Learning to Rank: From Pairwise Approach to Listwise Approach

Zhe Cao* CAOZHE@MAILS.THU.EDU.CN
Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, P. R. China

Tao Qin* TSINTAO@GMAIL.COM
Department of Electronic Engineering, Tsinghua University, Beijing, 100084, P. R. China

Tie-Yan Liu TYLIU@MICROSOFT.COM
Microsoft Research Asia, No.49 Zhichun Road, Haidian District, Beijing 100080, P. R. China

Ming-Feng Tsai* MFTSAI@NLG.CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan

Hang Li HANGLI@MICROSOFT.COM
Microsoft Research Asia, No.49 Zhichun Road, Haidian District, Beijing 100080, P. R. China

Abstract

The paper is concerned with learning to rank, which is to construct a model or a function for ranking objects. Learning to rank is useful for document retrieval, collaborative filtering, and many other applications. Several methods for learning to rank have been proposed, which take object pairs as 'instances' in learning. We refer to them as the pairwise approach in this paper. Although the pairwise approach offers advantages, it ignores the fact that ranking is a prediction task on list of objects. The paper postulates that learning to rank should adopt the listwise approach in which lists of objects are used as 'instances' in learning. The paper proposes a new probabilistic method for the approach. Specifically it introduces two probability models, respectively referred to as permutation probability and top k probability, to define a listwise loss function for learning. Neural Network and Gradient Descent are then employed as model and algorithm in the learning method. Experimental results on information retrieval show that the proposed listwise approach performs better than the pairwise approach.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

*This work was done when the first, second and fourth authors were visiting Microsoft Research Asia.

1. Introduction

The central issues of many applications are ranking. These include document retrieval, collaborative filtering, expert finding, anti web spam, sentiment analysis, and product rating. In this paper, we address learning to rank and without loss of generality we take document retrieval as example.

Learning to rank, when applied to document retrieval, is a task as follows. Assume that there is a collection of documents. In retrieval (i.e., ranking), given a query, the ranking function assigns a score to each document, and ranks the documents in descending order of the scores. The ranking order represents the relevance of documents with respect to the query. In learning, a number of queries are provided; each query is associated with a perfect ranking list of documents; a ranking function is then created using the training data, such that the model can precisely predict the ranking lists in the training data.

Due to its importance, learning to rank has been drawing broad attention in the machine learning community recently. Several methods based on what we call the pairwise approach have been developed and successfully applied to document retrieval. This approach takes document pairs as instances in learning, and formalizes the problem of learning to rank as that of classification. Specifically, in learning it collects document pairs from the ranking lists, and for each document pair it assigns a label representing the relative relevance of the two documents. It then trains a classification model with the labeled data and makes use of the classification model in ranking. The uses of Support Vector Machines (SVM), Boosting, and Neural Network as the

classification model lead to the methods of Ranking SVM (Herbrich et al., 1999), RankBoost (Freund et al., 1998), and RankNet (Burges et al., 2005).

There are advantages with taking the pairwise approach. First, existing methodologies on classification can be directly applied. Second, the training instances of document pairs can be easily obtained in certain scenarios (Joachims, 2002). However, there are also problems with the approach. First, the objective of learning is formalized as minimizing errors in classification of document pairs, rather than minimizing errors in ranking of documents. Second, the assumption of that the document pairs are generated i.i.d. is also too strong. Third, the number of generated document pairs varies largely from query to query, which will result in training a model biased toward queries with more document pairs (Cao et al., 2006)

In this paper, we propose employing what we call the listwise approach, in which document lists instead of document pairs are used as instances in learning. The major question then is how to define a listwise loss function, representing the difference between the ranking list output by a ranking model and the ranking list given as ground truth.

We propose a probabilistic method to calculate the listwise loss function. Specifically we transform both the scores of the documents assigned by a ranking function and the explicit or implicit judgments of the documents given by humans into probability distributions. We can then utilize any metric between the probability distributions as the loss function. We consider the uses of two models for the transformation; one is referred to as permutation probability and the other top k probability.

We then propose a learning to rank method using the listwise loss function, with Neural Network as model and Gradient Descent as algorithm. We refer to it as ListNet.

We applied ListNet to document retrieval and compared the results of it with those of existing pairwise methods including Ranking SVM, RankBoost, and RankNet. The results on three data sets show that our method outperforms the existing methods, suggesting that it is better to employ the listwise approach than the pairwise approach in learning to rank.

The major contributions of this paper include (1) proposal of the listwise approach, (2) formulation of the listwise loss function on the basis of probability models, (3) development of the ListNet method, (4) empirical verification of the effectiveness of the approach.

