BoltzRank: Learning to Maximize Expected Ranking Gain

Maksims N. Volkovs MVOLKOVS@CS.TORONTO.EDU
Richard S. Zemel ZEMEL@CS.TORONTO.EDU
Department of Computer Science, University of Toronto. Toronto, ON, M5S 3H5, CANADA

Abstract

Ranking a set of retrieved documents according to their relevance to a query is a popular problem in information retrieval. Methods that learn ranking functions are difficult to optimize, as ranking performance is typically judged by metrics that are not smooth. In this paper we propose a new listwise approach to learning to rank. Our method creates a conditional probability distribution over rankings assigned to documents for a given query, which permits gradient ascent optimization of the expected value of some performance measure. The rank probabilities take the form of a Boltzmann distribution, based on an energy function that depends on a scoring function composed of individual and pairwise potentials. Including pairwise potentials is a novel contribution, allowing the model to encode regularities in the relative scores of documents; existing models assign scores at test time based only on individual documents, with no pairwise constraints between documents. Experimental results on the LETOR3.0 data set show that our method out-performs existing learning approaches to ranking.

1. Introduction

Ranking in general, and in particular web document ranking, has received a lot of attention recently primarily due to its direct application in search engines. In a document retrieval domain, the problem of learning to rank can be described as follows. During training, the system is given queries and, for each query, a retrieved set of documents and their relevance levels. The goal is then to construct a ranking function which, when presented with a new query, would accurately rank the corresponding retrieved documents.

Several metrics are used in information retrieval (IR) to evaluate the performance of a ranking function. Two standard metrics are Normalized Discounted Cumulative Gain (NDCG) (Jarvelin et al., 2000) and Mean Average Precision (MAP) (Baeza-Yates at al., 1999). NDCG offers some advantages, e.g., a truncation level parameter that reflects how many documents are shown to the user, that make it especially well-suited to document retrieval. We will thus concentrate on NDCG throughout this paper, but will also demonstrate that MAP or any other IR evaluation metric can also be successfully optimized using our approach.

Most current ranking algorithms that have been proposed to solve this problem can be divided into three main categories: individual, pairwise, and listwise. The individual methods such as PRank (Crammer et al., 2001) do not use any relative information between documents, instead attempting to directly create a scoring function, scores of which are then used to rank the documents. On the other hand, the pairwise methods, including RankNet (Burges et al., 2005), its extension LambdaRank (Burges et al., 2006), and RankBoost (Freund et al., 2003), concentrate on minimizing the relative pairwise misclassification error in document rankings. Finally, listwise methods such as AdaRank (Xu et al., 2007), SoftRank (Taylor et al., 2008), ListNet (Cao et al., 2007) and C-CRF (Quin at al., 2008) use lists of ranked documents as "instances" during training, and learn a ranking model by minimizing some listwise loss function. The approach presented in this paper also falls into the listwise category. We chose to work in the listwise domain because it is the most natural way to model the ranking problem, as it is the only approach that allows direct incorporation of IR evaluation metrics, which are always functions of ranked lists and not individual documents or their pairs.

Existing ranking methods share two main disadvantages that significantly affect their performance.

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

First, current methods are do not directly incorporate NDCG into the learning process. A surrogate function such as pairwise misclassification loss or a bound on NDCG is typically introduced, and optimizing this function leads to an indirect optimization of NDCG. The main problem with this approach is that by optimizing a different function a significant amount of effort of the system can be wasted in areas that have little or no effect on NDCG (Burges, 2006). Second, none of the current methods consider higher-order interactions between documents at test time. Even the pairwise methods, which learn based on relative information in pairs of documents, produce a scoring function that operates only on single documents for test queries. As a result, all the relative information that is used to learn the parameters of the scoring function is either disregarded or converted into a function based on individual documents. Therefore, potentially highly useful information in feature correlations between documents is not fully exploited.

In this paper we develop a new, flexible, ranking model that aims to solve both of the above mentioned problems. We refer to it as BoltzRank. BoltzRank utilizes a scoring function composed of individual and pairwise potentials to define a conditional probability distribution, in the form of a Boltzmann distribution, over all permutations of documents retrieved for a given query. We also formulate our approach based on a general loss function, which allows BoltzRank to directly include any IR performance measure into the learning process.

