Learning to rank with document ranks and scores

Yan Pan a,*, Hai-Xia Luo b, Yong Tang c, Chang-Qin Huang d

a School of Software, Sun Yat-sen University, Guangzhou 510006, China
b Department of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
c Department of Computer Science, South China Normal University, Guangzhou 510631, China
d Engineering Research Center of Computer Network and Information Systems, South China Normal University, Guangzhou 510631, China

A R T I C L E   I N F O

Article history:
Received 12 March 2010
Received in revised form 30 November 2010
Accepted 15 December 2010
Available online 19 December 2010

Keywords:
Learning to rank
Boosting algorithm
Loss function
Machine learning
Information retrieval

A B S T R A C T

The problem of “Learning to rank” is a popular research topic in Information Retrieval (IR) and machine learning communities. Some existing list-wise methods, such as AdaRank, directly use the IR measures as performance functions to quantify how well a ranking function can predict rankings. However, the IR measures only count for the document ranks, but do not consider how well the algorithm predicts the relevance scores of documents. These methods do not make best use of the available prior knowledge and may lead to suboptimal performance. Hence, we conduct research by combining both the document ranks and relevance scores. We propose a novel performance function that encodes the relevance scores. We also define performance functions by combining our proposed one with MAP or NDCG, respectively. The experimental results on the benchmark data collections show that our methods can significantly outperform the state-of-the-art AdaRank baselines.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Learning to rank, a task that seeks to learn ranking functions from function space and sort a set of entities/documents by applying machine learning techniques, has been drawing increasing interest in Information Retrieval (IR) and machine learning research. Burges et al. [3], Cao et al. [5], Freund et al. [9], Joachims [13], Li et al. [14], Yue et al. [25], Taylor et al. [20], Okabe et al. [17], Angizl et al. [11], ElAlami [7], Subramanyam Rallabandi and Sett [19]. Precisely, considering document retrieval as an example, one is given a labeled training set \( T = \{ (q, D_1, Y_1), \ldots, (q_n, D_n, Y_n) \} \) in which \( q_i \) represents a query, \( D_i \) represents the list of corresponding retrieved documents for \( q_i \), and \( Y_i \) is the list of corresponding relevance judgments annotated by human. The task of learning to rank is to construct a ranking function \( f \) from the training data, then sort the examples in the test set based on the learned function \( f \).

Several methods have been developed for the task of learning to rank. These methods seek to train ranking functions by combining many kinds of low level and high level document features (i.e. TF-IDF, BM25). Roughly speaking, most of these methods tend to take one of the two approaches: the pair-wise approach and the list-wise approach.

In the pair-wise approach, the task of learning to rank can be viewed as a classification problem to appropriately classify the preference relationships of document pairs. Ranking SVM [Joachims [13], Herbrich et al. [10], RankBoost Freund et al. [9] and RankNet Burges et al. [3]] are three well-known pair-wise algorithms. However, there are some problems in the pair-wise approach: (1) the document pairs in a query are viewed as equivalently important. However, since the users usually pay more attention to the top-ranked documents than those among lower ranked ones (i.e. in web search), we should also pay more attention to the document pairs among higher ranked documents than those among lower ranked ones. (2) The number of corresponding retrieved documents may be quite different from query to query. The pair-wise approach may be biased towards the queries with more relevant documents Cao et al. [5].

While in the list-wise approach, the list of retrieved documents for a query is viewed as an example in learning. Recent work Cao et al. [5], Xia et al. [22], Xu and Li [23] shows that the list-wise approach usually performs better than the pair-wise one. And list-wise methods can be mainly categorized into two ways. The first one is directly optimizing IR performance measures, such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG). The second one is defining list-wise loss/performance functions.