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 gives a general description on the listwise approach to learning to rank. Probability models for defining a listwise loss function are introduced in Section 4 and the learning method ListNet is explained in Section 5. Section 6 reports our experimental results. Finally, Section 7 makes conclusions.

2. Related Work

2.1. Learning to Rank

Learning to rank is a new and popular topic in machine learning. There is one major approach to learning to rank, referred to as the pairwise approach in this paper. For other approaches, see (Shashua & Levin, 2002; Crammer & Singer, 2001; Lebanon & Lafferty, 2002), for example.

In the pairwise approach, the learning task is formalized as classification of object pairs into two categories (correctly ranked and incorrectly ranked). Herbrich et al. (1999) proposed employing the approach and using the SVM techniques to build the classification model. The method is referred to as Ranking SVM. Freund et al. (1998) proposed performing the task in the same way but by means of Boosting. Burges et al. (2005) also adopted the approach and developed a method called RankNet. They employed Cross Entropy as loss function and Gradient Descent as algorithm to train a Neural Network model.

Learning to rank, particularly the pairwise approach, has been successively applied to information retrieval. For instance, Joachims (2002) applied Ranking SVM to document retrieval. He developed a method of deriving document pairs for training, from users' clicks-through data. Burges et al. (2005) applied RankNet to large scale web search. Cao et al. (2006) adapted Ranking SVM to document retrieval by modifying the loss function. See also (Qin et al., 2007; Tsai et al., 2007).

2.2. Probability Models on Ranking

In statistics, probability models for representing ranking lists of objects and methods for estimation of the models have been proposed. For example, following the work by Luce (1959), Plackett (1975) defined probability models on ranking lists of objects. He further proposed a method for estimating the models. In this paper, we make use of similar probability distributions. However, the underlying structures (i.e., parameters) and the fundamental usages (i.e., transformation of scores to probability distributions) of our models differ from those of Plackett's.

3. Listwise Approach

In this section, we give a general description on learning to rank, with document retrieval as example. Particularly we describe in details the listwise approach. In this paper, we use superscript to denote the id of a query and subscript to denote the id of a document.

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 documents $d^{(i)} = \left(d_1^{(i)}, d_2^{(i)}, \cdots, d_{n^{(i)}}^{(i)}\right)$, where $d_j^{(i)}$ denotes the j-th document and $n^{(i)}$ denotes the sizes of $d^{(i)}$. Furthermore, each list of documents $d^{(i)}$ is associated with a list of judgments (scores) $y^{(i)} = \left(y_1^{(i)}, y_2^{(i)}, \cdots, y_{n^{(i)}}^{(i)}\right)$ where $y_j^{(i)}$ denotes the judgment on document $d_j^{(i)}$ with respect to query $q^{(i)}$. The judgment $y_j^{(i)}$ represents the relevance degree of $d_j^{(i)}$ to $q^{(i)}$, and can be a score explicitly or implicitly given by humans. For example, $y_j^{(i)}$ can be the number of clicks on $d_j^{(i)}$ when $d_j^{(i)}$ is retrieved and returned for query $q^{(i)}$ at a search engine (Joachims, 2002). The assumption is that the higher click-on rate is observed for $d_j^{(i)}$ and $q^{(i)}$ the stronger relevance exists between them.

A feature vector $x_j^{(i)} = \Psi(q^{(i)}, d_j^{(i)})$ is created from each query-document pair $(q^{(i)}, d_j^{(i)}), i = 1, 2, \cdots, m; j = 1, 2, \cdots, n^{(i)}$. Each list of features $x^{(i)} = \left(x_1^{(i)}, \cdots, x_{n^{(i)}}^{(i)}\right)$ and the corresponding list of scores $y^{(i)} = \left(y_1^{(i)}, \cdots, y_{n^{(i)}}^{(i)}\right)$ then form an 'instance'. The training set can be denoted as $\mathcal{T} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$.

We then create a ranking function f; for each feature vector $x_j^{(i)}$ (corresponding to document $d_j^{(i)}$), it outputs a score $f(x_j^{(i)})$. For the list of feature vectors $x^{(i)}$ we obtain a list of scores $z^{(i)} = \left(f(x_1^{(i)}), \cdots, f(x_{n^{(i)}}^{(i)})\right)$. The objective of learning is formalized as minimization of the total losses with respect to the training data.

$$\sum_{i=1}^{m} L(y^{(i)}, z^{(i)}) \tag{1}$$

where L is a listwise loss function.

