Learning to Search Web Pages with Query-Level Loss Functions

Tao QIN[2], Tie-Yan LIU[1], Ming-Feng Tsai[3], Xu-Dong ZHANG[2], Hang Li[1]

[1]Microsoft Research Asia, No.49 Zhichun Road, Haidian District, Beijing 100080, P.R. China

[2]Dept. Electronic Engineering, Tsinghua University, Beijing, 100084, P.R. China

[3]Dept. Computer Science and Information Engineering, National Taiwan University, Taiwan 106, ROC

[1]{tyliu, hangli}@microsoft.com

[2]tsintao@gmail.com, zhangxd@tsinghua.edu.cn

[3]mftsai@nlg.csie.ntu.edu.tw

Abstract

Many machine learning technologies such as Support Vector Machines, Boosting, and Neural Networks have been applied to the ranking problem in information retrieval. However, since originally the methods were not developed for this task, their loss functions do not directly link to the criteria used in the evaluation of ranking. Specifically, the loss functions are defined on the level of documents or document pairs, in contrast to the fact that the evaluation criteria are defined on the level of queries. Therefore, minimizing the loss functions does not necessarily imply enhancing ranking performances. To solve this problem, we propose using query-level loss functions in learning of ranking functions. We discuss the basic properties that a query-level loss function should have, and propose a query-level loss function based on the cosine similarity between a ranking list and the corresponding ground truth. We further design a coordinate descent algorithm, referred to as RankCosine, which utilizes the proposed loss function to create a generalized additive ranking model. We also discuss whether the loss functions of existing ranking algorithms can be extended to query level. Experimental results on the datasets of TREC web track, OSHUMED, and a commercial web search engine show that with the use of the proposed query-level loss function we can significantly improve ranking accuracies. Furthermore, we found that it is difficult to extend the document-level loss functions to query level loss functions.

Keywords

Information Retrieval, Learning to Rank, Query-level Loss Function, RankCosine

1. INTRODUCTION

Web search engines are changing people's life, and continuously enhancing the accuracy (relevance) of search also becomes an endless endeavor for IR (information retrieval) researchers. The key issue in web search is to construct a ranking function such that given a query the ranking function can rank the retrieved web pages in a way that can maximally satisfy users' search needs. Traditional approaches (Baeza-Yates & Ribeiro-Neto, 1999) resort to empirical methods in ranking model construction. These include content based methods such as BM25 (Robertson, 1997) and link based methods such as PageRank (Page, 1998). As more and more information (e.g., query log data) useful for search becomes available, the limitation of empirical tuning also becomes clearer, that is, it becomes very difficult, if not impossible, to tune the models with hundreds or thousands of features. The approach of employing machine learning techniques to address the problem naturally emerges as an effective solution and several methods have been proposed along the direction. Typical methods include RankBoost (Freund et al., 2003), Ranking SVM (Herbrich et al., 2000; Joachims, 2002), and RankNet (Burges et al., 2005), which are based on Boosting, Support Vector Machines and Neural Networks respectively. From the machine learning perspective, ranking, in which given a query and its associated documents we are to rank the documents as correctly as possible, also becomes a new branch of supervised learning, in addition to classification, regression, and density estimation (Vapnik, 1998).

However, it should be noted that the aforementioned machine learning methods were not proposed directly for IR, and therefore their loss functions are only associated to some extent with the evaluation criteria in IR, such as mean average precision (MAP) (Baeza-Yates & Ribeiro-Neto, 1999), mean precision at n (P@n) (Baeza-Yates & Ribeiro-Neto, 1999), and normalized discounted cumulative gain (NDCG) (Jarvelin & Kekalainen, 2000; Jarvelin & Kekalainen, 2002). All the IR criteria are on the query level; specifically, given two queries, no matter how different the numbers of documents retrieved for the two queries are, they contribute equally to the final performance evaluation. In contrast, the loss functions of the learning algorithms are defined on the level of documents (Nallapati, 2004) or document pairs (Burges et al., 2005; Freund et al., 2003; Herbrich et al., 2000; Joachims, 2002). Therefore, minimizing the loss functions does not necessarily lead to enhancing the accuracy in terms of the evaluation measures.

In order to solve this problem, we propose employing query-level loss functions in learning of ranking functions for IR.

In this paper, we first discuss what kind of properties a good query-level loss function should have. Then we propose a query-level loss function, *cosine loss*, as an example, which is based on the cosine similarity between a ranking list and the corresponding ground truth with respect to a given query. With the new loss function, we further derive a learning algorithm, *RankCosine*, which learns a generalized additive model as ranking function.

Next, we discuss whether it is possible to extend the document or document-pair level loss functions of the existing methods (Ranking SVM, RankBoost, and RankNet) to the query level.

We used two public datasets and one web search dataset to evaluate the effectiveness of our method. Experimental results show that the proposed query-level loss function is very effective for information retrieval. Furthermore, we found that it is in general difficult to extend the loss functions in the existing methods to the query level.

The rest of this paper is organized as follows. In Section 2, we give a brief review on related work. In Section 3, we justify the necessity of using query-level loss functions for IR and discuss the properties that a good query-level loss function should have. We then give an example of query-level loss function, cosine loss, and derive an efficient algorithm to minimize the loss function in Section 4. Experimental results are reported in Section 5. In Section 6, we discuss the possibility of extending the loss functions of the existing methods to the query level. Conclusions and future work are given in Section 7.

2. RELATED WORK

In recent years many machine learning technologies (Burges, et al., 2005; Crammer & Singer, 2002; Dekel et al., 2004; Freund et al., 2003; Herbrich et al., 2000; Joachims, 2002; Nallapati, 2004) were applied to the problem of ranking for information retrieval. Some early work simply tackled this problem as a binary classification problem (Nallapati, 2004), in which the assumption is made that a document is either relevant or irrelevant to the query , and the goal of learning is to classify relevant documents from irrelevant documents. However, in real-world applications, the degree of relevance of a document to a query can be discretized to multiple levels. For example, we can consider the use of three categories: highly relevant, partially relevant, and highly irrelevant. And so, ranking is a problem different from classification. To solve the problems, many other methods were proposed including Ranking SVM, RankBoost, and RankNet.

Herbrich et al. (2000) and Joachims (2002) took an SVM approach to learn a ranking function and proposed the Ranking SVM algorithm. The basic idea of Ranking SVM is the same as the conventional SVM: minimizing the sum of empirical loss and regularizer. The difference is that the constraints in Ranking SVM are defined on partial order relationships within document pairs. The optimization formulation of Ranking SVM is shown as follows[1]:

$$\begin{gathered} min\, V(\omega,\varepsilon) = \tfrac{1}{2}\omega^T\omega + C\sum_{i,j,q}\varepsilon_{i,j,q} \\ s.t.: \forall(d_i,d_j) \in r_1^* :\ \omega\varphi(q_1,d_i) \geq \omega\varphi(q_1,d_j) + 1 - \varepsilon_{i,j,1} \\ \cdots \\ \forall(d_i,d_j) \in r_n^* :\ \omega\varphi(q_n,d_i) \geq \omega\varphi(q_n,d_j) + 1 - \varepsilon_{i,j,n} \end{gathered} \tag{1}$$

where C is a parameter which controls the trade-off between empirical loss and regularizer, $\varphi(q,d_i)$ is the feature vector calculated from document d_i and query q, and the constraint $\omega\varphi(q,d_i) > \omega\varphi(q,d_j)$ means that document d_i is more relevant than document d_j with respect to query q. Theoretically, Ranking SVM is well-formed in the framework of structural risk minimization, and empirically the effectiveness of Ranking SVM has been verified in various experiments. Modifications of Ranking SVM for information retrieval (Cao et al., 2006; Qin et al., 2007) have also been proposed.

