Learning to select a ranking function

Jie Peng, Craig Macdonald, and Iadh Ounis

Department of Computing Science,
University of Glasgow, G12 8QQ, UK
{pj, craigm, ounis}@dcs.gla.ac.uk

Abstract. Learning To Rank (LTR) techniques aim to learn an effective document ranking function by combining several document features. While the function learned may be uniformly applied to all queries, many studies have shown that different ranking functions favour different queries, and the retrieval performance can be significantly enhanced if an appropriate ranking function is selected for each individual query. In this paper, we propose a novel Learning To Select framework that selectively applies an appropriate ranking function on a per-query basis. The approach employs a query feature to identify similar training queries for an unseen query. The ranking function which performs the best on this identified training query set is then chosen for the unseen query. In particular, we propose the use of divergence, which measures the extent that a document ranking function alters the scores of an initial ranking of documents for a given query, as a query feature. We evaluate our method using tasks from the TREC Web and Million Query tracks, in combination with the LETOR 3.0 and LETOR 4.0 feature sets. Our experimental results show that our proposed method is effective and robust for selecting an appropriate ranking function on a per-query basis. In particular, it always outperforms three state-of-the-art LTR techniques, namely Ranking SVM, AdaRank, and the automatic feature selection method.

1 Introduction

The effective ranking of documents in search engines is based on various document features, such as the frequency of query terms in each document, the length of each document, or link analysis. In order to obtain a better retrieval performance, instead of using a single or a few features, there is a growing trend to create a ranking function by applying a Learning To Rank (LTR) technique on a large set of features [1–3].

Typically, a LTR technique learns a ranking function by assigning a weight to each document feature, then uses this obtained ranking function to estimate the relevance scores for each document [1, 4]. In recent years, many effective LTR techniques have been proposed to build such ranking functions, such as AdaRank [1], Ranking SVM [5, 6] or the Automatic Feature Selection (AFS) method [2]. Most of the current LTR literature mainly focuses on how to efficiently and effectively learn such ranking functions, but simply equally applies the learned ranking function to all queries. However, many studies have shown that different queries benefit differently from each ranking function [7–11] and

the retrieval performance can be significantly enhanced if an appropriate ranking function is used for each individual query.

In this paper, we propose the Learning To Select (LTS) framework for selectively applying an appropriate ranking function on a per-query basis. We believe that the effectiveness of a ranking function for an unseen query can be estimated based on similar training queries. A divergence measure can be used to determine the extent that a document ranking function alters the scores of an initial ranking of documents for a given query. We propose that this divergence can be used to identify similar training queries. In this case, a ranking function which performs well for training queries that have a similar divergence to the unseen query, will also perform well on the unseen query.

We conduct a comprehensive experimental investigation using query and relevance assessment sets from the TREC Web and Million Query tracks, in combination with the LETOR 3.0 and LETOR 4.0 feature sets [4]. Moreover, we use three state-of-the-art LTR techniques as our baselines, namely Ranking SVM, AdaRank, and the AFS method.

There are four key contributions from this work. First, we propose the novel LTS framework for selecting an appropriate ranking function on a per-query basis. This approach estimates the effectiveness of a ranking function for an unseen query based on the retrieval performance of this ranking function on already seen neighbour queries. Second, we propose the use of divergence, which measures the extent that a document ranking function alters the scores of an initial ranking of documents for a given query, as a query feature for identifying neighbour queries. Third, we show the effectiveness of our approach by comparing it to three state-of-the-art LTR techniques. Fourth, we show the robustness of our approach by selecting an appropriate ranking function from a large candidate set, which is created by using a LTR technique on different feature sets.

The remainder of this paper is organised as follows. Section 2 introduces the motivation of this paper. In Section 3, we describe several selective approaches presented in the literature. Section 4 describes our proposed LTS framework, which is used to select an appropriate ranking function for a given query. We present the experimental setup in Section 5, and analyse the experimental results in Section 6. Finally, we draw conclusions in Section 7.