Some existing state-of-the-art list-wise methods, such as AdaRank Xu and Li [23], directly use the performance measures (i.e. MAP, NDCG) of IR as optimization objectives or performance functions. Since the IR measures only count for the document labels and their ranks (positions in the sorted list), this approach can be viewed as a way that only enforces the learned ranking function \( f \) to predict good rankings. However, how well the learned function \( f \) can predict the relevance scores of documents is also important.
for achieving low generalization error. Take Ranking SVM Joachims [13], Herbrich et al. [10] as an example, for any document pair \((x_i,x_j)\) within a query (assume \(x_i\) ranks before \(x_j\) without loss of generality), the SVM learner enforces the learned ranking function \(f\) to achieve large margin, i.e., maximize the minimum of \(f(x_i) - f(x_j)\). In other words, the SVM learner wants the learned function \(f\) to assign a higher relevance score \(f(x_i)\) to \(x_i\) and a lower one \(f(x_j)\) to \(x_j\). The large margin is an important factor that leads to low generalization error. Hence, in addition to the document ranks, the relevance scores can also be helpful for ranking learning. Ignoring them could lead to suboptimal performance.

Summarizing the previous discussion, the most important intuition of our work is that both the relevance scores and ranks of documents are beneficial to enhance the ranking accuracy. A learning method that involves the relevance scores with the document ranks can expected to obtain low generalization error and make better performance than those existing ones that only use the document ranks for ranking learning. We investigate a list-wise approach by incorporating document ranks and relevance scores. We first define a novel list-wise performance function by encoding relevance scores of documents and thus offset the above mentioned drawbacks of the pair-wise approach. Moreover, we also define performance functions that combine our proposed performance function with MAP or NDCG. Then we derive algorithms on the basis of AdaRank to learn ranking functions for these two methods. Experimental results show that our methods can significantly outperform the state-of-the-art AdaRank-MAP and AdaRank-NDCG Xu and Li [23] baselines.

The rest of this paper is organized as follows. We briefly review the previous research in Section 2. In Section 3, we present our list-wise learning approach. First, we define our novel list-wise performance function in Sub Section 3.1. Then we describe the combination of our performance function with MAP/NDCG in Sub Section 3.2. In Sub Section 3.3, we revisit the AdaRank framework and illustrate our derived learning algorithm. Section 4 will present the experimental results. Finally, Section 5 will conclude the paper with some remarks on future directions.

2. Related work

In recent years many machine learning techniques have been studied for the task of learning to rank Burges et al. [3], Cao et al. [5], Freund et al. [9], Joachims [13], Yue et al. [25], Xu and Li [23], Nallapati [16]. In the methods based on so-called pair-wise approach, the process of learning to rank is viewed as a task to classify the preference order within document pairs. Ranking SVM Joachims [13], Herbrich et al. [10], RankBoost Freund et al. [9] and RankNet Burges et al. [3] are representative pair-wise algorithms. Ranking SVM adopts a large margin optimization approach like the traditional SVM Vapnik et al. [21]. It minimizes the number of incorrectly ordered instance pairs. Several extensions of Ranking SVM have also been proposed to enhance the ranking performance Cao et al. [4], Qin et al. [18]. RankBoost is a boosting algorithm for ranking using pair-wise preference data. RankNet is another well-known algorithm using Neural Network for ranking and cross-entropy as its loss function.

Recently, however, the research on learning to rank have been extended from the pair-wise approach to the list-wise one, in which there are mainly two categories:

The first category is optimizing a loss function directly based on IR performance measures. SVM-MAP Yue et al. [25] adopts structural Support Vector Machine to minimize a loss function that is the upper bound of MAP. AdaRank is a boosting algorithm that optimizes an exponential loss which upper bounds the measures of MAP and NDCG.

The second category is defining list-wise loss/performance functions which take the list of retrieved documents for the same query as an example. ListNet Cao et al. [5] defines a loss function based on KL-divergence between two permutation probability distributions. ListMLE Xia et al. [22] defines another list-wise likelihood loss function based on Luce Model.

In the list-wise case, document ranks and their relevance scores are two kinds of information available for ranking learning. Unfortunately, some existing methods, such as AdaRank Xu and Li [23], only consider the document ranks but totally ignores how well the algorithm can predict the relevance scores of documents. They directly use the performance measures (i.e. MAP, NDCG) of IR to construct loss functions, which are only related to the ranks of documents. These methods do not make best use of available information and may lead to suboptimal performance. Hence, in this paper, we conduct research on ranking learning by incorporating document ranks with relevance scores.

