AdaRank: A Boosting Algorithm for Information Retrieval

Jun Xu
Microsoft Research Asia
No. 49 Zhichun Road, Haidian Distinct
Beijing, China 100080
junxu@microsoft.com

Hang Li
Microsoft Research Asia
No. 49 Zhichun Road, Haidian Distinct
Beijing, China 100080
hangli@microsoft.com

ABSTRACT

In this paper we address the issue of learning to rank for document retrieval. In the task, a model is automatically created with some training data and then is utilized for ranking of documents. The goodness of a model is usually evaluated with performance measures such as MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain). Ideally a learning algorithm would train a ranking model that could directly optimize the performance measures with respect to the training data. Existing methods, however, are only able to train ranking models by minimizing loss functions loosely related to the performance measures. For example, Ranking SVM and RankBoost train ranking models by minimizing classification errors on instance pairs. To deal with the problem, we propose a novel learning algorithm within the framework of boosting, which can minimize a loss function *directly defined on the performance measures*. Our algorithm, referred to as AdaRank, repeatedly constructs 'weak rankers' on the basis of re-weighted training data and finally linearly combines the weak rankers for making ranking predictions. We prove that the training process of AdaRank is exactly that of enhancing the performance measure used. Experimental results on four benchmark datasets show that AdaRank significantly outperforms the baseline methods of BM25, Ranking SVM, and RankBoost.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models

General Terms

Algorithms, Experimentation, Theory

Keywords

Information retrieval, Learning to rank, Boosting

1. INTRODUCTION

Recently 'learning to rank' has gained increasing attention in both the fields of information retrieval and machine learning. When applied to document retrieval, learning to rank becomes a task as follows. In training, a ranking model is constructed with data consisting of queries, their corresponding retrieved documents, and relevance levels given by humans. In ranking, given a new query, the corresponding retrieved documents are sorted by using the trained ranking model. In document retrieval, usually ranking results are evaluated in terms of performance measures such as MAP (Mean Average Precision) [1] and NDCG (Normalized Discounted Cumulative Gain) [15]. Ideally, the ranking function is created so that the accuracy of ranking in terms of one of the measures with respect to the training data is maximized.

Several methods for learning to rank have been developed and applied to document retrieval. For example, Herbrich et al. [13] propose a learning algorithm for ranking on the basis of Support Vector Machines, called Ranking SVM. Freund et al. [8] take a similar approach and perform the learning by using boosting, referred to as RankBoost. All the existing methods used for document retrieval [2, 3, 8, 13, 16, 20] are designed to optimize loss functions loosely related to the IR performance measures, not loss functions directly based on the measures. For example, Ranking SVM and RankBoost train ranking models by minimizing classification errors on instance pairs.

In this paper, we aim to develop a new learning algorithm that can directly optimize any performance measure used in document retrieval. Inspired by the work of AdaBoost for classification [9], we propose to develop a boosting algorithm for information retrieval, referred to as AdaRank. AdaRank utilizes a linear combination of 'weak rankers' as its model. In learning, it repeats the process of re-weighting the training sample, creating a weak ranker, and calculating a weight for the ranker.

We show that AdaRank algorithm can iteratively optimize an exponential loss function based on any of IR performance measures. A lower bound of the performance on training data is given, which indicates that the ranking accuracy in terms of the performance measure can be continuously improved during the training process.

AdaRank offers several advantages: ease in implementation, theoretical soundness, efficiency in training, and high accuracy in ranking. Experimental results indicate that AdaRank can outperform the baseline methods of BM25, Ranking SVM, and RankBoost, on four benchmark datasets including OHSUMED, WSJ, AP, and .Gov.

Tuning ranking models using certain training data and a performance measure is a common practice in IR [1]. As the number of features in the ranking model gets larger and the amount of training data gets larger, the tuning becomes harder. From the viewpoint of IR, AdaRank can be viewed as a machine learning method for ranking model tuning.

Recently, direct optimization of performance measures in learning has become a hot research topic. Several methods for classifi-

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.

cation [17] and ranking [5, 19] have been proposed. AdaRank can be viewed as a machine learning method for direct optimization of performance measures, based on a different approach.

The rest of the paper is organized as follows. After a summary of related work in Section 2, we describe the proposed AdaRank algorithm in details in Section 3. Experimental results and discussions are given in Section 4. Section 5 concludes this paper and gives future work.

2. RELATED WORK

2.1 Information Retrieval

The key problem for document retrieval is ranking, specifically, how to create the ranking model (function) that can sort documents based on their relevance to the given query. It is a common practice in IR to tune the parameters of a ranking model using some labeled data and one performance measure [1]. For example, the state-of-the-art methods of BM25 [24] and LMIR (Language Models for Information Retrieval) [18, 22] all have parameters to tune. As the ranking models become more sophisticated (more features are used) and more labeled data become available, how to tune or train ranking models turns out to be a challenging issue.

Recently methods of 'learning to rank' have been applied to ranking model construction and some promising results have been obtained. For example, Joachims [16] applies Ranking SVM to document retrieval. He utilizes click-through data to deduce training data for the model creation. Cao et al. [4] adapt Ranking SVM to document retrieval by modifying the Hinge Loss function to better meet the requirements of IR. Specifically, they introduce a Hinge Loss function that heavily penalizes errors on the tops of ranking lists and errors from queries with fewer retrieved documents. Burges et al. [3] employ Relative Entropy as a loss function and Gradient Descent as an algorithm to train a Neural Network model for ranking in document retrieval. The method is referred to as 'RankNet'.

2.2 Machine Learning

There are three topics in machine learning which are related to our current work. They are 'learning to rank', boosting, and direct optimization of performance measures.

Learning to rank is to automatically create a ranking function that assigns scores to instances and then rank the instances by using the scores. Several approaches have been proposed to tackle the problem. One major approach to learning to rank is that of transforming it into binary classification on instance pairs. This 'pair-wise' approach fits well with information retrieval and thus is widely used in IR. Typical methods of the approach include Ranking SVM [13], RankBoost [8], and RankNet [3]. For other approaches to learning to rank, refer to [2, 11, 31].