Freund *et al* (2003) adopted the Boosting approach to ranking and proposed the RankBoost algorithm. Similarly to Ranking SVM, RankBoost operates on document pairs. Suppose $d_i \succ_q d_j$ denotes that document d_i should be ranked higher than d_j for query q. Consider the use of model f, where $f(\varphi(q,d_i)) > f(\varphi(q,d_j))$ means that the model asserts $d_i \succ_q d_j$. Then the loss for a document pair in RankBoost is defined as

$$L(d_i \succ_q d_j) = e^{-\left(f(\varphi(q,d_i)) - f\left(\varphi(q,d_j)\right)\right)} \tag{2}$$

Consequently, the total loss on training data in RankBoost is defined as the sum of losses from all document pairs:

$$L = \sum_q \sum_{d_i \succ_q d_j} L(d_i \succ_q d_j) \tag{3}$$

The advantages of RankBoost include that it is easy to implement the algorithm and it is possible to run the algorithm in parallel. The effectiveness of RankBoost has also been verified.

[1] For details, please refer to Joachims (2002).

Neural networks have also been applied to ranking recently. Burges et al (2005) proposed the RankNet algorithm, in which relative entropy is used as loss function and neural network is used as the underlying ranking function. Similarly to Ranking SVM and RankBoost, training samples of RankNet are also document pairs. Let us denote the modeled posterior $P(d_i \succ_q d_j)$ as P_{ij}, and let us denote $\overline{P_{ij}}$ be the true value of the posterior, and[2] $o_{q,i,j}=f(\varphi(q,d_i))-f(\varphi(q,d_j))$. Then the loss for a document pair in RankNet is defined as follows.

$$L_{q,i,j} \equiv L(o_{q,i,j}) = -\overline{P_{ij}} log P_{ij} - (1-\overline{P_{ij}})\log(1-P_{ij})$$

$$= -\overline{P_{ij}} o_{q,i,j} + \log(1+e^{o_{q,i,j}}) \quad (4)$$

Similarly, the total loss in RankNet is defined as the sum of all document pairs

$$L = \sum_q \sum_{i,j} L_{q,i,j} \quad (5)$$

RankNet has been successfully applied to web search. Further improvements on RankNet can be found in (Matveeva et al. 2006, Tsai et al. 2007, and Cao et al. 2007).

One major problem Ranking SVM, RankBoost, and RankNet have is that the loss functions used are not in accordance with the IR evaluation measures. We will elaborate on this in more details in the next section.

3. QUERY-LEVEL LOSS FUNCTIONS FOR INFORMATION RETRIEVAL

Let us first use Table 1 to summarize the loss functions in the existing algorithms described in Section 2. In the classification approach (Nallapati, 2004), the loss function is defined on the document level. The loss functions of Ranking SVM, RankBoost, and RankNet are defined on the document-pair level. These loss functions can model the partial order relationship within a pair of documents, but not the total order relationship between all the documents. In this regard, these loss functions are not in accordance with the evaluation criteria for IR such as MAP and NDCG which are defined on the query level. This motivates us to propose the use of query-level loss functions, as will be discussed below.

Table 1 Loss functions for web search

Loss Function	Algorithms
Document-level	Binary classification (Nallapati, 2004)

[2] The definitions of f and φ can be found in the conventional studies on Ranking SVM and RankBoost.

Pairwise	Ranking SVM (Cao et al., 2006; Herbrich et al., 2000; Joachims, 2002) RankBoost(Freund et al., 2003) RankNet(Burges et al., 2005)
Query-level	Our work

3.1 Why Is Query-Level Loss Function Needed

As mentioned above, the loss functions in Ranking SVM, RankBoost, and RankNet are not in accordance with the evaluation criteria in IR. This may penalize the accuracies of the learning algorithms.

Let us consider a simple example. Suppose there are two queries q_1 and q_2 with 40 and 5 documents respectively. In the extreme case of using complete partial-order document pairs for training, we can get $40 \times 39/2=780$ pairs for q_1 and only $5 \times 4/2=10$ pairs for q_2. If a learning algorithm can rank all document pairs correctly, then there will be no problem. However, if this is not the case, for example, we can only rank 780 out of the 790 pairs correctly, then a problem will arise. With the pairwise loss function, the losses will be the same if the learning algorithm correctly ranks all pairs of q_2 but only 770 pairs of q_1, or correctly ranks all pairs of q_1 but no pairs of q_2. However, for these two cases, the performances based on a query-level evaluation criterion will be completely different. As shown in Table 2, case 1 is much better than case 2. This example indicates that *using a document-level loss function is not suitable for IR*. Actually only when all the queries have the same number of document pairs for training, the document-level loss function and the query-level loss can lead to the same result. However, this assumption does not hold in real-world scenarios. Therefore it is better to define loss function on the query level when training ranking functions.

Table 2 Document-pair level loss v.s. query-level loss

		Case 1	Case 2
Document pairs of q_1	correctly ranked	770	780
	wrongly ranked	10	0
	Accuracy	98.72%	100%
Document pairs of q_2	correctly ranked	10	0
	wrongly ranked	0	10
	Accuracy	100%	0%
overall accuracy	Document-pair level	98.73%	98.73%

	query level	**99.36%**	**50%**

3.2 What Is a Good Loss Function

One may ask what properties a good query-level loss function should have. Here we list some properties, and discuss the necessities.

We first give explanations on the notations. Suppose $n(q)$ denote the number of documents for a query q to be ranked. There are in total $n(q)!$ possible ranking lists for query q in total. Suppose Q is the space of queries, and $\mathcal{F}$ is the space of ranking functions. We denote the ground truth ranking list of query q as $\tau_g(q)$, and the ranking list generated by a ranking function $f \in \mathcal{F}$ as $\tau_f(q)$. Then a query level loss function is a function of two ranking lists to a non-negative real number. That is,

$$L\big(\tau_g(q), \tau_f(q)\big) \geq 0$$

1) Insensitive to number of document pairs

This has been made clear through the example in Table 2. In this regard, the loss functions of Ranking SVM, RankBoost and RankNet are not good loss functions.

This property can be expressed formally as:

Property 1:

Define $L_k = \sup_{f\in\mathcal{F}, q\in Q \text{ and } n(q)=k} L\big(\tau_g(q), \tau_f(q)\big)$ for a query level loss function L, for any $k1 > 1, k2 > 1$, if $L_{k1} > 0$ *and* $L_{k2} > 0$, then there should exist a constant C, which makes the loss function L satisfy

$$\frac{L_{k1}}{L_{k2}} < C, and\ \frac{L_{k2}}{L_{k1}} < C$$

2) Important to rank top results correctly

In contemporary IR, precision is often considered more important than recall, because information to be searched is usually abundant . Many IR evaluation criteria embody the requirement of conducting accurate ranking on the top of

lists. For example, a ranking error at the 5th position is more harmful than a ranking error at the 95th position. A good loss function should also reflect this property.