In ranking, when a new query $q^{(i')}$ and its associated documents $d^{(i')}$ are given, we construct feature vectors $x^{(i')}$ from them and use the trained ranking function to assign scores to the documents $d^{(i')}$. Finally we rank the documents $d^{(i')}$ in descending order of the scores. We call the learning problem described above as the listwise approach to learning to rank.

By contrast, in the pairwise approach, a new training data set $\mathcal{T}'$ is created from $\mathcal{T}$, in which each feature vector pair $x_j^{(i)}$ and $x_k^{(i)}$ forms a new instance where $j \neq k$, and $+1$ is assigned to the pair if $y_j^{(i)}$ is larger than $y_k^{(i)}$ otherwise -1. It turns out that the training data $\mathcal{T}'$ is a data set of binary classification. A classification model like SVM can be created. As explained in Section 1, although the pairwise approach has advantages, it also suffers from drawbacks. The listwise approach can naturally deal with the problems, which will be made clearer in Section 6.

4. Probability Models

We propose using two probability models to calculate the listwise loss function in Eq. (1). Specifically, we map a list of scores to a probability distribution using one of the two probability models and then take any metric between probability distributions as a loss function . The two models are referred to as permutation probability and top k probability.

4.1. Permutation Probability

Suppose that the set of objects to be ranked are identified with the numbers $1, 2, ..., n$. A permutation π on the objects is defined as a bijection from $\{1, 2, ..., n\}$ to itself. We write the permutation as $\pi = \langle \pi(1), \pi(2), ..., \pi(n) \rangle$. Here, $\pi(j)$ denotes the object at position j in the permutation. The set of all possible permutations of n objects is denoted as Ω_n.

Suppose that there is a ranking function which assigns scores to the n objects. We use s to denote the list of scores $s = (s_1, s_2, ..., s_n)$, where s_j is the score of the j-th object. Hereafter we sometimes make interchangeable the ranking function and the list of scores given by the ranking function.

We assume that there is uncertainty in the prediction of ranking lists (permutations) using the ranking function. In other words, any permutation is assumed to be possible, but different permutations may have different likelihood values calculated based on the ranking function. We define the permutation probability, so that it has desirable properties for representing the likelihood of a permutation (ranking list), given the ranking function.

Definition 1 *Suppose that π is a permutation on n objects, and $\phi(.)$ is an increasing and strictly positive function. Then, the probability of permutation π given the list of scores s is defined as*

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})}$$

where $s_{\pi(j)}$ denotes the score of object at position j of permutation π.

Let us consider an example with three objects $\{1, 2, 3\}$ having scores $s = (s_1, s_2, s_3)$. The probabilities of permutations $\pi = \langle 1, 2, 3 \rangle$ and $\pi' = \langle 3, 2, 1 \rangle$ are calculated as follows:

$$P_s(\pi) = \frac{\phi(s_1)}{\phi(s_1) + \phi(s_2) + \phi(s_3)} \cdot \frac{\phi(s_2)}{\phi(s_2) + \phi(s_3)} \cdot \frac{\phi(s_3)}{\phi(s_3)}.$$

$$P_s(\pi') = \frac{\phi(s_3)}{\phi(s_1) + \phi(s_2) + \phi(s_3)} \cdot \frac{\phi(s_2)}{\phi(s_2) + \phi(s_1)} \cdot \frac{\phi(s_1)}{\phi(s_1)}.$$

Lemma 2 *The permutation probabilities $P_s(\pi)$, $\pi \in \Omega_n$ form a probability distribution over the set of permuta-*

tions, i.e., for each $\pi \in \Omega_n$, *we have* $P_s(\pi) > 0$, *and* $\sum_{\pi \in \Omega_n} P_s(\pi) = 1$.

Theorem 3 *Given any two permutations* π *and* $\pi' \in \Omega_n$, *if (1)* $\pi(p) = \pi'(q), \pi(q) = \pi'(p), p < q$; *(2)* $\pi(r) = \pi'(r), r \neq p, q$; *(3)* $s_{\pi(p)} > s_{\pi(q)}$, *then* $P_s(\pi) > P_s(\pi')$.

Theorem 4 *For the n objects, if* $s_1 > s_2 > ... > s_n$, *then* $P_s(\langle 1, 2, ..., n\rangle)$ *is the highest permutation probability and* $P_s(\langle n, n-1, ..., 1\rangle)$ *is the lowest permutation probability among the permutation probabilities of the n objects.*

It is easy to verify that Theorem 4 holds. Proofs for Lemma 2 and Theorem 3 can be found in our technical report (Cao et al., 2007). Theorem 3 indicates that, for a permutation in which an object with a larger score is ranked ahead of another object with a smaller score, if we exchange their positions, the permutation probability of the resulting permutation will be lower than that of the original permutation. Theorem 4 indicates given the scores of *n* objects, the list of objects sorted in descending order of the scores has the highest permutation probability, while the list of objects sorted in ascending order has the lowest permutation probability.