In the pair-wise approach to ranking, the learning task is formalized as a problem of classifying instance pairs into two categories (correctly ranked and incorrectly ranked). Actually, it is known that reducing classification errors on instance pairs is equivalent to maximizing a lower bound of MAP [16]. In that sense, the existing methods of Ranking SVM, RankBoost, and RankNet are only able to minimize loss functions that are loosely related to the IR performance measures.

Boosting is a general technique for improving the accuracies of machine learning algorithms. The basic idea of boosting is to repeatedly construct 'weak learners' by re-weighting training data and form an ensemble of weak learners such that the total performance of the ensemble is 'boosted'. Freund and Schapire have proposed the first well-known boosting algorithm called *AdaBoost* (**Ada**ptive **Boost**ing) [9], which is designed for binary classification (0-1 prediction). Later, Schapire & Singer have introduced a generalized version of AdaBoost in which weak learners can give confidence scores in their predictions rather than make 0-1 decisions [26]. Extensions have been made to deal with the problems of multi-class classification [10, 26], regression [7], and ranking [8]. In fact, AdaBoost is an algorithm that ingeniously constructs a linear model by minimizing the 'exponential loss function' with respect to the training data [26]. Our work in this paper can be viewed as a boosting method developed for ranking, particularly for ranking in IR.

Recently, a number of authors have proposed conducting direct optimization of multivariate performance measures in learning. For instance, Joachims [17] presents an SVM method to directly optimize nonlinear multivariate performance measures like the F_1 measure for classification. Cossock & Zhang [5] find a way to approximately optimize the ranking performance measure DCG [15]. Metzler et al. [19] also propose a method of directly maximizing rank-based metrics for ranking on the basis of manifold learning. AdaRank is also one that tries to directly optimize multivariate performance measures, but is based on a different approach. AdaRank is unique in that it employs an exponential loss function based on IR performance measures and a boosting technique.

3. OUR METHOD: ADARANK

3.1 General Framework

We first describe the general framework of learning to rank for document retrieval. In retrieval (testing), given a query the system returns a ranking list of documents in descending order of the relevance scores. The relevance scores are calculated with a ranking function (model). In learning (training), a number of queries and their corresponding retrieved documents are given. Furthermore, the relevance levels of the documents with respect to the queries are also provided. The relevance levels are represented as ranks (i.e., categories in a total order). The objective of learning is to construct a ranking function which achieves the best results in ranking of the training data in the sense of minimization of a loss function. Ideally the loss function is defined on the basis of the performance measure used in testing.

Suppose that $Y = \{r_1, r_2, \cdots, r_\ell\}$ is a set of ranks, where ℓ denotes the number of ranks. There exists a total order between the ranks $r_\ell \succ r_{\ell-1} \succ \cdots \succ r_1$, where '$\succ$' denotes a preference relationship.

In training, a set of queries $Q = \{q_1, q_2, \cdots, q_m\}$ is given. Each query q_i is associated with a list of retrieved documents $\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,n(q_i)}\}$ and a list of labels $\mathbf{y}_i = \{y_{i1}, y_{i2}, \cdots, y_{i,n(q_i)}\}$, where $n(q_i)$ denotes the sizes of lists $\mathbf{d}_i$ and $\mathbf{y}_i$, d_{ij} denotes the j^{th} document in $\mathbf{d}_i$, and $y_{ij} \in Y$ denotes the rank of document d_{ij}. A feature vector $\vec{x}_{ij} = \Psi(q_i, d_{ij}) \in \mathcal{X}$ is created from each query-document pair $(q_i, d_{ij}), i = 1, 2, \cdots, m; j = 1, 2, \cdots, n(q_i)$. Thus, the training set can be represented as $S = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^m$.

The objective of learning is to create a ranking function $f : \mathcal{X} \mapsto \Re$, such that for each query the elements in its corresponding document list can be assigned relevance scores using the function and then be ranked according to the scores. Specifically, we create a permutation of integers $\pi(q_i, \mathbf{d}_i, f)$ for query q_i, the corresponding list of documents $\mathbf{d}_i$, and the ranking function f. Let $\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,n(q_i)}\}$ be identified by the list of integers $\{1, 2, \cdots, n(q_i)\}$, then permutation $\pi(q_i, \mathbf{d}_i, f)$ is defined as a bijection from $\{1, 2, \cdots, n(q_i)\}$ to itself. We use $\pi(j)$ to denote the position of item j (i.e., d_{ij}). The learning process turns out to be that of minimizing the loss function which represents the disagreement between the permutation $\pi(q_i, \mathbf{d}_i, f)$ and the list of ranks $\mathbf{y}_i$, for all of the queries.

Table 1: Notations and explanations.

Notations	Explanations
$q_i \in Q$	i^{th} query
$\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,n(q_i)}\}$	List of documents for q_i
$y_{ij} \in \{r_1, r_2, \cdots, r_\ell\}$	Rank of d_{ij} w.r.t. q_i
$\mathbf{y}_i = \{y_{i1}, y_{i2}, \cdots, y_{i,n(q_i)}\}$	List of ranks for q_i
$S = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^m$	Training set
$\vec{x}_{ij} = \Psi(q_i, d_{ij}) \in \mathcal{X}$	Feature vector for (q_i, d_{ij})
$f(\vec{x}_{ij}) \in \Re$	Ranking model
$\pi(q_i, \mathbf{d}_i, f)$	Permutation for q_i, $\mathbf{d}_i$, and f
$h_t(\vec{x}_{ij}) \in \Re$	t^{th} weak ranker
$E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i) \in [-1, +1]$	Performance measure function

In the paper, we define the rank model as a linear combination of weak rankers: $f(\vec{x}) = \sum_{t=1}^T \alpha_t h_t(\vec{x})$, where $h_t(\vec{x})$ is a weak ranker, α_t is its weight, and T is the number of weak rankers.

In information retrieval, query-based performance measures are used to evaluate the 'goodness' of a ranking function. By query based measure, we mean a measure defined over a ranking list of documents with respect to a query. These measures include MAP, NDCG, MRR (Mean Reciprocal Rank), WTA (Winners Take ALL), and Precision@n [1, 15]. We utilize a general function $E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i) \in [-1, +1]$ to represent the performance measures. The first argument of E is the permutation π created using the ranking function f on $\mathbf{d}_i$. The second argument is the list of ranks $\mathbf{y}_i$ given by humans. E measures the agreement between π and $\mathbf{y}_i$. Table 1 gives a summary of notations described above.