We give a formal definition of this property:

Property 2:

Suppose the ground truth rank list of query q is

$$\tau_g(q) = \{d_1^{(1)} \succ \cdots \succ d_{i-j}^{(i-j)} \succ \cdots \succ d_i^{(i)} \succ \cdots \succ d_{i+j}^{(i+j)} \succ \cdots \succ d_{n(q)}^{(n(q))}\}$$

where $d_i^{(j)}$ means document d_i is ranked at position j

$\tau_{f1}(q)$ and $\tau_{f2}(q)$ are two ranking lists generated by ranking functions $f1$ and $f2$

$$\tau_{f1}(q) = \{d_1^{(1)} \succ \cdots \succ d_i^{(i-j)} \succ \cdots \succ d_{i-j}^{(i)} \succ \cdots \succ d_{i+j}^{(i+j)} \succ \cdots \succ d_{n(q)}^{(n(q))}\}$$

$$\tau_{f2}(q) = \{d_1^{(1)} \succ \cdots \succ d_{i-j}^{(i-j)} \succ \cdots \succ d_{i+j}^{(i)} \succ \cdots \succ d_i^{(i+j)} \succ \cdots \succ d_{n(q)}^{(n(q))}\}$$

Then a good query level loss function L should satisfy

$$L\big(\tau_g(q), \tau_{f1}(q)\big) \geq L\big(\tau_g(q), \tau_{f2}(q)\big)$$

The loss functions of Ranking SVM, RankBoost and RankNet have a similar tendency. The reason is that if a definitely irrelevant document is ranked high, it will violate many constraints regarding to ranking of document pairs, while if it is ranked at the middle of the list, the number of constraints violated will be reduced.

3) Upper bound

Query-level loss function should not be easily biased by difficult queries. For this purpose, a natural requirement is that the loss for each query should have an upper bound. Otherwise, queries with extremely large losses will dominate the training process.

The formal definition of this property is

Property 3:

For any $f \in \mathcal{F}, q \in Q$, there should exist a constant C, such that

$$0 \leq L\big(\tau_g(q), \tau_f(q)\big) \leq C$$

We can see that the document-pair level loss functions of RankBoost, Ranking SVM, and RankNet do not have an upper bound. For RankBoost, because the exponential loss is used, the value of the loss function can be extremely large. We can make a similar conclusion for the hinge loss of Ranking SVM. For RankNet, the loss function does not have an upper bound either, according to the analysis in (Burges et al. 2005).

4. RANKCOSINE

In this section, we propose a new loss function which retains all the aforementioned properties.

4.1 Cosine Loss

We first give explanations on the notations.

Suppose there are $n(q)$ documents for query q, and the ground-truth ranking list for this query is $\boldsymbol{g}(q)$, where $\boldsymbol{g}(q)$ is a $n(q)$-dimension vector, and the k-th element in this vector is the rating (level of relevance) of the k-th document, given by humans. The absolute value of a score is not very important, it is really the difference between scores that matters.

Let us denote the output of a learning machine for query q as $\boldsymbol{H}(q)$. Similarly to $\boldsymbol{g}(q)$, $\boldsymbol{H}(q)$ is also an $n(q)$-dimension vector, and the k-th element in it is the output of the k-th document given by the learning machine.

Let $\phi(x)$ and $\psi(x)$ be two order preserving mapping functions. That is, if $x > y$, then we have $\phi(x) > \phi(y)$ and $\psi(x) > \psi(y)$. $\psi(x)$ is used to mapping the ground truth score to a real number, and $\phi(x)$ is used to mapping ranking score outputted by a ranking model to a real number. These two functions play a role of generalizing the feasibility of Cosine loss. Note that we can have many choices for the mapping functions, such as linear function

$$\phi(x) = ax + b, \forall a > 0$$

exponential function

$$\phi(x) = \exp(ax), \forall a > 0$$

or sigmoid function

$$\phi(x) = \frac{\exp(ax)}{1+\exp(ax)}, \forall a > 0.$$

Next, we define the ranking loss for query q as follows

$$L\big(\boldsymbol{g}(q), \boldsymbol{H}(q)\big) = \frac{1}{2}(1 - \cos(\boldsymbol{\psi}(\boldsymbol{g}(q)), \phi(\boldsymbol{H}(q))))$$

$$= \frac{1}{2}\left(1 - \frac{\boldsymbol{\psi}\big(\boldsymbol{g}(q)\big)^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{H}(q))}{\|\boldsymbol{\psi}(\boldsymbol{g}(q))\|\|\boldsymbol{\phi}(\boldsymbol{H}(q))\|}\right) \quad (6)$$

where ||.|| is L-2 norm of a vector. Since we use cosine similarity in (6), this loss function is referred to as cosine loss.

The goal of learning then turns out to minimize the total loss function over all training queries

$$L(\boldsymbol{H}) = \sum_{q\in Q} L\big(\boldsymbol{g}(p), \boldsymbol{H}(q)\big) \quad (7)$$

where the loss function is defined in (6).

Now we show that the cosine loss has all the three properties defined above.

Firstly, since

$$-1 \le \frac{\boldsymbol{\psi}\big(\boldsymbol{g}(q)\big)^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{H}(q))}{\|\boldsymbol{\psi}(\boldsymbol{g}(q))\|\|\boldsymbol{\phi}(\boldsymbol{H}(q))\|} \le 1$$

we have $0 \le L\big(\boldsymbol{g}(p), \boldsymbol{H}(q)\big) \le 1$, which is independent from the number of documents for query q. The upper bound of cosine loss is

$$L_k = \sup_{\boldsymbol{H}\in\mathcal{F}, q\in Q \text{ and } n(q)=k} L\big(\tau_{\boldsymbol{g}}(q), \tau_{\boldsymbol{H}}(q)\big) = 1$$

For any *k1*>0 and *k2*>0, we get

$$\frac{L_{k1}}{L_{k2}} = \frac{L_{k2}}{L_{k1}} = 1$$

Therefore, Property 1 is satisfied with $\forall C > 1$ for the cosine loss.

Secondly, we can put more emphasis on training of top results by setting an appropriate ground truth label. For example, we can set the scores of ground truth using an exponential function. Specifically we set the ground truth score of top 1 document as e^{-1}, and the ground truth score of the document at position i as e^{-i}.

Thirdly, the cosine loss function has an explicit lower bound and upper bound:

$$0 \le L\big(\boldsymbol{g}(p), \boldsymbol{H}(q)\big) \le 1 \tag{8}$$

4.2 Learning Algorithm

In this subsection, we explain how to optimize the cosine loss function. Here we consider a simple case for the two mapping functions:

$$\psi(x) = x, \phi(x) = x.$$

We choose to employ a generalized additive model as the final ranking function:

$$\boldsymbol{H}(q) = \sum_{t=1}^{T} \alpha_t \boldsymbol{h}_t(q) \tag{9}$$

where α_t is a combination coefficient, $\boldsymbol{h}_t(q)$ is a weak learner which maps an input matrix (a row of this matrix is the feature vector of a document) to an output vector (an element of this vector is the score of a document) and d is the dimension of feature vector:

$$\boldsymbol{h}_t(q)\colon R^{n(q)\times d} \to R^{n(q)}$$

With the use of the additive model and the cosine loss function, we can derive the learning process as follows. For simplicity, we assume the ground truth for each query has already been normalized:

$$\boldsymbol{g}(q) = \frac{\boldsymbol{g}(q)}{\|\boldsymbol{g}(q)\|}$$

Since $\psi(x) = x, \phi(x) = x$, we can re-write (6) as

$$L\big(\boldsymbol{g}(p), \boldsymbol{H}(q)\big) = \frac{1}{2}\Big(1 - \frac{\boldsymbol{g}(q)^{\boldsymbol{T}}\boldsymbol{H}(q)}{\|\boldsymbol{H}(q)\|}\Big) \tag{10}$$