2 Motivation

In this section, we provide an illustrative example showing the importance of the selective application of a ranking function. Table 1 shows the retrieval performance of three LTR techniques, namely Ranking SVM, AdaRank and the AFS method, on four different datasets[1]. Moreover, the upper bounds (denoted MAX) are achieved by manually selecting the most effective ranking function on a per-query basis. From this table, it is clear that the retrieval performance can be significantly enhanced if we apply the most appropriate ranking function for each query. This observation suggests that different ranking functions do favour different queries and that the appropriate selective application of a ranking function could enhance the retrieval performance.

[1] The detailed settings can be found in Section 5.

Table 1. MAX is the upper bound (100% correct per-query application). The highest score in each column is highlighted in bold and scores that are statistically better than $RankingSVM$, $AdaRank$, and AFS are marked with $\star$, $*$, and †, respectively (Wilcoxon matched-pairs signed-ranks test, $p < 0.05$).

	MAP			
	TREC2003	TREC2004	TREC2007	TREC2008
Ranking SVM	0.5366	0.4193	0.4641	0.4752
AdaRank	0.5977	0.5062	0.4537	0.4766
AFS	0.6145	0.5079	0.4596	0.4784
MAX	**0.6933** $\star * \dagger$	**0.5744** $\star * \dagger$	**0.5057** $\star * \dagger$	**0.5226** $\star * \dagger$

3 Related works

Some selective application techniques have been previously proposed in Information Retrieval IR [3, 8–11, 14, 15]. For example, in [3], Geng et al. proposed a query-dependent ranking approach. For each given query, they employ a specific ranking function, which is obtained by applying a LTR technique (e.g. Ranking SVM) on a training query set. This training query set is dependent on the given query, which can be identified by using a classification technique (K-nearest neighbour (KNN)) based on a query feature. The query feature used in their work is the mean of the document feature scores (e.g. $tf \cdot idf$) of the top retrieved documents, which can be obtained by a reference model (e.g. BM25), given as follows:

$$score(q, r_i) = \frac{\sum_{\tau=1}^{n} rel(d_\tau)}{n} \tag{1}$$

where n is the number of the top retrieved documents returned by ranking function r_i for a given query q. $rel(d_\tau)$ is the document relevance score of a document d at position τ of the document ranking list.

Extensive experiments on a large dataset, which was sampled from a commercial search engine, showed the effectiveness of the aforementioned approach [3]. However, [3] only investigated the selective application of a ranking function obtained from a single LTR technique and using a fixed set of document features. Hence, the effectiveness of the query-dependent ranking approach is not clear when there is more than one LTR technique and the number of features is varied. Moreover, the use of the mean of the feature scores from the top retrieved documents simply ignores the importance of the distribution of the feature scores, which has been effectively used in many retrieval applications [11, 12]. For example, Manmatha et al. [12] use the relevance score distribution to estimate the effectiveness of a search engine.

In [8], a query performance predictor was used to decide whether to apply collection enrichment for a given query. Generally speaking, query performance prediction relies on the statistics of the collection for a given query, such as query term frequency in the collection and the number of documents containing the query term. Hence, query performance predictors may not be applicable to the selective application of ranking functions, as these statistics are invariant to changes in the ranking function.

Plachouras et al. [9, 10] proposed a method to selectively apply an appropriate retrieval approach for a given query, which is based on a Bayesian decision mechanism. Features such as the link patterns in the retrieved document set and the occurrence of query terms in the documents were used to determine the applicability of the retrieval approaches. This method was shown to be effective when there were only two candidate retrieval approaches. However, the retrieval performance obtained using this method only improved slightly and actually decreased when more than two candidate retrieval approaches were used.

Peng et al. [11] select a single query-independent feature for a given query. Such a query-independent feature could be, for example, HostRank [13], PageRank or document length. However, current IR systems usually apply a large set of features in order to achieve a high retrieval performance, and it is not clear how to select multiple document features using this approach. Other approaches [14, 15] attempt to predict the type of the query (e.g. known-item search query, information seeking query), and from this, apply different retrieval approaches. However, the accuracy of state-of-the-art query type prediction is not high [16]. Moreover, queries of the same type may benefit from having different retrieval approaches applied [11].

In order to selectively apply an appropriate ranking functions from a large set of candidate ranking functions, in this paper, we propose the LTS framework, which will be presented in the following section. The proposed method is agnostic to the number of ranking functions, as well as to the type of the queries.