Theorem 5 *(1) For linear function* $\phi(x) = \alpha x, \alpha > 0$, *the permutation probability is scale invariant:*

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})} = P_{\lambda s}(\pi) = \prod_{j=1}^{n} \frac{\phi(\lambda s_{\pi(j)})}{\sum_{k=j}^{n} \phi(\lambda s_{\pi(k)})},$$

$\forall \lambda > 0$. *Here* λs *means each component of score list s is multiplied by a positive constant* λ.

(2) For exponential function $\phi(x) = \exp(x)$, *the permutation probability is translation invariant:*

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})} = P_{\lambda+s}(\pi) = \prod_{j=1}^{n} \frac{\phi(\lambda + s_{\pi(j)})}{\sum_{k=j}^{n} \phi(\lambda + s_{\pi(k)})}$$

$\forall \lambda \in \mathcal{R}$. *Here* $\lambda + s$ *means adding a constant* λ *to each component of score list s.*

Given two lists of scores, we can first calculate the two corresponding permutation probability distributions, and then take the metric between the two distributions as the listwise loss function. Since the number of permutations is of order $O(n!)$, the calculation might be intractable in practice. To cope with the problem, we consider the use of top *k* probability. (We note that although the calculation of permutation probabilities is intractable, the notion itself is still valuable for the studies on learning to rank.)

4.2. Top *k* Probability

The top *k* probability of objects $(j_1, j_2, ..., j_k)$ represents the probability of their being ranked on the top *k* positions, given the scores of all the objects. Before giving the definition of top *k* probability, we first define the top *k* subgroup of permutations.

Definition 6 *The top k subgroup* $\mathcal{G}_k(j_1, j_2, ..., j_k)$ *contains all the permutations in which the top k objects are exactly* $(j_1, j_2, ..., j_k)$:

$$\mathcal{G}_k(j_1, j_2, ..., j_k) = \{\pi \in \Omega_n | \pi(t) = j_t, \forall t = 1, 2, ..., k\},$$

and $\mathcal{G}_k$ *is the collection of all top k subgroups:*

$$\mathcal{G}_k = \{\mathcal{G}_k(j_1, j_2, ..., j_k) | j_t = 1, 2, ..., n, \forall t = 1, 2, ..., k, \text{ and } j_u \neq j_v, \forall u \neq v\} \quad (2)$$

Note that there are in total $\frac{n!}{(n-k)!}$ elements in the collection $\mathcal{G}_k$; the number is much smaller than the number of elements in Ω_n.

Definition 7 *The top k probability of objects* $(j_1, j_2, ..., j_k)$ *is the probability of subgroup* $\mathcal{G}_k(j_1, j_2, ..., j_k)$:

$$P_s(\mathcal{G}_k(j_1, j_2, ..., j_k)) = \sum_{\pi \in \mathcal{G}_k(j_1, j_2, ..., j_k)} P_s(\pi),$$

where $P_s(\pi)$ *is permutation probability of* π *given s.*

That is, the top *k* probability of objects $(j_1, j_2, ..., j_k)$ equals the sum of the permutation probabilities of permutations in which objects $(j_1, j_2, ..., j_k)$ are ranked on the top *k* positions.

One may argue that from Definition 7, in order to calculate the $\frac{n!}{(n-k)!}$ top *k* probabilities, we still need to calculate *n*! permutation probabilities. Theorem 8 shows that we can do the calculation in a different way, which is efficient.

Theorem 8 *For top k probability* $P_s(\mathcal{G}_k(j_1, j_2, ..., j_k))$, *we have*

$$P_s(\mathcal{G}_k(j_1, j_2, ..., j_k)) = \prod_{t=1}^{k} \frac{\phi(s_{j_t})}{\sum_{l=t}^{n} \phi(s_{j_l})},$$

where s_{j_t} *is the score of object* j_t *which is ranked in position* t, $t = 1, 2, ..., n$.

Lemma 9 *Top k probabilities form a probability distribution over collection* $\mathcal{G}_k$.

Theorem 10 *Given any two objects* j_u *and* j_v, *if* $s_{j_u} > s_{j_v}$, $u \neq v$, $u, v = 1, 2, ..., n$, *then* $P_s(\mathcal{G}_k(j_1, ..., j_u, ..., j_v, ..., j_k)) > P_s(\mathcal{G}_k(j_1, ..., j_v, ..., j_u, ..., j_k))$.