We employ a stage-wise greedy search strategy, used in the Boosting algorithms (Friedman et al., 1998), to train the parameters in the ranking function. Let us denote $\boldsymbol{H}_k(q)$ as

$$\boldsymbol{H}_k(q) = \sum_{t=1}^{k} \alpha_t \boldsymbol{h}_t(q)$$

where $\boldsymbol{h}_t(q)$ is a weak learner[3] at the t-th step. Then the total loss of $\boldsymbol{H}_k(q)$ over all queries becomes

$$L(\boldsymbol{H}_k) = \sum_q \frac{1}{2}\Big(1 - \frac{\boldsymbol{g}(q)^{\boldsymbol{T}}\boldsymbol{H}_{\boldsymbol{k}}(q)}{\|\boldsymbol{H}_{\boldsymbol{k}}(q)\|}\Big) \tag{11}$$

[3] Here one can choose different ways to define the weak learners. For example, we can take the same approach as in RankBoost.

Given $\boldsymbol{H}_{k-1}(q)$ and $\boldsymbol{h}_k(q)$, (11) can be re-written as

$$L(\boldsymbol{H}_k) = \sum_q \frac{1}{2}\left(1 - \frac{\boldsymbol{g}(q)^{\boldsymbol{T}}(\boldsymbol{H}_{\boldsymbol{k-1}}(q)+\alpha_k \boldsymbol{h}_k(q))}{\sqrt{(\boldsymbol{H}_{\boldsymbol{k-1}}(q)+\alpha_k \boldsymbol{h}_k(q))^{\boldsymbol{T}}(\boldsymbol{H}_{\boldsymbol{k-1}}(q)+\alpha_k \boldsymbol{h}_k(q))}}\right) \tag{12}$$

Setting the derivative of $L(\boldsymbol{H}_k)$ with respect to α_k to zero, with some relaxation, we can get the optimal value of α_k as follows,

$$\alpha_k = \frac{\sum_q \boldsymbol{W}_{1,k}^T(q)\boldsymbol{h}_k(q)}{\sum_q \boldsymbol{W}_{2,k}^T(q)\left(\boldsymbol{h}_k(q)\boldsymbol{g}^{\boldsymbol{T}}(q)\boldsymbol{h}_k(q) - \boldsymbol{g}(q)\boldsymbol{h}_k^T(q)\boldsymbol{h}_k(q)\right)} \tag{13}$$

where $\boldsymbol{W}_{1,k}(q)$ and $\boldsymbol{W}_{2,k}(q)$ are two $n(q)$-dimension weight vectors with the following definitions.

$$\boldsymbol{W}_{1,k}(q) = \frac{\boldsymbol{g}^{\boldsymbol{T}}(q)\boldsymbol{H}_{k-1}(q)\boldsymbol{H}_{k-1}(q) - \boldsymbol{H}_{k-1}^T(q)\boldsymbol{H}_{k-1}(q)\boldsymbol{g}(q)}{\|\boldsymbol{H}_{k-1}(q)\|^{3/2}} \tag{14}$$

$$\boldsymbol{W}_{2,k}(q) = \frac{\boldsymbol{H}_{k-1}(q)}{\|\boldsymbol{H}_{k-1}(q)\|^{3/2}} \tag{15}$$

With (13) and (12), we can calculate the optimal weight α_k, evaluate the cosine loss for each weak learner candidate, and select the one with the smallest loss as the k-th weak learner. In this way, we can get a sequence of weak learners and their combination coefficients, and thus the final ranking function.

We summarize the details in Fig. 1. Note that $\boldsymbol{e}_{n(q)}$ is an $n(q)$ dimensional vector with all elements being 1. The time complexity of RankCosine is $O(m \cdot k \cdot T \cdot n_{max})$, where m denotes number of training queries, k denotes number of weak learner candidates, T denotes number of iterations, and n_{max} denotes maximum number of documents per query. RankBoost also adopts a Boosting algorithm for learning of ranking function and the time complexity of RankBoost is $O(m \cdot k \cdot T \cdot n^2_{max})$. It is easy to see that the complexity of RankCosine is much lower than that of RankBoost.

Algorithm: **RankCosine**

Given: ground truth $\boldsymbol{g}(q)$ over Q, and weak learner candidates $\boldsymbol{h}_i(q)$, i=1,2, …

Initialize: $\boldsymbol{W}_{1,1}(q) = \boldsymbol{W}_{2,1}(q) = \frac{\boldsymbol{e}_{n(q)}}{n(q)}$

For t=1,2, …, T

(a) For each weak learner candidate $\boldsymbol{h}_i(q)$

(a.1) Compute optimal $\alpha_{t,i}$ with (13)

(a.2) Compute the cosine loss $\varepsilon_{t,i}$ with (12)

(b) Choose weak learner $\boldsymbol{h}_{t,i}(q)$ having minimal loss as $\boldsymbol{h}_t(q)$

(c) Choose coefficient $\alpha_{t,i}$ as α_t

(d) Update query weight vectors $\boldsymbol{W}_{1,k}(q)$ and $\boldsymbol{W}_{2,k}(q)$ with (14) and (15)

Output the final ranking function $\boldsymbol{H}(q) = \sum_{t=1}^{T} \alpha_t \boldsymbol{h}_t(q)$

Fig. 1 The RankCosine Algorithm

5. EXPERIMENTS

To verify the benefit of using query-level loss functions, we conducted experiments on three data sets. We describe the details in this section.

5.1 Settings

We adopted three widely used evaluation criteria in our experiments: mean precision at n (P@*n*) (Baeza-Yates & Ribeiro-Neto, 1999), mean average precision (MAP) (Baeza-Yates & Ribeiro-Neto, 1999), and normalized discount cumulative gain (NDCG) (Borlund, 2003; Burges et al., 2005; Jarvelin & Kekalainen, 2002; Sormunen, 2002). All of them are widely used in IR.

In our experiments, we selected three machine learning algorithms for comparison: RankBoost, RankNet, and Ranking SVM. For RankBoost, the weak learners had a binary output of 0 or 1. For RankNet, we used a linear neural net and a two-layer net (Burges et al., 2005), which are referred to as Linear-RankNet and TwoLayer-RankNet respectively. For Ranking SVM, we used the tool of SVMlight (Joachims, 1999; Joachims, 2002). For RankCosine, each weaker learner was defined as a feature, taking continuous values from [0,1]. In order to make a comparison with traditional IR approaches, we also chose BM25 (Robertson, 1997) as baseline.

5.2 Experiments with TREC Web Track Data

We tested the performance of RankCosine using the data set from the TREC web track (Voorhees & Harman, 2005).

5.2.1 Data set

The TREC Web Track (2003) data set contains web pages crawled from the .gov domain in early 2002. There are totally 1,053,110 pages. The query set is from the topic distillation task (Craswell et al., 2003) and there are in total 50 queries. The ground truths of this task are provided by the TREC committee as binary judgments: relevant, or irrelevant. The number of relevant pages per query ranges from 1 to 86.