4 The Learning to Select Framework

In this section, we present a novel LTS framework for selectively applying an appropriate ranking function on a per-query basis. We first introduce the general idea of this framework, then provide a more detailed algorithm.

4.1 General Idea

A document ranking function created by a LTR technique is based on the assumption that the training dataset is representative of unseen queries. However, some queries may benefit from applying different ranking functions. We believe that the effectiveness of a ranking function for an unseen query can be estimated based on similar training queries. A divergence measure can be used to determine the extent that a document ranking function alters the scores of an initial ranking of documents. We propose that this divergence can be used to identify similar training queries. In this case, a ranking function which performs well for training queries that have a similar divergence to the unseen query, will also perform well on the unseen query.

4.2 Algorithm

Divergence Estimation In this work, the ranking function which is used to obtain the initial ranking of documents for a given query is called the *base ranking function* r_b. Other ranking functions which may be applied are called

candidate ranking functions r_i. They assign different document relevance scores to the same documents as were retrieved by r_b.

There are several different ways to estimate the divergence between two document score distributions that are obtained by using a base ranking function and a candidate ranking function. Two commonly used divergence measures are studied in this work, namely Kullback-Leibler (KL) [17] and Jensen-Shannon (JS) [18], given as follows:

$$KL(r_b||r_i, q) = \sum_{d=1}^{n} r_b(d) \cdot \log_2 \frac{r_b(d)}{r_i(d)} \tag{2}$$

$$\begin{aligned} JS(r_b||r_i, q) &= \frac{1}{2} \cdot KL(r_b||r_i, q) + \frac{1}{2} \cdot KL(r_i||r_b, q) \\ &= \sum_{d=1}^{n} r_b(d) \cdot \log_2 \frac{r_b(d)}{\frac{1}{2} \cdot r_b(d) + \frac{1}{2} \cdot r_i(d)} \end{aligned} \tag{3}$$

where for the top n retrieved documents of a given query q, $r_b(d)$ and $r_i(d)$ are the relevance scores of document d in the base ranking r_b and candidate ranking r_i, respectively.

It is easy to verify that adding a constant to r_i does not change the ranking position of each document in r_i, however, this does affect the divergence between r_b and r_i. In order to avoid the issue of translation invariance, we apply a score normalisation, which was proposed by Lee [19], on each document of the document rankings r_b and r_i:

$$r_N(d) = \frac{r(d) - r(min)}{r(max) - r(min)} \tag{4}$$

where $r(max)$ and $r(min)$ are the maximum and minimum document relevance scores that have been observed in the top retrieved documents from the input ranking r. $r(d)$ is the relevance score of document d in the input ranking.

Learning to Select We have shown how to estimate a divergence score between two rankings of documents. In this work, we consider the divergence score to be an example of a query feature. Next, we present how to use the query feature, such as the divergence score, to selectively apply an appropriate ranking function for a given query.

Initially, on a training dataset, we have a set of queries $Q = \{q_1, q_2, ..., q_m\}$ and a set of candidate ranking functions $R = \{r_1, r_2, ..., r_n\}$. For each query q_j, we use one of the above described divergence measures to estimate the divergence score $d(r_b||r_i, q_j)$ of each ranking function r_i from the base ranking function r_b. Note that one divergence score will be estimated for each ranking function on each query. For all training queries Q, each ranking function r_i's divergence scores set is denoted $\mathbf{d}(r_b||r_i, Q) = \{d(r_b||r_i, q_1), ..., d(r_b||r_i, q_m)\}$.

Next, in response to an unseen query q', for each ranking function r_i, we first estimate a divergence score $d(r_b||r_i, q')$, then employ KNN to identify the k nearest queries from $\mathbf{d}(r_b||r_i, Q)$ in a similar manner to [3] but using the divergence score. KNN is widely used for finding the closest objects in a metric

Table 2. Sample divergence scores and MAP evaluations for 5 training queries and 2 ranking functions.