Next, as examples of performance measures, we present the definitions of MAP and NDCG. Given a query q_i, the corresponding list of ranks $\mathbf{y}_i$, and a permutation π_i on $\mathbf{d}_i$, average precision for q_i is defined as:

$$AvgP_i = \frac{\sum_{j=1}^{n(q_i)} P_i(j) \cdot y_{ij}}{\sum_{j=1}^{n(q_i)} y_{ij}}, \quad (1)$$

where y_{ij} takes on 1 and 0 as values, representing being relevant or irrelevant and $P_i(j)$ is defined as precision at the position of d_{ij}:

$$P_i(j) = \frac{\sum_{k:\pi_i(k) \le \pi_i(j)} y_{ik}}{\pi_i(j)}, \quad (2)$$

where $\pi_i(j)$ denotes the position of d_{ij}.

Given a query q_i, the list of ranks $\mathbf{y}_i$, and a permutation π_i on $\mathbf{d}_i$, NDCG at position m for q_i is defined as:

$$N_i = n_i \cdot \sum_{j:\pi_i(j) \le m} \frac{2^{y_{ij}} - 1}{\log(1 + \pi_i(j))}, \quad (3)$$

where y_{ij} takes on ranks as values and n_i is a normalization constant. n_i is chosen so that a perfect ranking π_i^*'s NDCG score at position m is 1.

3.2 Algorithm

Inspired by the AdaBoost algorithm for classification, we have devised a novel algorithm which can optimize a loss function based on the IR performance measures. The algorithm is referred to as 'AdaRank' and is shown in Figure 1.

AdaRank takes a training set $S = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^m$ as input and takes the performance measure function E and the number of iterations T as parameters. AdaRank runs T rounds and at each round it creates a weak ranker $h_t (t = 1, \cdots, T)$. Finally, it outputs a ranking model f by linearly combining the weak rankers.

At each round, AdaRank maintains a distribution of weights over the queries in the training data. We denote the distribution of weights at round t as $\boldsymbol{P}_t$ and the weight on the i^{th} training query q_i at round t as $P_t(i)$. Initially, AdaRank sets equal weights to the queries. At each round, it increases the weights of those queries that are not ranked well by f_t, the model created so far. As a result, the learning at the next round will be focused on the creation of a weak ranker that can work on the ranking of those 'hard' queries.

Input: $S = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^m$, and parameters E and T
Initialize $P_1(i) = 1/m$.
For $t = 1, \cdots, T$

- Create weak ranker h_t with weighted distribution $\boldsymbol{P}_t$ on training data S.
- Choose α_t

$$\alpha_t = \frac{1}{2} \cdot \ln \frac{\sum_{i=1}^m P_t(i)\{1 + E(\pi(q_i, \mathbf{d}_i, h_t), \mathbf{y}_i)\}}{\sum_{i=1}^m P_t(i)\{1 - E(\pi(q_i, \mathbf{d}_i, h_t), \mathbf{y}_i)\}}.$$

- Create f_t

$$f_t(\vec{x}) = \sum_{k=1}^t \alpha_k h_k(\vec{x}).$$

- Update $\boldsymbol{P}_{t+1}$

$$P_{t+1}(i) = \frac{\exp\{-E(\pi(q_i, \mathbf{d}_i, f_t), \mathbf{y}_i)\}}{\sum_{j=1}^m \exp\{-E(\pi(q_j, \mathbf{d}_j, f_t), \mathbf{y}_j)\}}.$$

End For
Output ranking model: $f(\vec{x}) = f_T(\vec{x})$.

Figure 1: The AdaRank algorithm.

At each round, a weak ranker h_t is constructed based on training data with weight distribution $\boldsymbol{P}_t$. The goodness of a weak ranker is measured by the performance measure E weighted by $\boldsymbol{P}_t$:

$$\sum_{i=1}^m P_t(i) E(\pi(q_i, \mathbf{d}_i, h_t), \mathbf{y}_i).$$

Several methods for weak ranker construction can be considered. For example, a weak ranker can be created by using a subset of queries (together with their document list and label list) sampled according to the distribution $\boldsymbol{P}_t$. In this paper, we use single features as weak rankers, as will be explained in Section 3.6.

Once a weak ranker h_t is built, AdaRank chooses a weight $\alpha_t > 0$ for the weak ranker. Intuitively, α_t measures the importance of h_t.

A ranking model f_t is created at each round by linearly combining the weak rankers constructed so far $h_1, \cdots, h_t$ with weights $\alpha_1, \cdots, \alpha_t$. f_t is then used for updating the distribution $\boldsymbol{P}_{t+1}$.

3.3 Theoretical Analysis

The existing learning algorithms for ranking attempt to minimize a loss function based on instance pairs (document pairs). In contrast, AdaRank tries to optimize a loss function based on queries. Furthermore, the loss function in AdaRank is defined on the basis of general IR performance measures. The measures can be MAP, NDCG, WTA, MRR, or any other measures whose range is within $[-1, +1]$. We next explain why this is the case.

Ideally we want to maximize the ranking accuracy in terms of a performance measure on the training data:

$$\max_{f \in \mathcal{F}} \sum_{i=1}^m E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i), \quad (4)$$

where $\mathcal{F}$ is the set of possible ranking functions. This is equivalent to minimizing the loss on the training data

$$\min_{f\in\mathcal{F}} \sum_{i=1}^{m} (1 - E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i)). \quad (5)$$

It is difficult to directly optimize the loss, because E is a non-continuous function and thus may be difficult to handle. We instead attempt to minimize an upper bound of the loss in (5)

$$\min_{f\in\mathcal{F}} \sum_{i=1}^{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i)\}, \quad (6)$$

because $e^{-x} \geq 1 - x$ holds for any $x \in \mathfrak{R}$. We consider the use of a linear combination of weak rankers as our ranking model:

$$f(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x}). \quad (7)$$

The minimization in (6) then turns out to be

$$\min_{h_t\in\mathcal{H},\alpha_t\in\mathfrak{R}^+} L(h_t, \alpha_t) = \sum_{i=1}^{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, f_{t-1} + \alpha_t h_t), \mathbf{y}_i)\}, \quad (8)$$

where $\mathcal{H}$ is the set of possible weak rankers, α_t is a positive weight, and $(f_{t-1} + \alpha_t h_t)(\vec{x}) = f_{t-1}(\vec{x}) + \alpha_t h_t(\vec{x})$. Several ways of computing coefficients α_t and weak rankers h_t may be considered. Following the idea of AdaBoost, in AdaRank we take the approach of 'forward stage-wise additive modeling' [12] and get the algorithm in Figure 1. It can be proved that there exists a lower bound on the ranking accuracy for AdaRank on training data, as presented in Theorem 1.