We mentioned that usually the number of documents can vary largely according to queries and this phenomenon can be verified with this dataset. For instance, from Figure 2, we can see that the number of relevant documents per query has a non-uniform distribution: about two thirds queries have less than 10 relevant documents. If we use a document-pair level loss function, two thirds of the queries will be penalized. In other words, two thirds of the queries will not contribute to the learning process as much as they should.

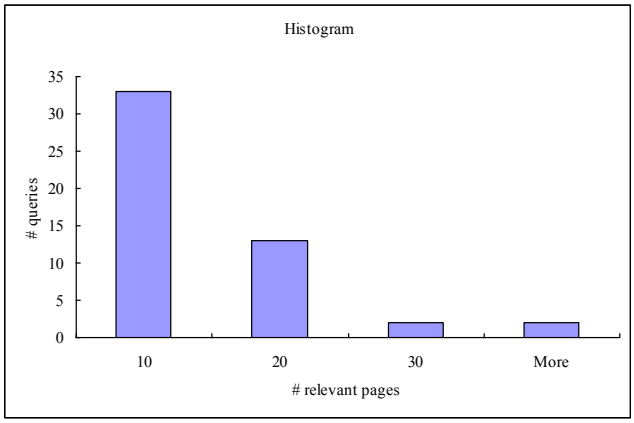


Fig. 2 Histogram of number of relevant pages per query

We extracted 14 features from each document for the learning algorithms. These features include content based features (BM25, MSRA1000), web structure based features (PageRank, HostRank), and their combinations (relevance propagation features). Some of them are traditional features (BM25, PageRank) and some are new features (HostRank, relevance propagation features). The details of the features are described in Table 3:

Table 3 Extracted features for TREC data

Name	Number
BM25(Robertson, 1997)	1
MSRA1000 (Song et al., 2004)	1
PageRank (Page et al., 1998)	1

HostRank (Xue et al., 2005)	1
Sitemap based Score Propagation (Qin et al., 2005)	2
Sitemap based Term Propagation (Qin et al., 2005)	2
Hyperlink based Score Propagation (Qin et al., 2005)	3
Hyperlink based Term Propagation (Qin et al., 2005)	3

5.2.2 Results

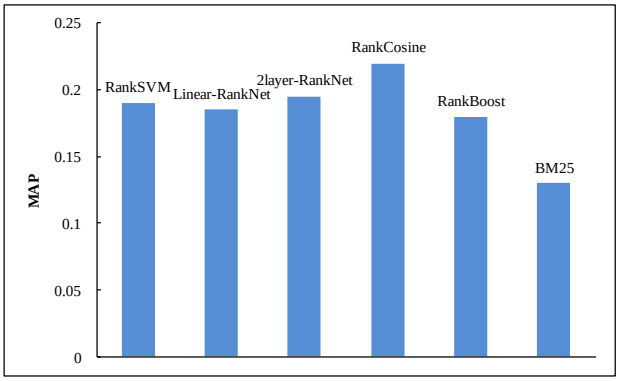
We conducted 4-fold cross validations for the learning algorithms. We tuned the parameters of BM25 in one trial and applied them to the other trials. The results reported in Figure 3 are those averaged over four trials.

From Figure 3(a), we can see that RankCosine outperforms all the other algorithms in terms of MAP, while the other learning algorithms perform similarly. This may imply that the three state-of-the-art algorithms (Ranking SVM, RankBoost, and RankNet) have similar learning abilities for information retrieval. Note all the learning algorithms outperform BM25. The MAP value of BM25 is about 0.13, which is comparable to the value reported in (Qin et al., 2005). RankBoost gets the lowest MAP value of 0.18 among all the learning algorithms, and this is still much higher than BM25. RankCosine, which obtains an MAP value of 0.21, improves upon BM25 for about 70%. This suggests that learning to rank is a promising approach for search, as it can leverage the information from various features.

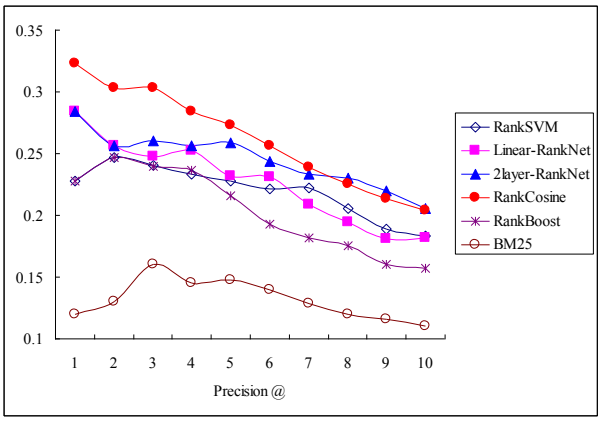
From Figure 3(b), we can see that RankCosine outperforms all the other algorithms, from P@1 to P@7. From P@8 to P@10, RankCosine is still much better than most of the other algorithms, except TwoLayer-RankNet. An interesting phenomenon is that RankCosine achieves more improvements at top when compared with the other algorithms, for example, more than 4 precision point[4] improvement for P@1, about 2 precision point improvements for P@5. Since correctly conducting ranking on the top is more important, this tendency is desirable anyway.

Figure 3(c) shows the result in terms of NDCG, and again RankCosine can work better than the other algorithms.

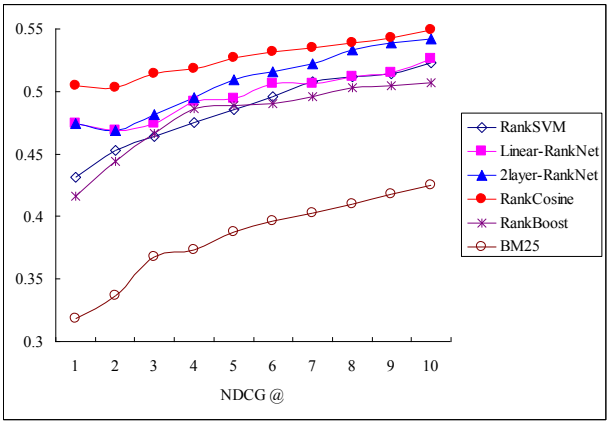
[4] We define a precision point by the precision score of 0.01



(a)



(b)



(c)

Fig. 3 Ranking accuracy on TREC web track data. (a) MAP, (b) P@*n*, (c) NDCG@*n*

5.3 Experiments with OHSUMED Data

We also conducted experiments with the OHSUMED data (Hersh et al., 1994), which has been used in many experiments in IR (Cao et al., 2006; Herbrich et al., 2000; Robertson & Hull, 2000).

5.3.1 Data

OHSUMED is a data set of documents and queries on medicine, consisting of 348,566 references and 106 queries. There are in total 16,140 query-document pairs upon which relevance judgments are made. Different from the TREC data, this data set has three levels of relevance judgments: "definitely relevant", "possibly relevant",and "not relevant".

We adopted 30 features, similar to those defined in (Nallapati, 2004). Table 4 shows some examples of the features. They include *tf* (term frequency), *idf* (inverse document frequency), *dl* (document length), and their combinations. BM25 score is another feature, as proposed in (Robertson, 1997). We took log on the feature values to re-scale them. This does not change the tendencies of the results, according to our preliminary experiments. Stop words were removed and stemming was conducted in indexing and retrieval. Note the features of this data set are different from those of the TREC data, since OHSUMED is a text document collection without hyperlink.