	MAP		divergence	
q_ϕ	$E(q_\phi, r_1)$	$E(q_\phi, r_2)$	$d(r_b \| \| r_1, q_\phi)$	$d(r_b \| \| r_2, q_\phi)$
q_1	0.1	0.2	0.5	0.3
q_2	0.5	0.3	0.7	0.6
q_3	0.3	0.2	0.4	0.5
q_4	0.4	0.5	0.2	0.4
q_5	0.2	0.1	0.8	0.7

space when there is little or no prior knowledge about the distribution of the objects. Each identified neighbour corresponds to a training query q_ϕ. Let $E(q_\phi, r_i)$ be the outcome of an evaluation measure calculated on the ranking function r_i for the query q_ϕ. The effectiveness of ranking function r_i on this test query q' is predicted based on the performance of r_i on the neighbours of q', denoted $\sum_{\phi=1}^{k} E(q_\phi, r_i)$.

We apply the ranking function r_i for the query q' that has the highest retrieval performance on the set of k nearest neighbouring queries:

$$r_i^*(q') = \arg\max_{r_i} \frac{\sum_{\phi=1}^{k} E(q_\phi, r_i)}{k} \tag{5}$$

4.3 Example of LTS

Let us illustrate the LTS framework using an example. Assuming our training dataset has 5 queries, namely $Q = \{q_1, q_2, q_3, q_4, q_5\}$, and that we have two candidate ranking functions, namely $R = \{r_1, r_2\}$. The retrieval performance (e.g. MAP) of each ranking function and its divergence score for each training query are presented in Table 2 for a particular divergence measure.

Then, for an unseen query q', we estimate a divergence for each ranking function. For the purpose of our example, let $d(r_b||r_1, q') = 0.3$ and $d(r_b||r_2, q') = 0.6$, and let $k = 3$ in order to find the three nearest neighbouring queries. In this case, according to the difference between divergence scores, the nearest queries for ranking function r_1 are $\{q_1, q_3, q_4\}$, while for ranking function r_2 they are $\{q_2, q_3, q_5\}$. Therefore, for q', we apply r_1 as its mean retrieval performance for the nearest queries is higher than for r_2 ($\frac{0.1+0.3+0.4}{3} > \frac{0.3+0.2+0.1}{3}$).

5 Experimental Settings

In our experiments, we address three main research questions:

- Firstly, we test how effective our proposed LTS framework is for selecting an appropriate ranking function for a given query, by comparing it to three state-of-the-art LTR techniques.
- Secondly, as the number of candidate ranking functions increases, the selection becomes more challenging. To test how robust our proposed LTS framework is, we apply it on a larger number of candidate ranking functions.

– Thirdly, we test how important the query feature is for identifying neighbour queries, by investigating three different query features, namely KL divergence, JS divergence and the mean of the relevance scores, which has been shown to be effective in identifying neighbouring queries [3].

We conduct our experiments on two datasets, namely LETOR 3.0 and LETOR 4.0 [4]. The LETOR 3.0 dataset contains 64 different document features, including document length and HostRank [13], among others. The documents in LETOR 3.0 are sampled from the top retrieved documents by using BM25 on the .GOV corpus with the TREC 2003 and TREC 2004 Web track queries. In contrast, the LETOR 4.0 dataset contains 46 document features for documents similarly sampled from the .GOV2 corpus using the TREC 2007 and TREC 2008 Million Query track queries.

Many different LTR techniques have been proposed during the past few years. In this work, we employ three state-of-the-art LTR techniques, namely Ranking SVM [5, 6], AdaRank [1], and the AFS method [2]. In addition, the average of the retrieval performance of the candidate ranking functions that were created by the three LTR techniques is used as an additional baseline (denoted AS).

In our experiments, we use a 5-fold cross-validation process by separating each LETOR dataset into 5 folds of equal size. We iteratively test our LTS framework on one fold after training on the remaining four folds.

BM25 is used as our base ranking function as the features included in the LETOR datasets are computed over the top retrieved documents, which are sampled using BM25 [4]. The feature weights that are related with each candidate ranking function by using the AFS method are set by optimising Mean Average Precision (MAP) on the training dataset, using a simulated annealing procedure [20]. The number of top retrieved documents and the number of neighbours, namely n and k in Section 4, are also set by optimising MAP over the training dataset, using a large range of different value settings. We evaluate our experimental results using MAP, Precision at N, and normalised Discounted Cumulative Gain (nDCG). We report the obtained results and their analysis in the following section.