THEOREM 1. *The following bound holds on the ranking accuracy of the AdaRank algorithm on training data:*

$$\frac{1}{m}\sum_{i=1}^{m} E(\pi(q_i, \mathbf{d}_i, f_T), \mathbf{y}_i) \geq 1 - \prod_{t=1}^{T} e^{-\delta^t_{min}} \sqrt{1 - \varphi(t)^2},$$

where $\varphi(t) = \sum_{i=1}^{m} P_t(i) E(\pi(q_i, \mathbf{d}_i, h_t), \mathbf{y}_i)$, $\delta^t_{min} = \min_{i=1,\cdots,m} \delta^t_i$, *and*

$$\delta^t_i = E(\pi(q_i, \mathbf{d}_i, f_{t-1} + \alpha_t h_t), \mathbf{y}_i) - E(\pi(q_i, \mathbf{d}_i, f_{t-1}), \mathbf{y}_i) - \alpha_t E(\pi(q_i, \mathbf{d}_i, h_t), \mathbf{y}_i),$$

for all $i = 1, 2, \cdots, m$ *and* $t = 1, 2, \cdots, T$.

A proof of the theorem can be found in appendix. The theorem implies that the ranking accuracy in terms of the performance measure can be continuously improved, as long as $e^{-\delta^t_{min}} \sqrt{1 - \varphi(t)^2} < 1$ holds.

3.4 Advantages

AdaRank is a simple yet powerful method. More importantly, it is a method that can be justified from the theoretical viewpoint, as discussed above. In addition AdaRank has several other advantages when compared with the existing learning to rank methods such as Ranking SVM, RankBoost, and RankNet.

First, AdaRank can incorporate any performance measure, provided that the measure is query based and in the range of $[-1, +1]$. Notice that the major IR measures meet this requirement. In contrast the existing methods only minimize loss functions that are loosely related to the IR measures [16].

Second, the learning process of AdaRank is more efficient than those of the existing learning algorithms. The time complexity of AdaRank is of order $O((k+T)\cdot m\cdot n \log n)$, where k denotes the number of features, T the number of rounds, m the number of queries in training data, and n is the maximum number of documents for queries in training data. The time complexity of RankBoost, for example, is of order $O(T \cdot m \cdot n^2)$ [8].

Third, AdaRank employs a more reasonable framework for performing the ranking task than the existing methods. Specifically in AdaRank the instances correspond to queries, while in the existing methods the instances correspond to document pairs. As a result, AdaRank does not have the following shortcomings that plague the existing methods. (a) The existing methods have to make a strong assumption that the document pairs from the same query are independently distributed. In reality, this is clearly not the case and this problem does not exist for AdaRank. (b) Ranking the most relevant documents on the tops of document lists is crucial for document retrieval. The existing methods cannot focus on the training on the tops, as indicated in [4]. Several methods for rectifying the problem have been proposed (e.g., [4]), however, they do not seem to fundamentally solve the problem. In contrast, AdaRank can naturally focus on training on the tops of document lists, because the performance measures used favor rankings for which relevant documents are on the tops. (c) In the existing methods, the numbers of document pairs vary from query to query, resulting in creating models biased toward queries with more document pairs, as pointed out in [4]. AdaRank does not have this drawback, because it treats queries rather than document pairs as basic units in learning.

3.5 Differences from AdaBoost

AdaRank is a boosting algorithm. In that sense, it is similar to AdaBoost, but it also has several striking differences from AdaBoost.

First, the types of instances are different. AdaRank makes use of queries and their corresponding document lists as instances. The labels in training data are lists of ranks (relevance levels). AdaBoost makes use of feature vectors as instances. The labels in training data are simply $+1$ and -1.

Second, the performance measures are different. In AdaRank, the performance measure is a generic measure, defined on the document list and the rank list of a query. In AdaBoost the corresponding performance measure is a specific measure for binary classification, also referred to as 'margin' [25].

Third, the ways of updating weights are also different. In AdaBoost, the distribution of weights on training instances is calculated according to the current distribution and the performance of the current weak learner. In AdaRank, in contrast, it is calculated according to the performance of the ranking model created so far, as shown in Figure 1. Note that AdaBoost can also adopt the weight updating method used in AdaRank. For AdaBoost they are equivalent (cf., [12] page 305). However, this is not true for AdaRank.

3.6 Construction of Weak Ranker

We consider an efficient implementation for weak ranker construction, which is also used in our experiments. In the implementation, as weak ranker we choose the feature that has the optimal weighted performance among all of the features:

$$\max_{k} \sum_{i=1}^{m} P_t(i) E(\pi(q_i, \mathbf{d}_i, x_k), \mathbf{y}_i).$$

Creating weak rankers in this way, the learning process turns out to be that of repeatedly selecting features and linearly combining the selected features. Note that features which are not selected in the training phase will have a weight of zero.

4. EXPERIMENTAL RESULTS

We conducted experiments to test the performances of AdaRank using four benchmark datasets: OHSUMED, WSJ, AP, and .Gov.

Table 2: Features used in the experiments on OHSUMED, WSJ, and AP datasets. $C(w,d)$ represents frequency of word w in document d; C represents the entire collection; n denotes number of terms in query; $|\cdot|$ denotes the size function; and $idf(\cdot)$ denotes inverse document frequency.