3. Our method

We first introduce the notations used hereafter and formulate the problem of learning to rank. Given a labeled training set \(S = \{(q_i,D_i),(y_i)\}_{i=1,2,...,n}\) and a test set \(T = \{(q_n,D_n),(y_n)\}_{n=1,2,...,m}\), in which \(q_i\) represents a query, \(D_i = (d_1,d_2,...,d_{|q_i|})\) represents the list of corresponding retrieved documents for \(q_i\) with \(n(q_i)\) as the number of the retrieved documents, and \(Y_i = \{y_{1,i},y_{2,i},...,y_{n(i)}\}\) is the list of corresponding relevance judgments annotated by human. \(y_{k,i} \in \{r_1,r_2,...,r_k\}\) (\(j = 1,2,...,n(q_i)\)) is the list of corresponding relevance judgments annotated by human. \(y_{k,i} \in \{r_1,r_2,...,r_k\}\) (\(j = 1,2,...,n(q_i)\)) is the list of corresponding relevance judgments annotated by human. \(y_{k,i} \in \{r_1,r_2,...,r_k\}\) (\(j = 1,2,...,n(q_i)\)) is the list of corresponding relevance judgments annotated by human. \(y_{k,i} \in \{r_1,r_2,...,r_k\}\) (\(j = 1,2,...,n(q_i)\)) is the list of corresponding relevance judgments annotated by human. \(y_{k,i} \in \{r_1,r_2,...,r_k\}\) (\(j = 1,2,...,n(q_i)\)) is the list of corresponding relevance judgments annotated by human.

3.1. A list-wise performance function

Following the Empirical Risk Minimization Principle Vapnik et al. [21], we need to define a loss function to quantify how well the ranking function \(h\) can predict rankings, and to minimize the following empirical risk (training error).

\[
R(h) = \frac{1}{n} \sum_{i=1}^{n} \text{loss}(h(X_i), y_i).
\]  

(3.1)

where \(X_i\) is the list of feature vectors for the \(i\)-th query \(q_i\) in training set, \(h(X_i)\) is the ranking function and \(n\) is the size of training data.

In this paper, we employ a new list-wise performance function to quantify the performance of the ranking function \(h\), in which document lists are viewed as examples. Initially we can define the performance function as follows.

\[
\text{per } f_{\text{init}}(h,q) = \frac{1}{2} \sum_{y \in S^+} \sum_{y' \in S^-} f_q[|h(x_i) - h(x_j)|].
\]  

(3.2)

where \(S^+\) and \(S^-\) denote the set of relevant and irrelevant documents for query \(q\), respectively. \(Z = |S^+| \times |S^-|\) is a normalization factor that is the number of document pairs for query \(q\). As for \(f_q[|h(x_i) - h(x_j)|]\) is an indicator function which is defined to be 1 if \(x_i\) is true, and -1 otherwise. It is easy to verify that Eq. (3.2) can be rewritten as
per \( f_{\text{init}}(h, q) = \frac{1}{Z} \sum_{i \in S} \sum_{j \in S} (h(x_i) - h(x_j)). \) (3.3)

As mentioned in Section 1, one of the drawbacks in pair-wise approach is that the learned model may be biased towards the queries with more relevant documents. In order to avoid this drawback, we take the number of pairs within a query as the normalization factor \( Z \) in our performance function. Moreover, since the users usually pay more attention to the top-ranked documents than the low-ranked ones, we use a parameter \( w (w > 0) \) to assign larger weights to those document pairs whose two documents are both ranked in top 10. Thus the performance function can be defined as

\[
per f(h, q) = \frac{1}{Z} \left[ w \sum_{i \in S} \sum_{j < 10} [h(x_i) - h(x_j)] + \sum_{i \in S} \sum_{j < 10} [h(x_i) - h(x_j)] \right]. \quad (3.4)
\]

Because the weights of the pairs with top 2 ranked documents are \( w + 1 \) and those of the other pairs are 1, the normalization factor \( Z \) should be changed to \( Z = |S|^2 + |S| + w \times P \), where \( P \) is the number of pairs with two top 10 ranked documents. Finally we define the loss function in the empirical risk (Eq. (3.1)) as

\[
\text{loss}(h, q) = 1 - per f(h, q). \quad (3.5)
\]

Hence, we can minimize the loss function by maximizing the performance function \( per f(h, q) \). In this paper, we use a derived algorithm based on AdaRank to optimize the performance function.