6 Results and Discussion

6.1 Effectiveness of our LTS framework

In order to test the effectiveness of our proposed method for selectively applying an appropriate ranking function for a given query, we compare it with the AS baseline and three state-of-the-art LTR techniques, namely Ranking SVM, AdaRank, and the AFS method, which are systematically applied to all queries.

Tables 3 & 4 present the evaluation of the retrieval performances obtained by using the three state-of-the-art LTR techniques and by applying our proposed LTS framework in terms of MAP, Precision at N and nDCG on the LETOR 3.0 and LETOR 4.0 datasets, respectively. The best retrieval performances for each evaluation measure on each different dataset are emphasised in bold. The α, β, γ and δ symbols indicate that the retrieval performance obtained by the best of our proposed LTS framework is significantly better than the Ranking

Table 3. Comparison between LTS and state-of-the-art LTR techniques using different evaluation measures on the LETOR 3.0 dataset. Results are the mean over 5 folds.

TREC 2003					
	MAP	P@5	P@10	nDCG@5	nDCG@10
Ranking SVM	0.5366 α	0.1811	0.1063	0.6450	0.6615
AdaRank	0.5977 β	0.1731	0.0974	0.6612	0.6679
AFS	<u>0.6145</u> γ	0.1777	0.1023	0.6766	0.6914
AS	0.5829 δ	0.1773	0.1020	0.6610	0.6736
LTS-Mean	0.6305 $*$	0.1771	0.1043	0.6776	0.7047
LTS-KL	0.6446 $*$	0.1794	0.1040	**0.6908**	**0.7059**
LTS-JS	**0.6483** $*$	**0.1843**	**0.1080**	0.6884	0.6959
TREC 2004					
	MAP	P@5	P@10	nDCG@5	nDCG@10
Ranking SVM	0.4193 α	0.2116	0.1427	0.5527	0.5659
AdaRank	0.5062 β	0.2124	0.1364	0.6053	0.6110
AFS	<u>0.5079</u> γ	0.2169	**0.1436**	0.6075	0.6186
AS	0.4778 δ	0.2136	0.1409	0.5885	0.5985
LTS-Mean	0.5158	0.2178	0.1427	0.6054	0.6145
LTS-KL	**0.5423** $*$	0.2124	0.1400	0.6152	0.6345
LTS-JS	0.5397 $*$	**0.2204**	0.1413	**0.6286**	**0.6365**

SVM, AdaRank, AFS and AS baselines, respectively, according to the Wilcoxon Matched-Pairs Signed-Ranks Test ($p < 0.05$). The best retrieval performance obtained by systematically applying a LTR technique on all queries is highlighted with underline. The $*$ symbol indicates that the retrieval performance obtained by using our proposed LTS framework is significantly better than the underlined score. $LTS-JS$, $LTS-KL$ and $LTS-Mean$ denote the application of the LTS framework by using JS divergence, KL divergence and the mean of the relevance scores as the query feature for identifying neighbour queries, respectively.

From the results in Tables 3 & 4, we observe that the best retrieval performance in each column is achieved by using our proposed LTS framework. The only exceptions are for the P@10 evaluation measure. However, in each case, the performance of LTS is close to the highest P@10. Note that all training was conducted using the MAP evaluation measure.

In particular, from the MAP column, the best retrieval performance obtained by using our proposed LTS framework makes improvements over all the LTR techniques and the AS baseline. Moreover, the improvements are statistically significant, e.g. on the TREC 2003 dataset: 0.5366 $\rightarrow$ 0.6483; 0.5977 $\rightarrow$ 0.6483; 0.6145 $\rightarrow$ 0.6483; and 0.5829 $\rightarrow$ 0.6483. Furthermore, by comparing our LTS framework with the best LTR technique, we note that the results obtained by using the LTS framework are significantly better in 3 out of 4 cases, e.g., on the TREC 2003 dataset: 0.6145 $\rightarrow$ 0.6305; 0.6145 $\rightarrow$ 0.6483; and 0.6145 $\rightarrow$ 0.6446.

The above observations suggest that our proposed LTS framework is effective in applying an appropriate ranking function on a per-query basis.

