.

[24] 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.

[25] J. Yeh, J. Lin, H. Ke, and W. Yang. Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.

[26] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.