When calculating MAP, we defined the category of "definitely relevant" as positive and the other two categories as negative.

Table 4 Example features. $c(w, d)$ represents frequency of word w in document d; C represents document collection; n denotes number of terms in query; $|.|$ denotes size of function; and $idf(.)$ denotes inverse document frequency.

Features	
$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i, C)} + 1\right)$
$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, d)}{\lvert d \rvert} idf(q_i) + 1\right)$	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, d)}{\lvert d \rvert} + 1\right)$
$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, d)}{\lvert d \rvert} \cdot \frac{\lvert C \rvert}{c(q_i, C)} + 1\right)$	$\sum_{q_i \in q \cap d} \log(idf(q_i))$
$\log(\text{BM25 score})$	

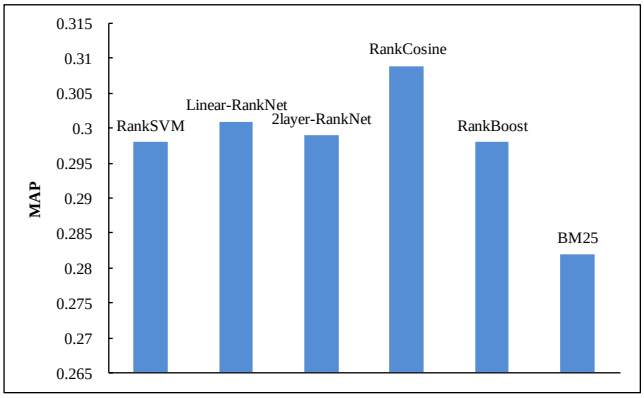
5.3.2 Results

In this experiment, we also conducted 4-fold cross validation for learning algorithms, and tuned the parameters for BM25 in one trial and applied them to the other trials. The results reported in Figure 4 are those averaged over four trials. From the figure, we can see that RankCosine outperforms all the other algorithms, from NDCG@1 to NDCG@10. On the other hand, the performances of the three learning methods (RankNet, Ranking SVM, and RankBoost) are similar. Therefore, we can draw the same conclusion as in the previous experiment.

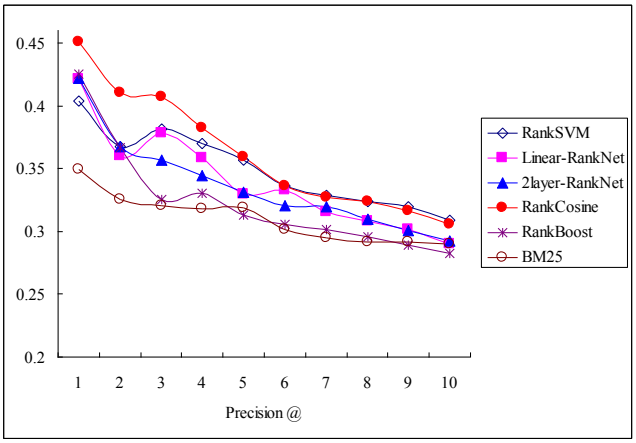
Comparing Fig 3(b) with 3(c), Fig 4(b) with 4(c), we may observe the differences between the corresponding evaluation criteria. P@*n* only considers the number of relevant documents at top *n* positions, and ignores the distribution of relevant documents. For example, the following two rank lists get the same P@4 value. Different from P@*n*, NDCG@*n* is sensitive to the ranked positions of relevant documents. For example, the NDCG@4 value of list B is much higher than that of list A.

A: {*irrelevant, irrelevant, relevant, relevant, …*}

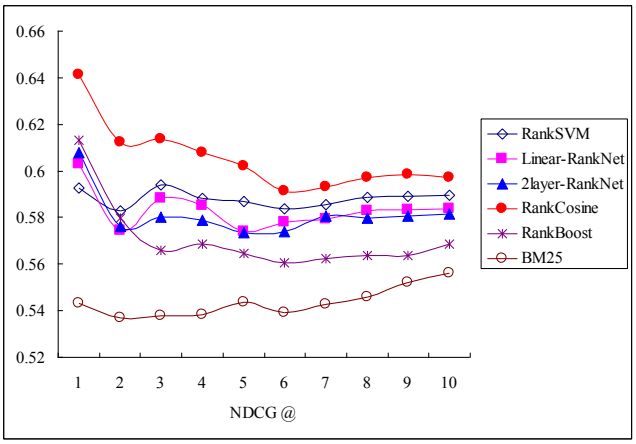
B: {*relevant, relevant, irrelevant, irrelevant, …*}



(a)



(b)



(c)

Fig. 4 Ranking accuracy on OHSUMED data. (a) MAP, (b) P@*n*, (c) NDCG@*n*

5.4 Experiments with Web Search Data

To verify the effectiveness of our algorithm, we also conducted experiments with a dataset from a commercial web search engine.

5.4.1 Data

This dataset contains over 2000 queries, with human-labeled judgments. The queries were randomly sampled from the query log of the search engine. Human labelers were asked to assign ratings (from 1 which means 'irrelevant' to 5 which means 'definitely relevant') to those top-ranked pages for each query. Each top-ranked page was rated by five

labelers, and the final rating was obtained by majority voting. If the voting failed, a meta labeler was asked to give a final judgment. Not all the documents were labeled due to resource limitation.

We randomly divided the dataset into a subset for training and a subset for testing. There were more than 1,300 queries in the training set, and about 1000 queries in the test set. Figure 5 shows the distribution of document pairs in the training set: about one third queries have less than 200 pairs; more than half of the queries have less than 400 pairs; and the remaining queries have from 400 to over 1400 pairs. If we use document pair level loss function, the majority of the queries with less pairs will be overwhelmed by the minority of the queries with more pairs.

The features are also from the search engine, which mainly consists of two types: query-dependent features and query-independent features. Query-dependent features include term frequencies in anchor text, URL, title, and body text, while query-independent features include page quality, number of hyperlinks and so on. In total, there are 334 features.

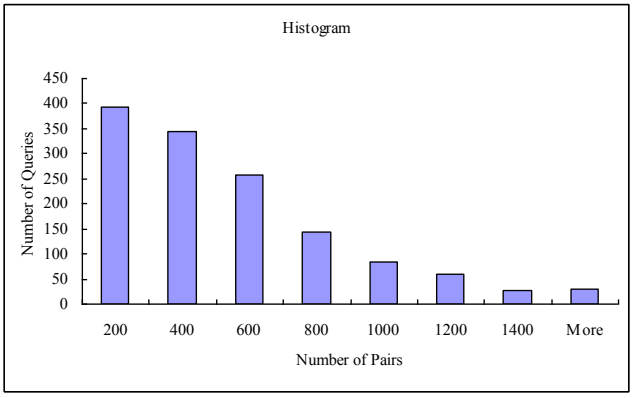


Fig. 5 Distribution of document pairs per query

Since there are five levels of judgment for this data set, only NDCG is suitable for the evaluation.

5.4.2 Result

The accuracies of the learning algorithms are shown in Fig. 6. From the figure, we can see that RankCosine achieves the best result in terms of all NDCG scores, beating the other algorithms by 2~6 NDCG[5] points (which

[5] Similarly, we define a NDCG point by the NDCG score of 0.01.

corresponds to about 4% to 13% relative improvements). TwoLayer-RankNet achieved the second best result, followed by the group of Ranking SVM, RankBoost and linear-RankNet. The results indicate that RankCosine (and using query-level loss function) is the approach one should take for search.