Table 4. Comparison between LTS and state-of-the-art LTR techniques using different evaluation measures on the LETOR 4.0 dataset. Results are the mean over 5 folds.

TREC 2007					
	MAP	P@5	P@10	nDCG@5	nDCG@10
Ranking SVM	0.4641 α	0.4113	**0.3844**	0.4120	0.4421
AdaRank	0.4537 β	0.3932	0.3686	0.3916	0.4223
AFS	0.4603 γ	0.4080	0.3753	0.4118	0.4378
AS	0.4594 δ	0.4042	0.3761	0.4051	0.4341
LTS-Mean	0.4637	0.4060	0.3760	0.4076	0.4352
LTS-KL	**0.4692**	0.4116	0.3795	0.4149	0.4419
LTS-JS	0.4676	**0.4132**	0.3783	**0.4182**	**0.4422**
TREC 2008					
	MAP	P@5	P@10	nDCG@5	nDCG@10
Ranking SVM	0.4752 α	0.3457	**0.2499**	0.4742	0.2296
AdaRank	0.4766 β	0.3419	0.2449	0.4701	0.2230
AFS	0.4784 γ	0.3480	0.2490	0.4761	0.2286
AS	0.4767 δ	0.3452	0.2480	0.4735	0.2281
LTS-Mean	0.4861 $*$	0.3465	0.2488	0.4747	0.2297
LTS-KL	0.4908 $*$	0.3485	0.2493	0.4794	0.2285
LTS-JS	**0.4911** $*$	**0.3488**	0.2494	**0.4813**	**0.2300**

6.2 Robustness of our LTS framework

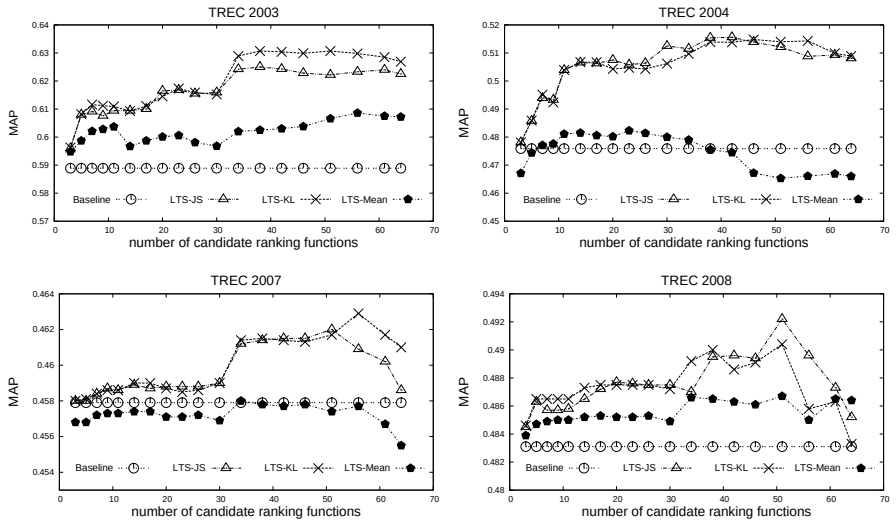
The analysis in Section 6.1 demonstrates the effectiveness of the proposed LTS framework on a small set of candidate ranking functions. In this section, we investigate the robustness of our LTS framework when the number of candidate ranking functions increases. To achieve this, we simulate a number of candidate ranking functions by applying a single LTR technique on several different combinations of document features. In particular, in order to have a strong baseline, we use AFS, since it produces on average higher retrieval performance than the other two LTR techniques.

In this investigation, we choose the best 6 features from the LETOR feature set, based on the training dataset. There are then $2^6 - 1$ possible combinations (excluding the empty combination). Integrating each combination into the base ranking function (namely BM25) produces one candidate ranking function. In this case, our task becomes to select an appropriate ranking function from a set of candidate ranking functions, which contains as many as $2^6 - 1 = 63$ candidate ranking functions.