The rest of the paper is organized as follows. Section 2 provides a detailed description of the problem, notation and IR metrics. Section 3 introduces the BoltzRank method. Experimental results are presented in Section 4, and the final section contains conclusions and future work.

2. General Framework

Here we formalize the problem of learning to rank in the document retrieval domain. At training time we are given a set of n queries $Q = \{q_1, ..., q_n\}$, and for each query q_i we are also given a set of documents $D_i = \{d_{i1}, ..., d_{im_i}\}$, and their associated relevance levels $L_i = \{l_{i1}, ..., l_{im_i}\}$, where m_i denotes the number of retrieved documents for query q_i. Each document d_{ij} is represented as a feature vector in $\Re^p$, and the corresponding relevance level $l_{ij} \in \Re$ (typically an integer) indicates how relevant that document is to the query q_i.

Given query q_i, our model produces a ranking of documents D_i through a scoring function $f(q_i, D_i)$, which outputs a set of scores $S_i = \{s_{i1}, ..., s_{im_i}\}$, $s_{ij} \in \Re$; we denote $R_i = \{r_{i1}, ..., r_{im_i}\}$, $r_{ij} \in \{1, ..., m_i\}$ as a set of ranks given to D_i when documents in D_i are ranked according to S_i. In this representation r_{ij} is the position of the document d_{ij} in the ranked order, where the document with the highest score is assigned a rank of 1 and the document with the lowest score a rank of m_i. The goal of learning then is to create a scoring function f (we omit the arguments q, D to reduce notational clutter) such that, given a set of documents D with relevance levels L retrieved for a new query q, the permutation R resulting from scores S assigned by f has maximal agreement with L.

NDCG and MAP are typically used to evaluate the agreement between the ranking produced by S and the relevance levels. For a given ranking R, and relevance levels L, NDCG is defined as:

$$NDCG(R, L)@T = N_q \sum_{j=1}^{T} \frac{2^{\mathrm{rel}(j)} - 1}{\log(1 + j)} \qquad (1)$$

where $\mathrm{rel}(j)$ is the relevance level of the document with rank j, and N_q is a normalizing constant that ensures that a perfect ordering has NDCG value of 1. The normalizing constant allows an NDCG measure averaged over multiple queries with different numbers of documents to be meaningful. Furthermore, T is a truncation constant and is generally set to a small value to emphasize the utmost importance of getting the top ranked documents correct.

MAP only allows binary (relevant/not relevant) document assignments, and is defined in terms of average precision (AP):

$$AP(R, L) = \frac{\sum_{j=1}^{m} P@j * \mathrm{rel}(j)}{\sum_{j=1}^{m} \mathrm{rel}(j)} \qquad (2)$$

where m is the number of documents; and $P@j$ is the precision at j:

$$P@j = \sum_{i=1}^{j} \frac{\mathrm{rel}(i)}{j} \qquad (3)$$

MAP is then computed by averaging AP over all queries.

Both NDCG and MAP include summations over sorted lists of documents, and therefore are not smooth, and can not be optimized by any direct gradient-based method. In this paper we present an expectation-based method which allows direct optimization of such non-smooth evaluation metrics frequently used in information retrieval. To further emphasize that any IR evaluation metric can be optimized with our approach, we will utilize a general error

Table 1. A Summary Of Notation

Variable	Description
$Q = \{q_1, ..., q_n\}$	input queries
m_i	number of documents for q_i
$D_i = \{d_{i1}, ..., d_{im_i}\}$	documents for q_i
$L_i = \{l_{i1}, ..., l_{im_i}\}$	relevance levels for q_i
$f(q_i, D_i)$	scoring function
$S_i = \{s_{i1}, ..., s_{im_i}\}$	scores given by f to D_i
$R_i = \{r_{i1}, ..., r_{im_i}\}$	ranks obtained by sorting S_i
rel(j)	relevance of document at rank j
$\mathcal{G}(R_i, L_i)$	IR performance measure

function $\mathcal{G}(R, L)$ to represent the target performance measure. Table 1 summarizes all the notation introduced above.