Due to space limitation, we omit the proofs of Theorem 8, Lemma 9, and Theorem 10. They can be found in our technical report (Cao et al., 2007).

Theorem 10 shows the desirable properties of top *k* probability. We can also prove that top *k* probability is scale

invariant or translation invariant with a carefully designed function $\phi(.)$. We omit the details here.

Given two lists of scores, we can define the metric between the corresponding top k probability distributions as the listwise loss function. For example, when we use Cross Entropy as metric, the listwise loss function in Eq. (1) becomes

$$L(y^{(i)}, z^{(i)}) = -\sum_{\forall g \in \mathcal{G}_k} P_{y^{(i)}}(g) \log(P_{z^{(i)}}(g)) \quad (3)$$

5. Learning Method: ListNet

We propose a new learning method for optimizing the listwise loss function based on top k probability, with Neural Network as model and Gradient Descent as optimization algorithm. We refer to the method as ListNet.

Again, let us take document retrieval as example. We denote the ranking function based on the Neural Network model ω as f_ω. Given a feature vector $x_j^{(i)}$, $f_\omega(x_j^{(i)})$ assigns a score to it. We define ϕ in Definition 1 as an exponential function, which is translation invariant as shown in Theorem 5. We then rewrite the top k probability in Theorem 8 as

$$P_s(\mathcal{G}_k(j_1, j_2, ..., j_k)) = \prod_{t=1}^{k} \frac{\exp(s_{j_t})}{\sum_{l=t}^{n^{(i)}} \exp(s_{j_l})},$$

Given query $q^{(i)}$, the ranking function f_ω can generate a score list $z^{(i)}(f_\omega) = \left(f_\omega(x_1^{(i)}), f_\omega(x_2^{(i)}), \cdots, f_\omega(x_{n^{(i)}}^{(i)})\right)$. Then the top k probability of documents $(d_{j_1}^{(i)}, d_{j_2}^{(i)}, ..., d_{j_k}^{(i)})$ is calculated as

$$P_{z^{(i)}(f_\omega)}(\mathcal{G}_k(j_1, j_2, ..., j_k)) = \prod_{t=1}^{k} \frac{\exp(f_\omega(x_{j_t}^{(i)}))}{\sum_{l=t}^{n^{(i)}} \exp(f_\omega(x_{j_l}^{(i)}))},$$

With Cross Entropy as metric, the loss for query $q^{(i)}$ becomes

$$L(y^{(i)}, z^{(i)}(f_\omega)) = -\sum_{\forall g \in \mathcal{G}_k} P_{y^{(i)}}(g) \log(P_{z^{(i)}(f_\omega)}(g)) \quad (4)$$

The gradient of $L(y^{(i)}, z^{(i)}(f_\omega))$ with respect to parameter ω can be calculated as follows

$$\triangle\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial \omega} = -\sum_{\forall g \in \mathcal{G}_k} \frac{\partial P_{z^{(i)}(f_\omega)}(g)}{\partial \omega} \frac{P_{y^{(i)}}(g)}{P_{z^{(i)}(f_\omega)}(g)} \quad (5)$$

Eq.(5) is then used in Gradient Descent. Algorithm 1 shows the learning algorithm of ListNet.

Notice that ListNet is similar to RankNet. The only major difference lies in that the former uses document lists as instances while the latter uses document pairs as instances; the former utilizes a listwise loss function while the latter utilizes a pairwise loss function. Interestingly, when there are only two documents for each query, i.e., $n^{(i)} = 2$, then the listwise loss function in ListNet becomes equivalent to the pairwise loss function in RankNet.

Algorithm 1 Learning Algorithm of ListNet

Input: training data $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
Parameter: number of iterations T and learning rate η
Initialize parameter ω
for $t = 1$ **to** T **do**
  for $i = 1$ **to** m **do**
    Input $x^{(i)}$ of query $q^{(i)}$ to Neural Network and compute score list $z^{(i)}(f_\omega)$ with current ω
    Compute gradient $\triangle\omega$ using Eq. (5)
    Update $\omega = \omega - \eta \times \triangle\omega$
  end for
end for
Output Neural Network model ω

In our experiments in this paper, we implemented ListNet with $k = 1$. With some derivation (Cao et al., 2007), we can see that for $k = 1$ we have

$$\begin{aligned}\triangle\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial \omega} &= -\sum_{j=1}^{n^{(i)}} P_{y^{(i)}}(x_j^{(i)}) \frac{\partial f_\omega(x_j^{(i)})}{\partial \omega} \\ &+ \frac{1}{\sum_{j=1}^{n^{(i)}} \exp(f_\omega(x_j^{(i)}))} \sum_{j=1}^{n^{(i)}} \exp(f_\omega(x_j^{(i)})) \frac{\partial f_\omega(x_j^{(i)})}{\partial \omega}\end{aligned} \quad (6)$$

For simplicity, we use a linear Neural Network model and omit the constant b in the model[1]:

$$f_\omega(x_j^{(i)}) = \langle \omega, x_j^{(i)} \rangle$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product.