3.2. Combining the performance function with MAP/NDCG

It is a natural idea to directly optimize performance measures for IR, such as MAP and NDCG. AdaRank uses MAP or NDCG as the performance function for the selection of weak ranker (i.e. a document feature) in each iteration. However, the calculation of these performance measures are only based on the document ranks (the order of documents), while the relevance scores (here they are the feature values of the selected weak ranker in each iteration) of documents are totally ignored. We argue that both the relevance scores and ranks of documents are useful to enhance the ranking accuracy and hence we should use both of them to define performance functions in learning. In our proposed performance function in Eq. (3.4), the document scores are used in computing the weights of document pairs. We can easily modify the AdaRank algorithm to use our proposed performance function in selecting weak rankers. Moreover, we can also incorporate our performance function with MAP/NDCG as the new one in weak ranker selection. And the new performance functions are

\[
per f_{\text{map}}(h, q) = \frac{1}{1 + \beta} (\beta \times per f(h, q) + \text{MAP}). \quad (3.6)
\]

and

\[
per f_{\text{nndcg}}(h, q) = \frac{1}{1 + \beta} (\beta \times per f(h, q) + \text{NDCG}). \quad (3.7)
\]

The parameter \( \beta \) is a trade-off factor to control the importance of the two parts. In our experiments, we select the parameter \( \beta \) by cross-validation. Experimental results show that these two methods (respectively, using our proposed performance function or the combined one) can significantly outperform the AdaRank baselines using MAP/NDCG as performance function.

3.3. Our derived AdaRank algorithm

AdaRank Xu and Li [23] is a boosting algorithm that extends AdaBoost Freund and Schapire [8] philosophy for binary classification. It takes document lists (queries) as examples and optimizes an exponential loss which upper bounds the measures of MAP and NDCG. AdaRank can be viewed as a process to find convex combination of weak rankers to minimize a specific cost functional (i.e. the loss function). AdaRank maintains a weight distribution for all queries, and iteratively selects a weak ranker (i.e. a document feature) that maximizes the performance measure (i.e. MAP or NDCG). If a query is not ranked well by the model created so far, it increases the weight of these “hard” queries, so that the ranking of the weak ranker in the next round will be more focused on them. AdaRank chooses a weight \( \alpha \) which measures the importance of the weak ranker, and the final ranking function is a weighted combination of \( T \) weak ranker as follows:

\[
f(h) = \sum_{t=1}^{T} \alpha_t h_t(x). \quad (3.8)
\]

where \( T \) is the total number of iterations, \( \alpha_t \) is the weight of the weak ranker in the \( t \)-th round.

In this paper, we derive our learning algorithm based on AdaRank and use AdaRank as baselines in our experiments. Thereasons are threefold: (1) AdaRank is one of the state-of-the-art list-wise algorithms for learning to rank. (2) AdaRank provides a flexible framework and in principle arbitrary performance functions based on document list can be used in selecting weak rankers. (3) The algorithm of AdaRank is simple and easy to implement. Our derived algorithm is shown in Algorithm 1. The algorithm framework is similar to the AdaRank algorithm in Section 3.2 of Xu and Li [23]. The difference is that we, respectively, use the performance functions in Eqs. (3.4), (3.6) or Eq. (3.7) as the performance function (the function perf in Algorithm 1).

Compared to the original AdaRank algorithm, our approach uses particular performance functions. This difference only leads to different criteria for selecting weak learners and updating weights. Hence, our approach does not increase the computational cost or make the algorithm more complicated.