3. Our Method: BoltzRank

In this section we describe in detail the idea behind our approach together with learning and inference methods. To simplify notation, for the remainder of this section, we drop the query index i, and work with general query q and document set $D = \{d_1, ..., d_m\}$.

3.1. Distribution Over Permutations

The main idea that motivates the BoltzRank approach is the observation that if we define a probability distribution over document permutations, and consider the expectation of the target performance measure under this distribution, then it should be possible to propagate the derivatives and update the parameters that govern the scoring function to maximize this expectation. Thus we begin this section by defining a flexible probability distribution over permutations. More formally, given a set of scores $S^{(f)} = \{s_1, ..., s_m\}$ assigned to D by f and corresponding ranking $R = \{r_1, ..., r_m\}$, we define the conditional energy of R given S as an average over unique document pairs in R:

$$E(R|S) = \frac{2}{m * (m-1)} \sum_{r_j > r_k} g_q(r_j - r_k)(s_j - s_k) \quad (4)$$

where g_q is any sign preserving function, e.g.,

$$g_q(x) = \alpha_q x \quad (5)$$

where α_q is a query-dependent positive constant. When $r_j >> r_k$ (k beats j), $E(R|S)$ gets a large negative contribution if $s_j << s_k$ and a large positive one if $s_j >> s_k$. $E(R|S)$ thus represents the lack of compatibility between the relative document orderings given by R and those given by $S^{(f)}$, with more positive energy indicating less compatibility. The scoring function f plays a very important role in this model and is described in detail in the following section.

Using the energy function we can now define the conditional Boltzmann distribution over document permutations by exponentiating and normalizing:

$$P(R|S) = \frac{1}{Z(S)} \exp(-E(R|S)) \quad (6)$$

$$Z(S) = \sum_R \exp(-E(R|S)) \quad (7)$$

Note that we can not compute $P(R|S)$ or $Z(S)$ exactly since both contain sums over exponentially many document permutations. In practice, however, we will show that efficient approximations of these quantities allow inference and learning in the model.

3.2. Properties of the Model

A fundamental problem faced by all the aforementioned methods is that the IR metric depends on ranks, which are non-smooth functions of the scores, that is, they depend on a rank vector obtained by *sorting* the scores, a difficult operation to differentiate. A key idea in BoltzRank is that if we treat the scores produced by the model as random variables, then we can update parameters with respect to the expectation of a rank-dependent objective, utilizing sufficient statistics based on the score distribution. Other methods have taken a similar approach. For example, SoftRank (Taylor et al., 2008) used a ranking form of a binomial distribution to approximate the distribution of ranks obtained by sorting scores drawn from a score distribution. BoltzRank differs from these methods in terms of how it approximates the score distribution.

BoltzRank directly estimates the probability of a particular rank vector R, based on the compatibility of each pairwise relationship in that ranking with the respective scores, combined in a simple manner in the energy $E(R|S)$. For comparison, SoftRank instead focuses on estimating the probability that a given document has a particular rank for a query. It obtains this by first directly estimating π_{ij}, the probability that document i out-ranks document j, for every other document $i \neq j$. It then uses these in two different ways (described below) to compute the probability $p_j(r)$, that document j has rank r.

Here we compare these approaches. We begin with a simple generative model to generate rank vectors from a score vector. A distribution over score vectors is formed by placing independent Gaussians centered on each document's score s_j, with a common standard deviation σ_s. The distribution $P(R|S)$ over rank vectors is obtained by drawing i.i.d. samples from this score

distribution and then sorting to obtain a rank vector. One way of understanding the resulting distribution over rank vectors is in terms of pairwise contests between two documents. Under this generative model, the probability that document i is ranked above document j, is the integral of the difference of two Gaussian random variables, which is a Gaussian centered on the difference in the documents' scores:

$$\pi_{ij} = P(S_i > S_j) = \int_0^\infty \mathcal{N}(s|s_i - s_j, 2\sigma_s^2)ds \quad (8)$$