6. Experimental Results

We compared the ranking accuracies of ListNet with those of three baseline methods: RankNet, Ranking SVM, and RankBoost using three data sets.

6.1. Data Collections

We used three data sets in the experiments: TREC, a data set obtained from web track of TREC 2003 (Craswell et al., 2003); OHSUMED, a benchmark data set for document retrieval (Hersh et al., 1994); and CSearch, a data set from a commercial search engine.

TREC consists of web pages crawled from the .gov domain in early 2002. There are in total 1,053,110 pages

[1] Note that Eq. (5) and Algorithm 1 can be applied to any continuous ranking function.

and 11,164,829 hyperlinks in the data set. It also contains 50 queries from the topic distillation task in Web Track of TREC 2003. The relevance judgments (relevant or irrelevant) on the web pages with respect to the queries are given. There are about 20 features extracted from each query document pair, including content features and hyperlink features.

OHSUMED (Hersh et al., 1994) is a collection of documents and queries on medicine, consisting 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 definitely relevant, possibly relevant, or not relevant. The standard features in document retrieval (Nallapati, 2004) are extracted for each query-document pair. There are 30 features in total.

CSearch is a data set from a commercial web search engine. It contains about 25,000 queries, and each query has one thousand associated documents. There are about 600 features in total for each query-document pair, including query dependent features and independent features. This data set provides five levels of relevance judgments, ranging from 4 (”perfect match”) to 0 (”bad match”).

To obtain the ground truth (i.e., the ranking list of true scores) for each query, we simply use the 'ranks' (relevance judgments) of the related documents, provided in the data sets[2].

In ranking performance evaluation, we adopted two common IR evaluation measures: Normalized Discounted Cumulative Gain (NDCG) (Jarvelin & Kekanainen, 2000) and Mean Average Precision (MAP)(Baeza-Yates & Ribeiro-Neto, 1999). NDCG is designed to measure ranking accuracy when there are more than two levels of relevance judgments. For MAP it is assumed that there are two levels: relevant and irrelevant. In calculation of MAP for OHSUMED, we treated 'definitive relevant' as relevant and the other two levels as irrelevant. For CSearch, we only used NDCG.

6.2. Ranking Accuracy

For TREC and OHSUMED we divided each data set into five subsets, and conducted 5-fold cross-validation. In each trial, three folds were used for training, one fold for validation, and one fold for testing. For RankNet and ListNet the validation set in each trial was used to determine the number of iterations. For Ranking SVM it was used to tune the coefficient C and for RankBoost it was used for selection of the number of weak learners. The accuracies we report in this section are those averaged over five trials.

[2]We note that this is only one way for creating training data in the listwise approach. If pairwise data is available (such as clickthrough Joachims (2002)), then we can use them to create training data as well, for example, by employing the algorithm proposed in Cohen et al. (1998)).

Table 1. Ranking accuracies in terms of MAP

ALGORITHMS	LISTNET	RANKBOOST	RANKSVM	RANKNET
TREC	0.216	0.174	0.193	0.197
OHSUMED	0.305	0.297	0.297	0.303

Figure 1 and Table 1 give the results for TREC. We can see that ListNet outperforms the three baseline methods of RankNet, Ranking SVM, and RankBoost in terms of all measures. Especially for NDCG@1 and NDCG@2, ListNet achieves more than 4 point gain, which is about 10% relative improvement.

Figure 2 and Table 1 show the results for OHSUMED. Again, ListNet outperforms RankNet and RankBoost in terms of all measures. Moreover, ListNet works better than Ranking SVM in terms of NDCG@1, NDCG@2, NDCG@4 and MAP, with exceptions of NDCG@3 and NDCG@5.

CSearch is a large data set, and thus we did not conduct cross-validation. Instead, we randomly selected one third of the data for training, one third for validation, and the remaining one third for testing. Figure 3 shows the results of ListNet, RankNet and RankBoost. Again, ListNet outperforms RankNet and RankBoost in terms of all measures. Since the size of training data is large, we were not able to run Ranking SVM with the SVMlight tool (Joachims, 1999).

6.3. Discussions

We investigated why the listwise method ListNet can outperform the pairwise methods of RankNet, Ranking SVM, and RankBoost.