**Algorithm 1.** Our derived AdaRank algorithm.

| Input: training data \( S = \{(q_i, D_i, Y_i)\}_{i=1,2,...,n} \) Parameters: number of iterations \( T \) and combined parameter \( \beta, w \). Initialize: \( D_1(i) = \frac{1}{n} \) For \( t = 1 \) to \( T \) do Create weak ranker \( h_t \) with weighted distribution \( D_t \) on training data Choose \( \alpha_t \)
| \( \alpha_t = \frac{1}{Z} \ln \frac{\sum_{i=1}^{n} D_t(i) \{1 + per f(h_t, q_i)\}}{\sum_{i=1}^{n} D_t(i) \{1 - per f(h_t, q_i)\}} \) Get \( f_t(x) \)
| \( f_t(x) = \sum_{i=1}^{T} \alpha_t h_t(x) \) Update \( D_{t+1} \)
| \( D_{t+1} = \frac{\exp(-per f(f_t, q_i))}{\sum_{i=1}^{T} \exp(-per f(f_t, q_i))} \) End for Output: ranking function \( f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \).
4. Experiments

In this section, we evaluate the ranking performance of our proposed methods. The experimental results show that our methods outperform the AdaRank-MAP and AdaRank-NDCG baselines.

4.1. Datasets

We conduct our experiments on the LETOR 2.0 and LETOR 4.0 data collections Liu et al. [15], which is public available benchmark data collections for the research on learning to rank. LETOR 2.0 contains three datasets for document retrieval: TREC 2003 (TD2003), TREC 2004 (TD2004) and OHSUMED. LETOR 4.0 contains two large-scale datasets MQ2007 and MQ2008.

The TREC datasets Craswell et al. [6] contain many features extracted from query-document pairs in topic distillation task of TREC 2003 and TREC 2004. The documents are from the.GOV collection which is based on a January 2002 crawl of.gov web sites Liu et al. [15]. There are totally 44 features in TREC datasets, covering a wide range including low level features (i.e. TF, IDF) and high level features (i.e. BM25, Language models). In the TREC datasets, the relevance judgment for each query-document pair is binary, 1 for relevant and 0 for irrelevant. There are 50 queries in TREC 2003 and 75 queries in TREC 2004, respectively.

The OHSUMED dataset Hersh et al. [11] consists of 106 queries with 25 features extracted from the on-line medical information database MEDLINE, which is widely used in information retrieval research. The relevance judgment for each query-document pair is binary, 1 for definitely relevant, 1 for partially relevant and 0 for irrelevant.

The MQ2007 and MQ2008 datasets are two large-scale datasets. There are about 1700 queries in MQ2007 with labeled documents and about 800 queries in MQ2008 with labeled documents. Each of the five datasets is divided into fivefolds. There are a training set, a validation set and a test set, respectively, in each fold, which can be used to conduct cross-validation.

4.2. Evaluation measures

As to evaluate the performance of ranking models, we use Mean Average Precision (MAP) Baeza-Yates and Ribeiro-Neto [2] and Normalized Discounted Cumulative Gain (NDCG) Jarvelin and Kekalainen [12] as evaluation measures. MAP is a standard evaluation measure widely used in Information Retrieval systems. It works for the cases with binary relevance judgments: relevant and irrelevant. MAP is the mean of average precisions over a set of queries. Precision at position $j$ represents the proportion of relevant documents within the top $j$ retrieved documents, which can be calculated by

$$P(j) = \frac{N_{pos}(j)}{j},$$

where $N_{pos}(j)$ denotes the number of relevant documents within the top $j$ documents. Given a query $q$, the average precision of $q$ is defined as the average of all $P(j)$ ($j = 1, 2, \ldots, n$) and can be calculated by the following equation:

$$AvgP_i = \frac{\sum_{j=1}^{M} P(j) \times pos(j)}{N_{pos}},$$

where $j$ is the position, $M$ is the number of retrieved documents. And $pos(j)$ is an indicator function. If the document at position $j$ is relevant, then $pos(j)$ is 1, or else $pos(j)$ is 0. $N_{pos}$ represents the total number of relevant documents for query $q$. $P(j)$ is the precision at the given position $j$.