SoftRank uses this pairwise contest probability in two ways. The first is a recursive computation for the distribution of ranks of each document j, which uses the Rank-Binomial distribution, a ranking form of a binomial distribution, to estimate the probability of the various ranks under which j beats all but r of the other documents. The second is a less expensive normal approximation to the ranks, which approximates the expected rank of document j as $\sum_{i=1,i\neq j}^{m} \pi_{ij}$ and variance equal to $\sum_{i=1,i\neq j}^{m} \pi_{ij}(1-\pi_{ij})$, analogous to the normal approximation to m samples of a true binomial distribution. Both of these methods specify the joint probability of a specific rank vector as the product of each document having that particular rank. In our comparison of the methods, we assume that all the methods have the correct score vector, and that SoftRank has access to the true σ_s in the generator.

We consider a simple case, in which there are three documents to be ranked for a single query; we randomly select the scores to be between 0 and 2, and the standard deviation in the generator is 0.2. Figure 3.2 shows an example of a distribution across rankings for a random score vector, and the distributions estimated by both of SoftRank's approximations, the Normal and the Rank-Binomial, and our Boltzmann distribution.

We then evaluate the quality of these three approximations, by comparing each to the true empirical distribution, using the Kullback-Leibler (KL) divergence, averaged over 500 sampled score vectors, as a distance metric. In Figure 2, we see that BoltzRank provides a more accurate approximation to the true distribution than either of SoftRank's approximations.

We can gain some insight into this result by analyzing the approximation to the pairwise probabilities in BoltzRank. If we use a simple version of the $g_q()$ function, i.e., $g_q(x) = k * \text{sign}(x)$ we can analytically determine π_{ij}:

$$\pi_{ij} = P(S_i > S_j) = \frac{1}{1 + \exp(-k * (s_i - s_j))}$$

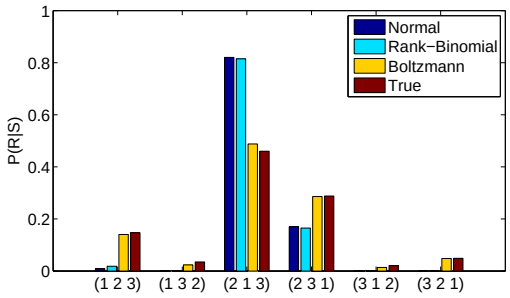

Figure 1. The distributions across rank vectors for a particular score vector. Note that the Normal distribution indeed provides a good approximation to the computationally intensive Rank-Binomial distribution, but neither of these provide as good a fit to the true distribution as that given by our Boltzmann distribution.

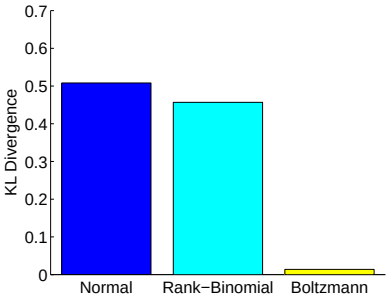

Figure 2. Comparison of the quality of the two approximations in SoftRank (Normal and Rank-Binomial) to the Boltzmann distribution used in BoltzRank, in terms of KL divergence to the true rank distribution, averaged across 500 sampled score vectors.

This provides a reasonable approximation to the true π_{ij}, as with a proper setting for k, this logistic function closely matches the cumulative for the normal distribution specified in Equation 8.

3.3. Scoring Function

The scoring function f consists of two potentials: individual potential ϕ and pairwise potential φ. ϕ operates on single documents and assigns absolute scores to them without considering any relative information. φ takes as input pairs of documents and predicts the relative difference in scores of the two documents in each pair. The final score for any given document d_j is then computed in the following way:

$$f(d_j|q, D) = \phi(d_j) + \sum_{k,k\neq j} \varphi(d_j, d_k) \quad (9)$$

The pairwise potential φ allows f to effectively enforce the learned second order relative constraints during inference which, to the best of our knowledge, has not been explored in existing ranking models. Enforcing these constraints comes at the cost of increased inference time of $O(m)$ for any given document. If φ is dropped from the model then f reduces to the standard scoring function, which allows for extremely efficient inference. In our experiments we demonstrate that adding φ significantly improves the accuracy of the ranker. But even with φ excluded, our model still achieves competitive performance. The system thus offers a trade-off between inference speed and ranking accuracy.