1	$\sum_{w_i \in q \cap d} \ln(c(w_i,d)+1)$	2	$\sum_{w_i \in q \cap d} \ln(\frac{\|C\|}{c(w_i,C)}+1)$
3	$\sum_{w_i \in q \cap d} \ln(idf(w_i))$	4	$\sum_{w_i \in q \cap d} \ln(\frac{c(w_i,d)}{\|d\|}+1)$
5	$\sum_{w_i \in q \cap d} \ln(\frac{c(w_i,d)}{\|d\|} \cdot idf(w_i)+1)$	6	$\sum_{w_i \in q \cap d} \ln(\frac{c(w_i,d)\cdot\|C\|}{\|d\|\cdot c(w_i,C)}+1)$
7	ln(BM25 score)		

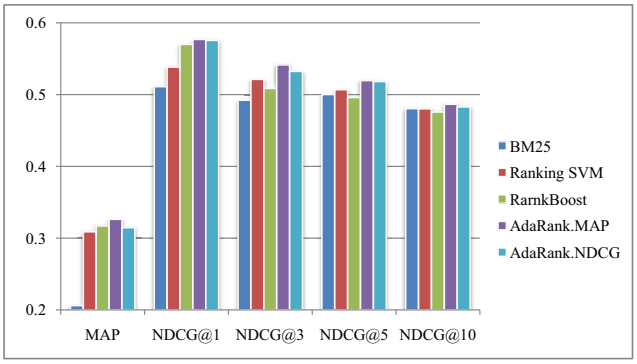

Figure 2: Ranking accuracies on OHSUMED data.

4.1 Experiment Setting

Ranking SVM [13, 16] and RankBoost [8] were selected as baselines in the experiments, because they are the state-of-the-art learning to rank methods. Furthermore, BM25 [24] was used as a baseline, representing the state-of-the-arts IR method (we actually used the tool Lemur[1]).

For AdaRank, the parameter T was determined automatically during each experiment. Specifically, when there is no improvement in ranking accuracy in terms of the performance measure, the iteration stops (and T is determined). As the measure E, MAP and NDCG@5 were utilized. The results for AdaRank using MAP and NDCG@5 as measures in training are represented as AdaRank.MAP and AdaRank.NDCG, respectively.

4.2 Experiment with OHSUMED Data

In this experiment, we made use of the OHSUMED dataset [14] to test the performances of AdaRank. The OHSUMED dataset consists of 348,566 documents and 106 queries. There are in total 16,140 query-document pairs upon which relevance judgments are made. The relevance judgments are either 'd' (*definitely relevant*), 'p' (*possibly relevant*), or 'n'(*not relevant*). The data have been used in many experiments in IR, for example [4, 29].

As features, we adopted those used in document retrieval [4]. Table 2 shows the features. For example, *tf* (term frequency), *idf* (inverse document frequency), *dl* (document length), and combinations of them are defined as features. BM25 score itself is also a feature. Stop words were removed and stemming was conducted in the data.

We randomly divided queries into four even subsets and conducted 4-fold cross-validation experiments. We tuned the parameters for BM25 during one of the trials and applied them to the other trials. The results reported in Figure 2 are those averaged over four trials. In MAP calculation, we define the rank 'd' as relevant and the other two ranks as irrelevant. From Figure 2, we see that both AdaRank.MAP and AdaRank.NDCG outperform BM25, Ranking SVM, and RankBoost in terms of all measures. We conducted significant tests (t-test) on the improvements of AdaRank.MAP over BM25, Ranking SVM, and RankBoost in terms of MAP. The results indicate that all the improvements are statistically significant (p-value < 0.05). We also conducted t-test on the improvements of AdaRank.NDCG over BM25, Ranking SVM, and RankBoost in terms of NDCG@5. The improvements are also statistically significant.

[1]http://www.lemurproject.com

Table 3: Statistics on WSJ and AP datasets.

Dataset	# queries	# retrieved docs	# docs per query
AP	116	24,727	213.16
WSJ	126	40,230	319.29

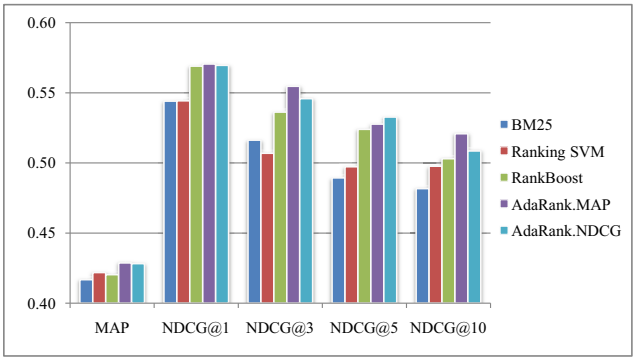

Figure 3: Ranking accuracies on WSJ dataset.

4.3 Experiment with WSJ and AP Data

In this experiment, we made use of the WSJ and AP datasets from the TREC ad-hoc retrieval track, to test the performances of AdaRank. WSJ contains 74,520 articles of Wall Street Journals from 1990 to 1992, and AP contains 158,240 articles of Associated Press in 1988 and 1990. 200 queries are selected from the TREC topics (No.101 ~ No.300). Each query has a number of documents associated and they are labeled as 'relevant' or 'irrelevant' (to the query). Following the practice in [28], the queries that have less than 10 relevant documents were discarded. Table 3 shows the statistics on the two datasets.

In the same way as in section 4.2, we adopted the features listed in Table 2 for ranking. We also conducted 4-fold cross-validation experiments. The results reported in Figure 3 and 4 are those averaged over four trials on WSJ and AP datasets, respectively. From Figure 3 and 4, we can see that AdaRank.MAP and AdaRank.NDCG outperform BM25, Ranking SVM, and RankBoost in terms of all measures on both WSJ and AP. We conducted t-tests on the improvements of AdaRank.MAP and AdaRank.NDCG over BM25, Ranking SVM, and RankBoost on WSJ and AP. The results indicate that all the improvements in terms of MAP are statistically significant (p-value < 0.05). However only some of the improvements in terms of NDCG@5 are statistically significant, although overall the improvements on NDCG scores are quite high (1-2 points).