Figure 1 shows the effect on MAP as the number of candidate ranking functions is varied, on different TREC datasets. We order the 63 ranking functions according to their performance on the training dataset and the best performing ranking function is used as our baseline. The x axis denotes the number of top performing ranking functions applied. For example, for TREC 2003, 10 in the x axis means we use the top 10 performing ranking functions, which are assessed on the training dataset, as candidate ranking functions to selectively apply on the test dataset.

From Figure 1, we can observe that both LTS-JS and LTS-KL consistently outperform the baseline while the LTS-Mean sometimes underperforms the base-

Fig. 1. MAP versus the number of candidate ranking functions on the LETOR 3.0 and LETOR 4.0 datasets.



line when the number of candidate ranking functions is increased. This suggests that the effectiveness of our LTS framework is related to the query feature that is used for identifying the neighbouring queries. A more detailed analysis of this point is presented in Section 6.3. In the remaining of this section, we only report the MAP distributions based on LTS-JS and LTS-KL as both of these proposed query features always perform better than LTS-Mean.

Indeed, in contrast to LTS-Mean, both LTS-JS and LTS-KL are enhanced as the number of candidate ranking functions increases from 2 to 35. This is mainly because each newly added ranking function (from 2 to 35) has different behaviour and favours different queries. Hence, with more ranking functions added, the retrieval performance can be further improved if they can be selectively applied in an appropriate manner.

However, the retrieval performance only improves slightly when the number of candidate ranking functions increases to 50. The reason for this is that these ranking functions are created based on the combination of a small set of features, most of them with similar behaviour, which results in these newly added ranking functions (from 35 to 50) favouring the same queries as the previously chosen ranking functions.

We also observe that the best retrieval performance is obtained when there are around 50 candidate ranking functions. However, the performance starts to decrease after 50 − as the last added ranking functions perform poorly on the training dataset and bring noise to the LTS framework.

This investigation suggests that our method is a robust approach, which can increasingly improve the retrieval performance as more ranking functions are added. The only caveat is when the most poorly-performing ranking functions are added to the candidate set. This can be controlled by setting the number of top-performing candidate ranking functions from which to select.

6.3 Importance of Query Feature

The above two investigations have shown the effectiveness and robustness of our proposed LTS framework. It is of note that the key component of the proposed LTS framework is the query feature that is used to identify the neighbouring queries, namely JS divergence, KL divergence, and the mean of the relevance scores.

By using different query features for identifying similar queries in our proposed LTS framework, from Tables 3 & 4, we observe that the JS divergence measure (LTS-JS) and the KL divergence measure (LTS-KL) are producing very close retrieval performances and both of them consistently outperform the mean of the relevance scores (LTS-Mean). For example, on the TREC 2003 dataset: $0.6305 \rightarrow 0.6483$; and $0.6305 \rightarrow 0.6483$.

In addition, from the distribution of MAP versus the number of candidate ranking functions shown in Figure 1, we observe that LTS-JS, LTS-KL and LTS-Mean have a similar distribution when increasing the number of candidate ranking functions. The only exception is on the TREC 2004 dataset (right top), where the MAP distribution obtained by LTS-Mean goes down quickly just after the number of candidate ranking functions reaches 30. In addition, LTS-Mean always underperforms compared to both LTS-JS and LTS-KL, and can fail to improve the baseline. This is particularly noticeable for the TREC 2007 (left bottom), where LTS-Mean always fails to improve over the baseline. Finally, we note that KL divergence and JS divergence have comparable performance, which is explained in that they are mathematically related.

The above observations suggest that our proposed use of divergence measures as a query feature for identifying neighbour queries is very effective.

7 Conclusion

In this paper, we proposed a novel Learning To Select (LTS) framework for selecting an appropriate ranking function for a given query. In particular, for an unseen query, we identify similar training queries by using novel query features based on the divergence between document score distributions. Such similar queries are then used to select an appropriate highly performing ranking function to apply. We tested our framework on the LETOR 3.0 & LETOR 4.0 feature sets and their corresponding TREC tasks, by comparing it with three state-of-the-art Learning To Rank (LTR) techniques, namely Ranking SVM, AdaRank, and the AFS method.

Our experimental results showed that the retrieval performance obtained by using our proposed LTS framework could constantly outperform the three state-of-the-art techniques using different evaluation measures and on different datasets, the only exception being for the $P@10$ measure. In addition, improvements over all LTR techniques were statistically significant in most cases.