As explained in Section 1, for the pairwise approach the number of document pairs varies largely from query to query. As a result, the trained model may be biased toward those queries with more document pairs. We observed the tendencies in all data sets. As example, Table 2 shows the distribution of the number of document pairs per query in OHSUMED. We can see that the distribution is skewed: most queries only have a small number of document pairs (e.g. less than 5, 000), while a few queries have a large number of document pairs (e.g. more than 15,000). In the listwise approach the loss function is defined on each query, the problem does not exist. This appears to be one of the reasons for the higher performance by ListNet.

The pairwise approach actually employs a 'pairwise' loss function, which might be too loose as an approximation of

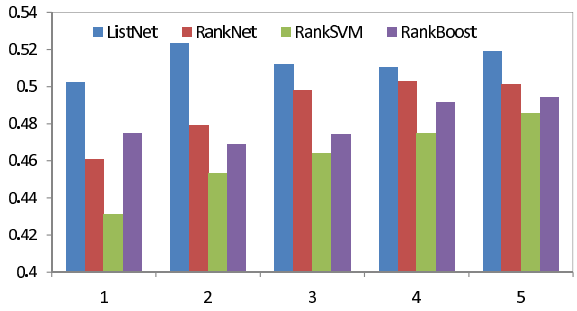

Figure 1. Ranking accuracies in terms of NDCG@n on TREC

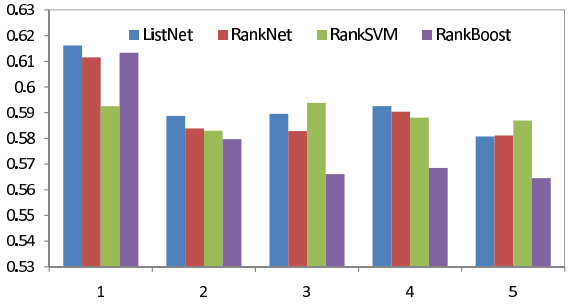

Figure 2. Ranking accuracies in terms of NDCG@n on OHSUMED

the performance measures of NDCG and MAP. By contrast, the listwise loss function used in the listwise approach can more properly represent the performance measures. This appears to be another reason that ListNet outperforms RankNet, etc. To verify the correctness of the claim, we further examined the optimization processes of the two methods. We looked at the correlations between the loss functions used by ListNet and RankNet and the measure of NDCG during the learning phase. Note that the major difference between the two methods is the loss function. The results using the TREC data are shown in Figures 4 and 5. From the figures, we can see that the pairwise loss of RankNet does not inversely correlate with NDCG. From iteration 20 to iteration 50, NDCG@5 increases while the pairwise loss of RankNet decreases. However, after iteration 60, NDCG@5 starts to drop, although pairwise loss is still decreasing. In contrast, the listwise loss of ListNet completely inversely correlates with NDCG. More specifically, from iteration 20 to iteration 50, listwise loss decreases, NDCG@5 increases accordingly. After iteration 50, listwise loss reaches its limit, while NDCG@5 also converges. Another point is that pairwise loss converges more slowly than listwise loss, which means RankNet needs run more iterations in training than ListNet. Similar trends were observed on the results evaluated in terms of MAP.

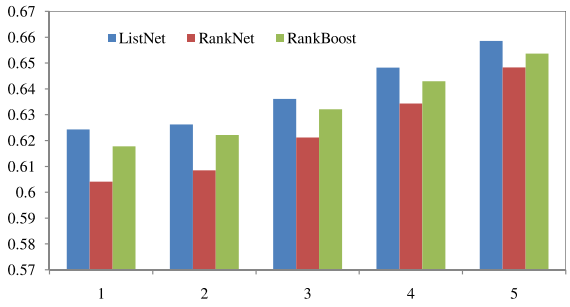

Figure 3. Ranking accuracies in terms of NDCG@n on CSearch

Table 2. Document-pair number distribution

Pair Number	Query Number
<5000	61
<10000	29
<15000	8
<20000	6
>=20000	2

We conclude that the listwise approach is more effective than the pairwise approach for learning to rank.

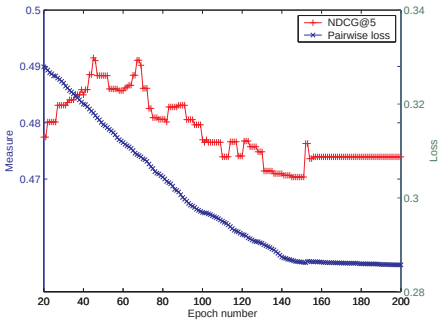

Figure 4. Pairwise loss v.s. NDCG@5 in RankNet

7. Conclusions

In this paper, we have proposed a new approach to learning to rank, referred to as the listwise approach. We argue that it is better to take this approach than the traditional pairwise approach in learning to rank. In the listwise approach, instead of using object pairs as instances, we use list of objects as instances in learning.