4.4 Experiment with .Gov Data

In this experiment, we further made use of the TREC .Gov data to test the performance of AdaRank for the task of web retrieval. The corpus is a crawl from the .gov domain in early 2002, and has been used at TREC Web Track since 2002. There are a total

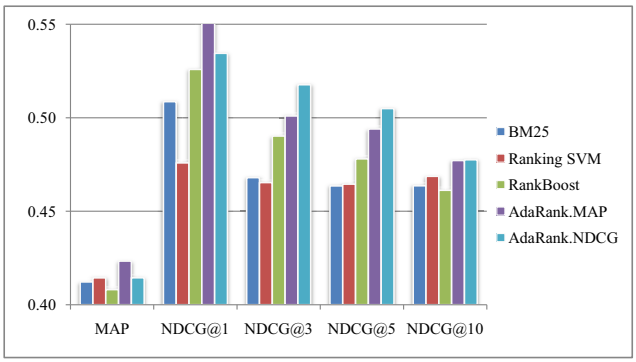

Figure 4: Ranking accuracies on AP dataset.

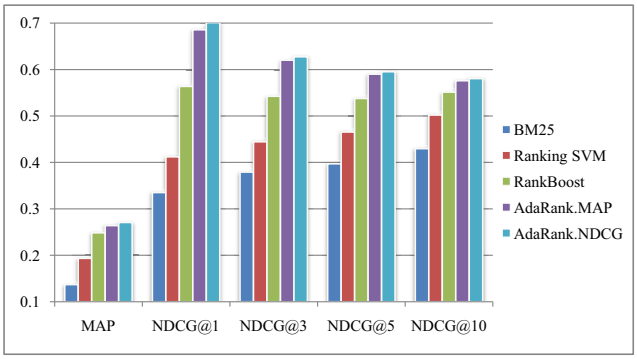

Figure 5: Ranking accuracies on .Gov dataset.

Table 4: Features used in the experiments on .Gov dataset.

1	BM25 [24]	2	MSRA1000 [27]
3	PageRank [21]	4	HostRank [30]
5	Relevance Propagation [23] (10 features)		

of 1,053,110 web pages with 11,164,829 hyperlinks in the data. The 50 queries in the topic distillation task in the Web Track of TREC 2003 [6] were used. The ground truths for the queries are provided by the TREC committee with binary judgment: relevant or irrelevant. The number of relevant pages vary from query to query (from 1 to 86).

We extracted 14 features from each query-document pair. Table 4 gives a list of the features. They are the outputs of some well-known algorithms (systems). These features are different from those in Table 2, because the task is different.

Again, we conducted 4-fold cross-validation experiments. The results averaged over four trials are reported in Figure 5. From the results, we can see that AdaRank.MAP and AdaRank.NDCG outperform all the baselines in terms of all measures. We conducted t-tests on the improvements of AdaRank.MAP and AdaRank.NDCG over BM25, Ranking SVM, and RankBoost. Some of the improvements are not statistically significant. This is because we have only 50 queries used in the experiments, and the number of queries is too small.

4.5 Discussions

We investigated the reasons that AdaRank outperforms the baseline methods, using the results of the OHSUMED dataset as examples.

First, we examined the reason that AdaRank has higher performances than Ranking SVM and RankBoost. Specifically we compared the error rates between different rank pairs made by Ranking SVM, RankBoost, AdaRank.MAP, and AdaRank.NDCG on the test data. The results averaged over four trials in the 4-fold cross validation are shown in Figure 6. We use 'd-n' to stand for the pairs between 'definitely relevant' and 'not relevant', 'd-p' the pairs between 'definitely relevant' and 'partially relevant', and 'p-n' the pairs between 'partially relevant' and 'not relevant'. From Figure 6, we can see that AdaRank.MAP and AdaRank.NDCG make fewer errors for 'd-n' and 'd-p', which are related to the tops of rankings and are important. This is because AdaRank.MAP and AdaRank.NDCG can naturally focus upon the training on the tops by optimizing MAP and NDCG@5, respectively.

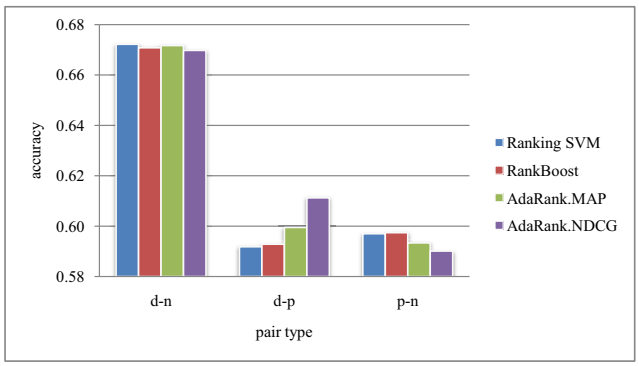

Figure 6: Accuracy on ranking document pairs with OHSUMED dataset.

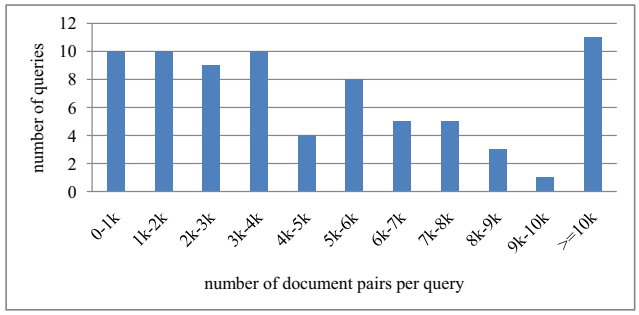

Figure 7: Distribution of queries with different number of document pairs in training data of trial 1.

We also made statistics on the number of document pairs per query in the training data (for trial 1). The queries are clustered into different groups based on the the number of their associated document pairs. Figure 7 shows the distribution of the query groups. In the figure, for example, '0-1k' is the group of queries whose number of document pairs are between 0 and 999. We can see that the numbers of document pairs really vary from query to query. Next we evaluated the accuracies of AdaRank.MAP and RankBoost in terms of MAP for each of the query group. The results are reported in Figure 8. We found that the average MAP of AdaRank.MAP over the groups is two points higher than RankBoost. Furthermore, it is interesting to see that AdaRank.MAP performs particularly better than RankBoost for queries with small numbers of document pairs (e.g., '0-1k', '1k-2k', and '2k-3k'). The results indicate that AdaRank.MAP can effectively avoid creating a model biased towards queries with more document pairs. For AdaRank.NDCG, similar results can be observed.