Moreover, we investigated the effectiveness of our framework when the number of candidate ranking functions increases. By plotting the distribution of MAP versus the number of candidate ranking functions, we found that by using our proposed framework, the retrieval performance was enhanced when increasing the number of candidate ranking functions.

Furthermore, our proposed use of divergence measures as query features to identify neighbouring queries was always more effective than the mean of the relevance scores measure, which ignores the distribution of relevance scores. For our future work, we plan to investigate other query features.

References

1. Xu, J., Li, H.: AdaRank: A Boosting Algorithm for Information Retrieval. In *Proceedings of SIGIR'07*, Amsterdam, The Netherlands (2007).
2. Metzler, D.: Automatic Feature Selection in the Markov Random Field Model for Information Retrieval. In *Proceedings of CIKM'07*, Lisbon, Portugal (2007).
3. Geng, X., Liu, T.Y., Qin, T., Arnold, A., Li, H., Shum, H.Y.: Query Dependent Ranking Using K-Nearest Neighbour. In *Proceedings of SIGIR'08*, Singapore (2008).
4. Liu, T.Y., Qin, T., Xu, J., Xiong, W.Y., Li, H.: LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval, In *Proceedings of SIGIR'07 Learning to Rank workshop*, Amsterdam, The Netherlands (2007).
5. Herbrich, R., Graepel, T., Obermayer, K.: *Large Margin Rank Boundaries for Ordinal Regression*. MIT Press, Cambridge MA, USA (2000).
6. Joachims, T.: Optimizing Search Engines using Clickthrough Data. In *Proceedings of SIGKDD'02*, Alberta, Canada (2002).
7. Kamps, J., Mishne, G., de Rijke, M.: Language Models for Searching in Web Corpora. In *Proceedings of TREC 13*, Gaithersburg MD, USA (2004).
8. Peng, J., He, B., Ounis, I.: Predicting the Usefulness of Collection Enrichment for Enterprise Search. In *Proceedings of ICTIR'09*, Cambridge, UK (2009)
9. Plachouras, V., Ounis, I.: Usefulness of Hyperlink Structure for Query-Biased Topic Distillation. In *Proceedings of SIGIR'04*, Sheffield, UK (2004).
10. Plachouras, V.: Selective Web Information Retrieval. PhD thesis, University of Glasgow, UK (2006).
11. Peng, J., Ounis, I.: Selective Application of Query-Independent Features in Web Information Retrieval. In *Proceedings of ECIR'09*, Toulouse, France (2009).
12. Manmatha, R., Rath, T., Feng, F.: Modeling Score Distributions for Combining the Outputs of Search Engines. In *Proceedings of SIGIR'01*, New Orleans LA, USA (2001).
13. Xue, G.R., Yang, Q., Zeng, H.J., Yu, Y., Chen, Z.: Exploiting the Hierarchical Structure for Link Analysis. In *Proceedings of SIGIR'05*, Salvador, Brazil (2005).
14. Song, R., Wen, J.R., Shi, S., Xin, G., Liu, T.Y., Qin, T., Zheng, X., Zhang, J., Xue, G., Ma, W.Y.: Microsoft Research Asia at Web Track and Terabyte Track of TREC 2004. In *Proceedings of TREC'04*, Gaithersburg MD, USA, (2004).
15. Yang, K., Yu, N., Wead, A., La Rowe, G., Li, Y.H., Friend, C., Lee, Y.: WIDIT in TREC 2004 Genomics, Hard, Robust and Web Tracks. In *Proceedings of TREC'04*, Gaithersburg MD, USA (2004).
16. Craswell, N., Hawking, D.: Overview of the TREC 2004 Web Track. In *Proceedings of TREC'04*, Gaithersburg MD, USA (2004).
17. Kullback, S.: *Information Theory and Statistics*. John Wiley & Sons, New York NY, USA (1959)
18. Lin, J.: Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37(1):145-151 (1991).
19. Lee, J. H.: Analyses of Multiple Evidence Combination. In *Proceedings of SIGIR'97*, Philadelphia, USA (1997).
20. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by Simulated Annealing. *Science* 220(4598):671-680 (1983).