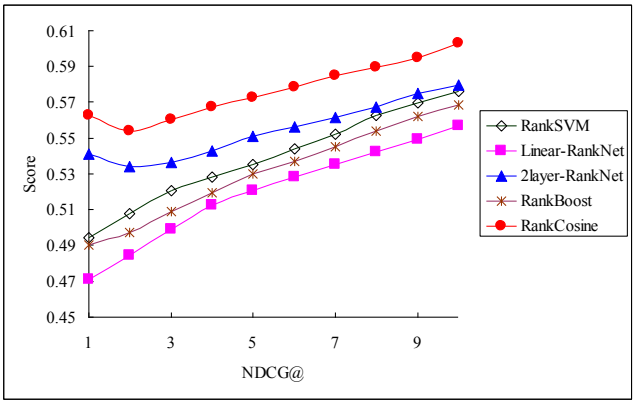


Fig. 6 Performance comparison on testing set

We conducted t-tests on the results of RankCosine and TwoLayer-RankNet. The p-values from NDCG@1 to NDCG@10 with respect to the confidence level of 98% are shown in Table 5. As can be seen, all the p-values are small, indicating that the improvements of RankCosine over TwoLayer-RankNet are statistically significant.

Table 5 P-value of t-tests

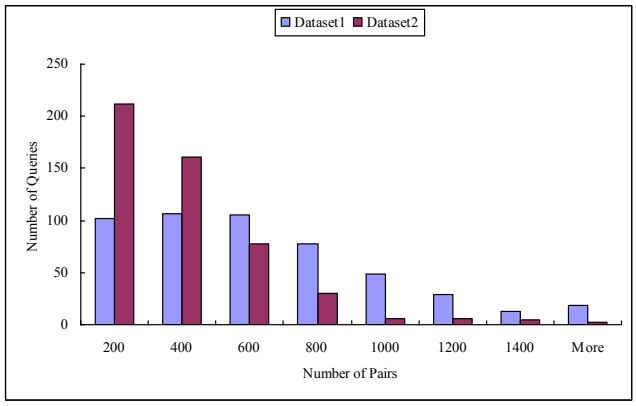
NDCG@*n*	**p-value**
1	7.51E-03
2	6.16E-03
3	2.02E-04
4	6.58E-06
5	4.36E-06
6	2.04E-06
7	2.43E-06
8	1.14E-06
9	4.19E-06
10	8.27E-08

5.4.3 Robustness to query variance

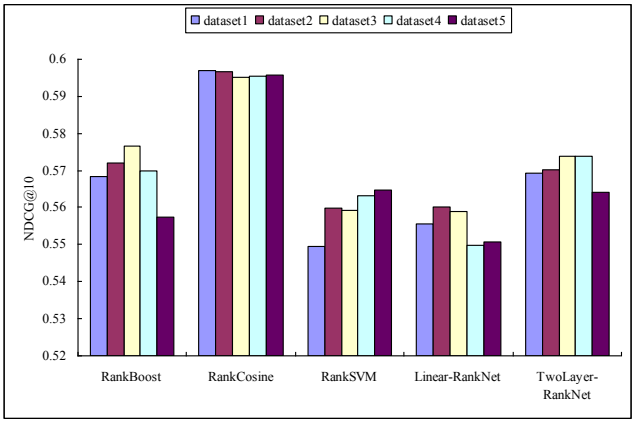
As indicated above, the number of document pairs can vary largely according to queries. We investigated the impact of this variance on the performances of ranking algorithms. For this purpose, we randomly sampled five datasets each with 500 queries from the original training set, trained five ranking models, and then tested the ranking performances

on the test in the same way as before. Since the five training sets were generated by random sampling, we can observe variances in number of documents across queries. Taking dataset 1 and 2 as example (see Fig. 7(a)), we can see that distributions of pair numbers in the two datasets are very different. Therefore, the new training sets can be used to test whether a ranking algorithm is robust to the variances of queries.

The performances of each algorithm with respect to the five training sets are shown in Fig. 7(b). As can be seen from the figure, the results of RankBoost on the five datasets have a large variance. It obtains the highest accuracy of 0.576 on the third dataset and the lowest accuracy of 0.557 on the fifth. The results of Ranking SVM also change dramatically over different training sets. The results indicate that both RankBoost and Ranking SVM are not very robust to query variances. The results of Two-layer RankNet are more stable. In contrast, RankCosine achieves the highest ranking accuracy of 0.597 and its performance varies only a little over different training sets. This shows that query-level loss functions is more robust to query variances than document-pair level loss functions.



(a)



(b)

Fig. 7 (a) Query variances in different training sets, (b) Comparison of NDCG @ 10 with respect to different training sets

6. Query Normalization

As shown in the experimental section, the query-level loss function performs better than the document-pair level loss functions. One may argue that employing a document-pair level loss function while conducting normalization on queries is another possible approach to deal with the query variance problem. In this section, we discuss this problem in details. Specifically, we introduce query normalization to loss functions of Ranking SVM, RankBoost, and RankNet, and look at the corresponding ranking performances.

For Ranking SVM, we can modify its loss function (Eq. (1)) as below,

$$RankSVM: V(\omega, \varepsilon) = \frac{1}{2}\omega^T\omega + C\sum_q \frac{1}{|\#q|}\sum_{i,j}\varepsilon_{i,j,q} \tag{16}$$

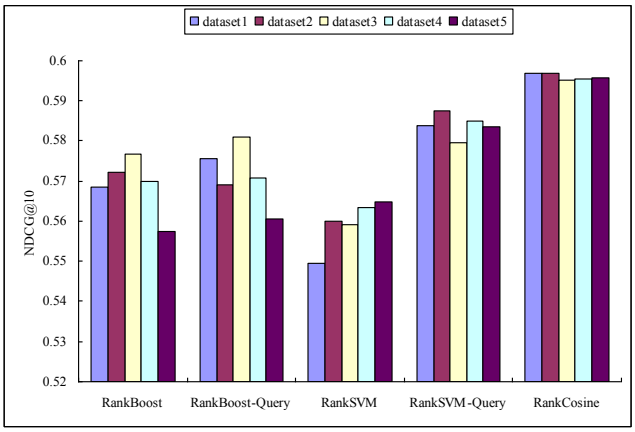
where $\#q$ denotes number of document pairs for query q. Similarly, we can modify the loss function of RankBoost as follows

$$RankBoost: L = \sum_q \frac{1}{|\#q|}\sum_{d_i \succ_q d_j} L(d_i \succ_q d_j) \tag{17}$$

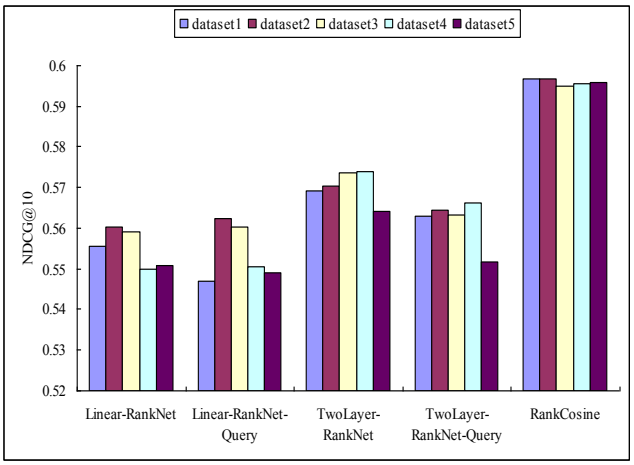
and modify the loss function of RankNet (Eq. (5)) as follows:

$$RankNet: L = \sum_q \frac{1}{|\#q|}\sum_{i,j} L_{q,i,j} \tag{18}$$

With the above modifications, we can still find suitable optimization procedures for performing the learning tasks. We have conducted experiments on the methods and Fig. 8 shows the results using the same setting as above.