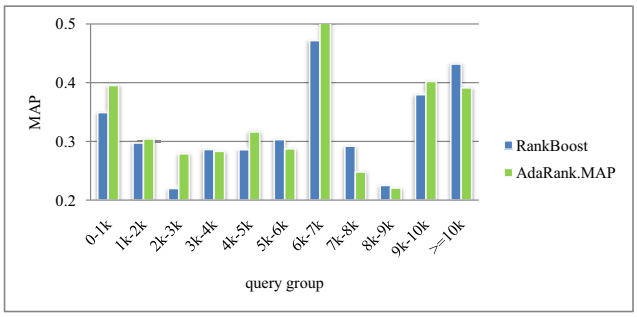

Figure 8: Differences in MAP for different query groups.

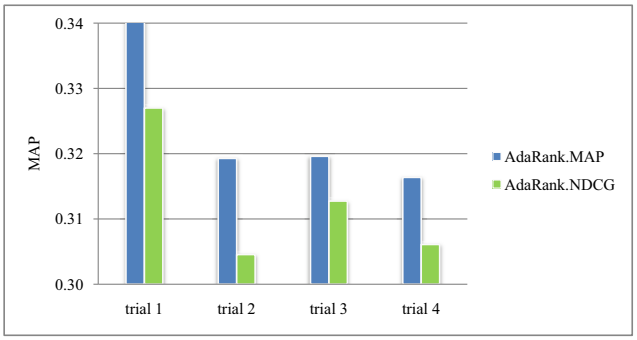

Figure 9: MAP on training set when model is trained with MAP or NDCG@5.

We further conducted an experiment to see whether AdaRank has the ability to improve the ranking accuracy in terms of a measure by using the measure in training. Specifically, we trained ranking models using AdaRank.MAP and AdaRank.NDCG and evaluated their accuracies on the training dataset in terms of both MAP and NDCG@5. The experiment was conducted for each trial. Figure 9 and Figure 10 show the results in terms of MAP and NDCG@5, respectively. We can see that, AdaRank.MAP trained with MAP performs better in terms of MAP while AdaRank.NDCG trained with NDCG@5 performs better in terms of NDCG@5. The results indicate that AdaRank can indeed enhance ranking performance in terms of a measure by using the measure in training.

Finally, we tried to verify the correctness of Theorem 1. That is, the ranking accuracy in terms of the performance measure can be continuously improved, as long as $e^{-\delta^t_{min}}\sqrt{1-\varphi(t)^2} < 1$ holds. As an example, Figure 11 shows the learning curve of AdaRank.MAP in terms of MAP during the training phase in one trial of the cross validation. From the figure, we can see that the ranking accuracy of AdaRank.MAP steadily improves, as the training goes on, until it reaches to the peak. The result agrees well with Theorem 1.

5. CONCLUSION AND FUTURE WORK

In this paper we have proposed a novel algorithm for learning ranking models in document retrieval, referred to as AdaRank. In contrast to existing methods, AdaRank optimizes a loss function that is *directly defined on the performance measures*. It employs a boosting technique in ranking model learning. AdaRank offers several advantages: ease of implementation, theoretical soundness, efficiency in training, and high accuracy in ranking. Experimental results based on four benchmark datasets show that AdaRank can significantly outperform the baseline methods of BM25, Ranking SVM, and RankBoost.

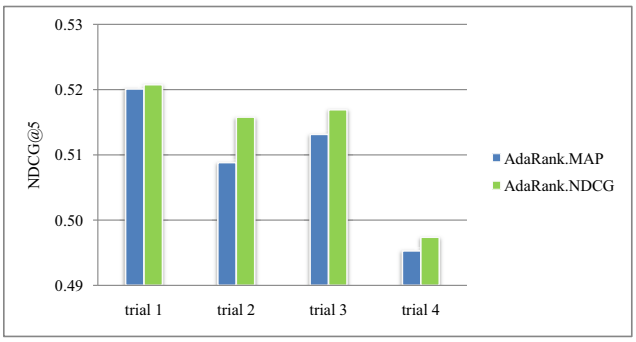

Figure 10: NDCG@5 on training set when model is trained with MAP or NDCG@5.

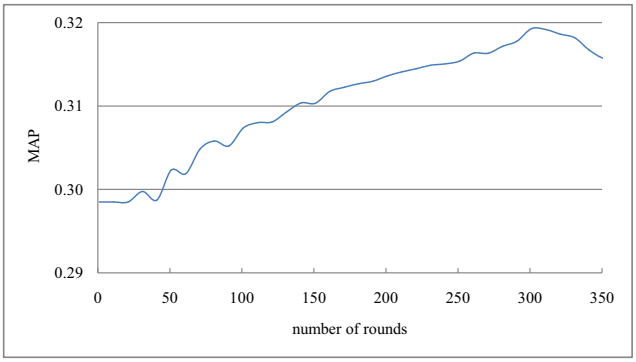

Figure 11: Learning curve of AdaRank.

Future work includes theoretical analysis on the generalization error and other properties of the AdaRank algorithm, and further empirical evaluations of the algorithm including comparisons with other algorithms that can directly optimize performance measures.

6. ACKNOWLEDGMENTS

We thank Harry Shum, Wei-Ying Ma, Tie-Yan Liu, Gu Xu, Bin Gao, Robert Schapire, and Andrew Arnold for their valuable comments and suggestions to this paper.

7. REFERENCES

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

[2] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems 18*, pages 395–402. MIT Press, Cambridge, MA, 2006.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML 22*, pages 89–96, 2005.

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *SIGIR 29*, pages 186–193, 2006.

[5] D. Cossock and T. Zhang. Subset ranking using regression. In *COLT*, pages 605–619, 2006.

[6] N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the TREC 2003 web track. In *TREC*, pages 78–92, 2003.