The key issue for the listwise approach is to define a listwise loss function. In this paper, we have proposed employing a probabilistic method to solve the problem. Specifically, we make use of probability models: permutation probability and top k probability to transform ranking scores into probability distributions. We can then utilize any metric between probability distributions (e.g., Cross

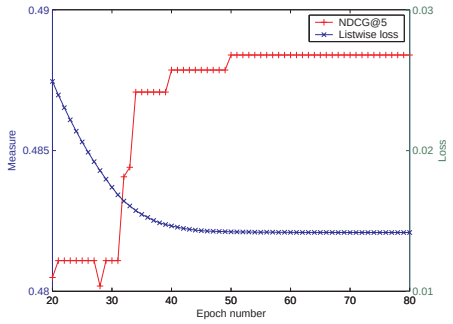

Figure 5. Listwise loss v.s. NDCG@5 in ListNet

Entropy) as the listwise loss function.

We have then developed a learning method based on the approach, using Neural Network and Gradient Descent. Experimental results with three data sets show that the method works better than the existing pairwise methods of RanNet, Ranking SVM, and RankBoost, suggesting that it is better to take the listwise approach in learning to rank.

As future work, we plan to study the effects of using other metrics and models in the listwise approach. We also intend to investigate the relationship between listwise loss functions and performance measures such as NDCG and MAP used in information retrieval.

Acknowledgments

We would like to express our gratitude to Wei-Ying Ma, Bin Gao, Jun Xu, Yu-Ting Liu, Yan-Yan Lan, Xiu-Bo Geng, and the anonymous reviewers to this paper for their valuable comments and suggestions. We would also like to thank Kai Yi for his help in our experiments.

References

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Addison Wesley.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. (2005). Learning to rank using gradient descent. *Proceedings of ICML 2005* (pp. 89–96).

Cao, Y. B., Xu, J., Liu, T. Y., Li, H., Huang, Y. L., & Hon, H. W. (2006). Adapting ranking SVM to document retrieval. *Proceedings of SIGIR 2006* (pp. 186–193).

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). *Learning to rank: From pairwise approach to listwise approach* (Technical Report MSR-TR-2007-40). Microsoft Corporation.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1998). Learning to order things. *Advances in Neural Information Processing Systems*. The MIT Press.

Crammer, K., & Singer, Y. (2001). Pranking with ranking. *Proceedings of NIPS 2001*.

Craswell, N., Hawking, D., Wilkinson, R., & Wu, M. (2003). Overview of the TREC 2003 web track. *Proceedings of TREC 2003* (pp. 78–92).

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proceedings of ICML 1998* (pp. 170–178).

Herbrich, R., Graepel, T., & Obermayer, K. (1999). Support vector learning for ordinal regression. *Proceedings of ICANN 1999* (pp. 97–102).

Hersh, W. R., Buckley, C., Leone, T. J., & Hickam, D. H. (1994). OHSUMED: An interactive retrieval evaluation and new large test collection for research. *Proceedings of SIGIR 1994* (pp. 192–201).

Jarvelin, K., & Kekanainen, J. (2000). IR evaluation methods for retrieving highly relevant documents. *Proceedings of SIGIR 2000* (pp. 41–48).

Joachims, T. (1999). Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, 169–184.

Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proceedings of KDD 2002* (pp. 133–142).

Lebanon, G., & Lafferty, J. (2002). Cranking: Combining rankings using conditional probability models on permutations. *Proceedings of ICML 2002* (pp. 363–370).

Luce, R. D. (1959). *Individual choice behavior*. Wiley.

Nallapati, R. (2004). Discriminative models for information retrieval. *Proceedings of SIGIR 2004* (pp. 64–71).

Plackett, R. L. (1975). The analysis of permutations. *Applied Statistics, 24(2)*, 193–202.

Qin, T., Liu, T.-Y., Lai, W., Zhang, X.-D., Wang, D.-S., , & Li, H. (2007). Ranking with multiple hyperplanes. *Proceedings of SIGIR 2007*.

Shashua, A., & Levin, A. (2002). Taxonomy of large margin principle algorithms for ordinal regression problems. *Proceedings of NIPS 2002*.

Tsai, M.-F., Liu, T.-Y., Qin, T., Chen, H.-H., & Ma, W.-Y. (2007). Frank: A ranking method with fidelity loss. *Proceedings of SIGIR 2007*.