NDCG is another popular evaluation criterion for comparing ranking performance in Information Retrieval. Unlike MAP, NDCG can deal with the cases which have more than two levels of relevance judgments. Given a query $q$, the NDCG score at position $m$ in the ranking list of documents can be calculated by the equation as follows:

$$N_i = \frac{1}{Z_i} \sum_{j=1}^{m} \frac{2^r(j) - 1}{\log(1 + j)}.$$  \hspace{1cm} (4.3)

where $r(j)$ is the grade of the $j$th document and $Z_i$ is a constant used for normalization, which is chosen so that the NDCG score for a perfect ranking is 1.

4.3. Experiment procedure

To evaluate the efficiency of our proposed methods, we compare the ranking accuracy of our methods with other state-of-the-art learning algorithms for ranking. Our approach uses derived AdaRank algorithm as the optimization framework. Hence, as to make fair comparison, we choose AdaRank-MAP and AdaRank-NDCG Xu and Li [23] as baselines.

The number of iterations $T$, the parameter $\beta$ and the parameter $w$ in our methods are chosen by cross-validation. Since the top 10 documents ranked by a ranking model are viewed as the most important ones in web search, we choose the parameters, which achieve the best value of NDCG@10 on the validation set, as test parameters on the test data.

4.4. Experimental results

Figs. 1–6 show the performance of five ranking algorithms on the TD2003, TD2004, OHSUMED, MQ2007 and MQ2008 datasets.

Fig. 1. Comparison of NDCG among AdaRank-NDCG, AdaRank-MAP, PERF, PERF-NDCG and PERF-MAP on TREC 2003 dataset.

Fig. 2. Comparison of NDCG among AdaRank-NDCG, AdaRank-MAP, PERF, PERF-NDCG and PERF-MAP on TREC 2004 dataset.
with respect to NDCG and MAP. Here, AdaRank-MAP and AdaRank-NDCG stand for the conventional AdaRank methods which optimize MAP and NDCG, respectively. PERF is the learning algorithm that uses our proposed performance function in Eq. (3.4). PERF-MAP and PERF-NDCG, respectively, denote the algorithm using combination of our proposed performance function with MAP (Eq. (3.6)) or NDCG (Eq. (3.7)).

Two observations can be made from the experimental results: (1) the algorithm PERF, based on our proposed performance function that encodes relevance scores of documents, is almost always the best. It makes significant improvement over the AdaRank baselines on all five datasets. It shows that the relevance scores encoded in our proposed performance function can be helpful to improve ranking accuracy, and heavily weighted the top document pairs is also beneficial. (2) The combined method PERF-MAP/PERF-NDCG, which incorporates our proposed performance function with MAP/NDCG, also outperforms the corresponding AdaRank-MAP/AdaRank-NDCG baseline. It is empirically verified that the combination of document ranks and relevance scores can be able to enhance the ranking performance.

What’s more, we demonstrate our improvement over the baseline from the statistic perspective by significance test, in which we use $P$-value to measure the difference of two systems’ performance. $P$-value Yang and Liu [24] is an indicator to measure the difference of two approaches by comparing the significance level by probabilistic methods of $S$-test or $t$-test Yang and Liu [24]. The smaller $P$-value is, the more significant improvement is. Here we use $t$-test Yang and Liu [24]. The experiments’ significance tests show that our proposed methods over the AdaRank baselines on all the five datasets with respect to NDCG are all statistically significant (the $P$-values are smaller than 0.05).

5. Conclusions

In this paper, we present a list-wise approach that incorporates document ranks and relevance scores to learn ranking functions. The main intuition of this work is that both the ranks and relevance scores of documents have potential to enhance ranking performance. The key challenge is how to effectively encode the relevance scores into the objective functions and how to combine the information of relevance scores with ranking measures. Our proposed approach mainly makes three contributions: (1) we propose a performance function to encode the relevance score for ranking; (2) we combine the proposed performance function with popular ranking measures to form new performance functions; (3) we derive an algorithm from AdaRank optimization framework to learn the
References