[7] N. Duffy and D. Helmbold. Boosting methods for regression. *Mach. Learn.*, 47(2-3):153–200, 2002.
[8] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.
[9] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.
[10] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.
[11] G. Fung, R. Rosales, and B. Krishnapuram. Learning rankings via convex hull separation. In *Advances in Neural Information Processing Systems 18*, pages 395–402. MIT Press, Cambridge, MA, 2006.
[12] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.
[13] R. Herbrich, T. Graepel, and K. Obermayer. *Large Margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.
[14] W. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *SIGIR*, pages 192–201, 1994.
[15] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR 23*, pages 41–48, 2000.
[16] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD 8*, pages 133–142, 2002.
[17] T. Joachims. A support vector method for multivariate performance measures. In *ICML 22*, pages 377–384, 2005.
[18] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR 24*, pages 111–119, 2001.
[19] D. A. Metzler, W. B. Croft, and A. McCallum. Direct maximization of rank-based metrics for information retrieval. Technical report, CIIR, 2005.
[20] R. Nallapati. Discriminative models for information retrieval. In *SIGIR 27*, pages 64–71, 2004.
[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
[22] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR 21*, pages 275–281, 1998.
[23] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. In *SIGIR 28*, pages 408–415, 2005.
[24] S. E. Robertson and D. A. Hull. The TREC-9 filtering track final report. In *TREC*, pages 25–40, 2000.
[25] R. E. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML 14*, pages 322–330, 1997.
[26] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Mach. Learn.*, 37(3):297–336, 1999.
[27] R. Song, J. Wen, S. Shi, G. Xin, T. yan Liu, T. Qin, X. Zheng, J. Zhang, G. Xue, and W.-Y. Ma. Microsoft Research Asia at web track and terabyte track of TREC 2004. In *TREC*, 2004.
[28] A. Trotman. Learning to rank. *Inf. Retr.*, 8(3):359–381, 2005.
[29] J. Xu, Y. Cao, H. Li, and Y. Huang. Cost-sensitive learning of SVM for ranking. In *ECML*, pages 833–840, 2006.
[30] G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *SIGIR 28*, pages 186–193, 2005.
[31] H. Yu. SVM selective sampling for ranking with application to data retrieval. In *SIGKDD 11*, pages 354–363, 2005.

APPENDIX

Here we give the proof of Theorem 1.

PROOF. Set $Z_T = \sum_{i=1}^{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, f_T), \mathbf{y}_i)\}$ and $\phi(t) = \frac{1}{2}(1 + \varphi(t))$. According to the definition of α_t, we know that $e^{\alpha_t} = \sqrt{\frac{\phi(t)}{1-\phi(t)}}$.

$$
\begin{aligned}
Z_T &= \sum_{i=1}^{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, f_{T-1} + \alpha_T h_T), \mathbf{y}_i)\} \\
&= \sum_{i=1}^{m} \exp\left\{-E(\pi(q_i, \mathbf{d}_i, f_{T-1}), \mathbf{y}_i) - \alpha_T E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i) - \delta_i^T\right\} \\
&\le \sum_{i=1}^{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, f_{T-1}), \mathbf{y}_i)\} \exp\{-\alpha_T E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i)\} e^{-\delta_{min}^T} \\
&= e^{-\delta_{min}^T} Z_{T-1} \sum_{i=1}^{m} \frac{\exp\{-E(\pi(q_i, \mathbf{d}_i, f_{T-1}), \mathbf{y}_i)\}}{Z_{T-1}} \exp\{-\alpha_T E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i)\} \\
&= e^{-\delta_{min}^T} Z_{T-1} \sum_{i=1}^{m} P_T(i) \exp\{-\alpha_T E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i)\}.
\end{aligned}
$$

Moreover, if $E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i) \in [-1, +1]$ then,

$$
\begin{aligned}
Z_T &\le e^{-\delta_{min}^T} Z_{T-1} \sum_{i=1}^{m} P_T(i) \left(\frac{1 + E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i)}{2} e^{-\alpha_T} + \frac{1 - E(\pi(q_i, \mathbf{d}_i, h_T), \mathbf{y}_i)}{2} e^{\alpha_T} \right) \\
&= e^{-\delta_{min}^T} Z_{T-1} \left(\phi(T) \sqrt{\frac{1-\phi(T)}{\phi(T)}} + (1 - \phi(T)) \sqrt{\frac{\phi(T)}{1-\phi(T)}} \right) \\
&= Z_{T-1} e^{-\delta_{min}^T} \sqrt{4\phi(T)(1-\phi(T))} \\
&\le Z_{T-2} \prod_{t=T-1}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&\le Z_1 \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&= m \sum_{i=1}^{m} \frac{1}{m} \exp\{-E(\pi(q_i, \mathbf{d}_i, \alpha_1 h_1), \mathbf{y}_i)\} \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&= m \sum_{i=1}^{m} \frac{1}{m} \exp\{-\alpha_1 E(\pi(q_i, \mathbf{d}_i, h_1), \mathbf{y}_i) - \delta_i^1\} \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&\le m e^{-\delta_{min}^1} \sum_{i=1}^{m} \frac{1}{m} \exp\{-\alpha_1 E(\pi(q_i, \mathbf{d}_i, h_1), \mathbf{y}_i)\} \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&\le m \left\{ e^{-\delta_{min}^1} \sqrt{4\phi(1)(1-\phi(1))} \right\} \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi(t)(1-\phi(t))} \\
&= m \prod_{t=1}^{T} e^{-\delta_{min}^t} \sqrt{1 - \varphi(t)^2}.
\end{aligned}
$$

$$
\begin{aligned}
\therefore \frac{1}{m} \sum_{i=1}^{m} E(\pi(q_i, \mathbf{d}_i, f_T), \mathbf{y}_i) &\ge \frac{1}{m} \sum_{i=1}^{m} \{1 - \exp(-E(\pi(q_i, \mathbf{d}_i, f_T), \mathbf{y}_i))\} \\
&\ge 1 - \prod_{t=1}^{T} e^{-\delta_{min}^t} \sqrt{1 - \varphi(t)^2}.
\end{aligned}
$$

□