(a)



(b)

Fig. 8 Comparison of ranking algorithms with query and pair level loss functions

From Fig. 8, we find that

(1) When compared with their original algorithms, normalized RankBoost and normalized Linear-RankNet obtain higher performance on some datasets while lower performance on the other datasets. Therefore, it is difficult to judge whether the two normalized algorithms are better or worse. However, we can at least say that normalized RankBoost and normalized Linear-RankNet are sensitive to query variance.

(2) The results of normalized TwoLayer-RankNet are worse than those of the original TwoLayer-RankNet for all the five datasets, indicating that the normalized version of TwoLayer-RankNet is not successful.

(3) Query-level Ranking SVM achieves better results on all the five datasets than its original version. This is consistent with the results obtained in (Cao et al. 2006), in which they show that modifying Ranking SVM with query normalization can improve ranking performances. However, in our experiment, the performance of normalized Ranking SVM is still not as good as that of RankCosine.

From the above experiments, we can come to the conclusion that it is non-trivial how to improve the existing ranking algorithms by query normalization. Note that both RankCosine and RankBoost employ Boosting techniques in learning, therefore the difference between the two should be mainly from the loss functions.

7. CONCLUSIONS AND FUTURE WORK

Applying machine learning techniques to ranking in information retrieval has become an important research problem. In this paper, we have investigated how to improve ranking accuracies of machine learning methods by employing suitable loss functions. Our contributions include:

1) We have pointed out that query-level loss functions are more suitable for information retrieval, when compared with document (pair) level loss functions, and have discussed the necessary properties of a good query-level loss function;

2) We have defined the cosine loss function as an example of query-level loss functions, and derived the RankCosine algorithm to minimize the loss function in creation of a generalized additive model;

3) Through empirical study, we have showed that it is difficult to extend the loss functions of the existing methods (e.g. Ranking SVM, RankBoost, RankNet) to the query level.

Experimental results on the datasets of TREC web track, OSHUMED, and a commercial web search engine all show that our proposed query-level loss function can significantly improve the search accuracy.

Future work includes investigation on the relationship between query level loss functions and information retrieval evaluation criteria, further analysis on the properties of a good query level loss function, and study on the generalization ability of the RankCosine algorithm.

ACKNOWLEDGEMENT

We would like to thank Xiu-Bo Geng for her discussion and comments for this work.

REFERENCES

Baeza-Yates, R., Ribeiro-Neto, B. (1999). Modern Information Retrieval, Addison Wesley..

Borlund, P. (2003). The Concept of Relevance in IR, Journal of the American Society for Information Science and Technology 54(10): 913-925

Burges, C.J.C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G. (2005). Learning to Rank using Gradient Descent, 22nd International Conference on Machine Learning, Bonn.

Cao, Y., Xu, J., Liu, T.Y., Li, H., Huang, Y., Hon, H.W. (2006). Adapting ranking SVM to document retrieval. In SIGIR 2006: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, Washington, USA.

Cao, Z., Qin, T., Liu, T. –Y., Tsai, M.-F., & Li, H. (2007). Learning to Rank: From Pairwise Approach to Listwise Approach, In Proc. Of the 17th Annual International Conference on Machine Learning (ICML2007), Oregon, US, 2007.

Crammer, K., & Singer, Y. (2002). Pranking with ranking. NIPS 14.

Craswell, N., Hawking, D., Wilkinson, R., and Wu, M. (2003). Overview of the TREC 2003 Web Track. In NIST Special Publication 500-255: The Twelfth Text REtrieval Conference (TREC 2003), pages 78–92, 2003.

Dekel, O., Manning, C., & Singer, Y. (2004). Loglinear models for label-ranking. NIPS 16.

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research, 2003 (4).

Friedman, J., Hastie, T., & Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. Dept. of Statistics, Stanford University Technical Report.

Hersh, W. R., Buckley, C., Leone, T. J., and Hickam, D. H. (1994). OHSUMED: An interactive retrieval evaluation and new large test collection for research. Proceedings of the 17th Annual ACM SIGIR Conference, pages 192-201.

Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. Advances in Large Margin Classiers, MIT Press, Pages: 115-132.

Jarvelin, K., & Kekalainen, J. (2000). IR evaluation methods for retrieving highly relevant documents. Proc. 23rd ACM SIGIR.

Jarvelin, K., & Kekalainen, J. (2002). Cumulated Gain-Based Evaluation of IR Techniques, ACM Transactions on Information Systems.

Joachims, T. (1999). Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press.

Joachims, T. (2002). Optimizing Search Engines Using Clickthrough Data, Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM.

Matveeva, I., Burges, C., Burkard, T., Laucius, A., & Wong, L. (2006). High accuracy retrieval with multiple nested ranker. Proceeings of SIGIR 2006 (pp. 437–444).

Nallapati, R. (2004). Discriminative models for information retrieval. In SIGIR 2004, Pages: 64-71.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The PageRank Citation Ranking: Bringing Order to the Web, Technical report, Stanford University, Stanford, CA.

Qin, T., Liu, T.Y., Zhang, X.D., Chen, Z., and Ma, W.Y. (2005). A study of relevance propagation for web search. In SIGIR 2005: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pages 408--415, New York, NY, USA. ACM Press.

Qin, T., Liu, T.-Y., Lai, W., Zhang, X.-D., Wang, D.-S., & Li, H. (2007). Ranking with multiple hyperplanes. Proceeings of SIGIR 2007: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval.

Robertson, S. E. (1997). Overview of the okapi projects, Journal of Documentation, Vol. 53, No. 1, pp. 3-7.

Robertson, S. and Hull, D. A. (2000). The TREC-9 Filtering Track Final Report. Proceedings of the 9th Text Retrieval Conference, pages 25-40.

Song, R., Wen, J. R., Shi, S. M., Xin, G. M., Liu, T. Y., Qin, T., Zheng, X., Zhang, J. Y., Xue, G. R., and Ma, W. Y. (2004). Microsoft Research Asia at Web Track and Terabyte Track of TREC 2004, in the 13th TREC

Sormunen, E. (2002). Liberal relevance criteria of TREC—Counting on negligible documents? In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.

Tsai, M.-F., Liu, T.-Y., Qin, T., Chen, H.-H., & Ma, W.-Y. (2007). Frank: A ranking method with fidelity loss. Proceeings of SIGIR 2007: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval.

Vapnik, V. (1998). Statistical learning theory. Wiley.

Voorhees, E.M. and Harman, D.K. (2005). TREC: Experiment and Evaluation in Information Retrieval. MIT Press.

Xue, G.R., Yang, Q., Zeng, H.J., Yu, Y., and Chen, Z. (2005). Exploiting the hierarchical structure for link analysis. In Proc. of the 28th annual international ACM SIGIR conference on Research and development in information retrieval Salvador, Brazil.