3.4. Learning and Inference

Our model produces a distribution over rank vectors based on the estimated score vector. Several loss functions could be minimized. We could perform a form of maximum likelihood learning, as is frequently performed on Boltzmann models, to match the model's estimate of the rank distribution, $P(R|S)$, to the true rank distribution, $P(R|L)$, as specified by the target relevance levels L.

Here we consider an alternative loss function, in order to incorporate the relevant IR evaluation metric to be used at test time. Our objective is to produce a probability distribution over rankings, as given in Equation 7, that assigns high probability to rankings that maximize the target performance measure $\mathcal{G}$. One direct way to achieve this is by maximizing the expected performance, which leads to a new learning objective:

$$\langle \mathcal{G}|S \rangle_P = \sum_R P(R|S)\mathcal{G}(R, L) \tag{10}$$

where $P(R|S)$ is given in Equation 7. The sum in Equation 10 is intractable. Therefore instead of optimizing $\langle \mathcal{G}|S \rangle_P$ directly, we will optimize its Monte-Carlo estimate:

$$\langle \mathcal{G}|S \rangle_P^{(R_q)} = \sum_{R \in R_q} P^{(R_q)}(R|S)\mathcal{G}(R, L) \tag{11}$$

where R_q is the *rank sample set*, and $P^{(R_q)}$ is the approximate rank probability, obtained by normalizing over this sample:

$$P^{(R_q)}(R|S) = \frac{\exp(-E(R|S))}{\sum_{R' \in R_q} \exp(-E(R'|S))} \tag{12}$$

There are a number of ways to get a representative rank sample set, including various sampling techniques. We exploit our knowledge of the relevance level set L to form an informative set, and we avoid re-sampling for computational reasons, which are of crucial importance in large learning-to-rank data sets. Ideally, we want R_q to contain samples that have a full range of values of the target performance measure, since this would make it most informative during learning. Given that most IR evaluation metrics place the most weight on the top ranked documents, it is not hard to design a procedure that outputs samples with this property for each query. We give an example of such a procedure in Section 4 below. It is important to note here that there is a total of $m!$ possible ranking assignments for a document set of m documents. Thus in order to make learning feasible we can include only a small subset of the rankings in R_q. Our experiments however show that even with a small subset of rankings our model is able to successfully learn and generalize to new queries.

Once the samples are computed for every query, we then define our target gain as summed over all queries:

$$\langle \mathcal{G} \rangle_{TOTAL} = \sum_{i=1}^{n} \langle \mathcal{G}|S_{q_i} \rangle_P^{(R_{q_i})} \tag{13}$$

Note that Equation 13 allows us to incorporate and optimize any IR performance measure. The derivatives of this function with respect to f can easily be computed, and straightforward gradient ascent can then be used to update f. The learning entails using f to get the scores for all documents and back propagating the derivatives of the target gain to learn f. This leads to the algorithm summarized in Algorithm 1. During inference we simply use Equation 9 to compute the scores for new documents and then use these scores to rank the documents.

Optimizing NDCG at training time may not optimize test NDCG (Taylor et al., 2008); one can gain some intuition into this by considering training with $\mathcal{G}$ based on NDCG@1, which only aims to get the top-ranked document correct, and no learning occurs for the rest of the documents. We thus adopt an approach motivated by our probabilistic model: we combine our target performance measure with a form of maximum likelihood. We aim to minimize the KL divergence between the true rank distribution, $P(R|L)$ and the model's predicted distribution $P(R|S)$. This reduces to minimizing the cross entropy between the rank sample distribution under the model and the target rank sample distribution given by the relevance levels:

$$C^{(R_q)} = -\sum_{R \in R_q} P^{(R_q)}(R|L) \log(P^{(R_q)}(R|S)) \tag{14}$$

where $P^{(R_q)}$ is given in Equation 12. The combined

Algorithm 1 BoltzRank Algorithm

Input: $\{(q_1, D_1, L_1), ..., (q_n, D_n, L_n)\}$
Parameters: learning rate η, tolerance ϵ
initialize scoring function: f
for $i = 1$ **to** n **do**
  compute samples: R_{q_i}
end for
repeat
  for $i = 1$ **to** n **do**
    compute scores: $S_{q_i} = f(q_i, D_i)$
    calculate query gain:
      $O_{q_i} = \lambda \langle \mathcal{G} | S_{q_i} \rangle_P^{(R_{q_i})} - (1 - \lambda) C^{(R_{q_i})}$
    compute gradients:$\nabla f = \partial O_{q_i} / \partial f$
    update scoring function: $f = f + \eta \nabla f$
  end for
  calculate total gain: O
until change in total gain is below ϵ
Output: learned scoring function f

learning objective to maximize then becomes:

$$O = \lambda \langle \mathcal{G} \rangle_{TOTAL} - (1 - \lambda) C_{TOTAL} \qquad (15)$$

where C_{TOTAL} sums $C^{(R_q)}$ over all queries. The cross entropy term provides additional learning information by showing the model what the "target" sample probabilities should be. We tune λ via cross-validation.

If only individual potentials are employed, the complexity of our learning algorithm is $O(knm_{max}^2)$, where $k = |R_q|$, is the number of query samples used to approximate the expectation, n is the number of queries, and m_{max} is the maximum number of documents per query. This complexity remains the same if pairwise potential is included, but as was discussed earlier, inference complexity increases from $O(nm_{max})$ to $O(nm_{max}^2)$. Thus if a reasonably small number of samples is used our learning algorithm will not be significantly slower than any pairwise algorithm.

4. Experiments

In this section we describe the experiments that we carried out on LETOR3.0 OHSUMED and TD2004 (Liu et al. 2007) data sets. We chose these sets because they are publicly available, include several baseline results, and provide evaluation tools to ensure accurate comparison between methods. Note that the recently-released LETOR3.0 extends LETOR2.0 by including significantly more features and providing three additional datasets.

4.1. Data Collections

OHSUMED is a data set of medical publication abstracts. There are 106 queries, and a total of 16,140 query document pairs for which the relevance levels are provided. The relevance judgments have three possible levels: 2 = definitely relevant, 1 = possibly relevant, and 0 = not relevant.

TD2004 is a Topic Distillation data set from TREC2004 web search track. There are 75 queries and each query is associated with 1,000 documents. Each query document pair is given binary relevant/not relevant relevance judgment. The data set is heavily dominated by uninformative irrelevant documents with only 1,116 relevant documents out of the total of 75,000. To speed up the learning we subsampled the training data by randomly removing 80 percent of the irrelevant documents for each query; the validation and test sets were unchanged. Note that subsampling of TREC data sets is a standard procedure used for example, by (Taylor at al., 2008).

Both data sets come with five precomputed folds, with 60/20/20 percent splits for training, validation, and test sets. All the models were trained using the training set, fine tuned using the validation set, and tested on the test set. The results shown for each model are the averages of the results for the five folds.

4.2. Parameter Selection

In all our experiments we chose to use one hidden layer neural networks for ϕ and φ potentials. In both networks we employed scaled tanh activation and output functions. In this representation the input to φ was a concatenation of features of the two documents in each pair. To investigate the usefulness of φ, we trained two versions of BoltzRank. The first, denoted as BoltzRank1, utilized a scoring function consisting of only the ϕ potential which, through cross validation, was selected to have 5 hidden units. The second, denoted as BoltzRank2, combined both ϕ and φ potentials with 3 and 5 hidden units respectively. Again, the number of hidden units was found to be optimal through cross validation.[1]

In order to train both methods, we used the same rank sample set R_q, formed as follows. First, for each query in the training set, we computed a set of 100 document rank permutations. Furthermore, we ensure that each permutation set included a full range of target performance measure values by carefully selecting which permutations to include into the set. Most IR eval-

[1]For both potentials the number of hidden units was chosen from the set {1, 3, 5, 10}.

Table 2. Test NDCG and MAP results, the results for the OHSUMED dat set are in the first half of the table and the results for the TD2004 dat set in the second half.

METHOD	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5	MAP
BOLTZRANK1	55.43	53.03	51.77	**50.26**	48.76	45.22
BOLTZRANK2	**56.81**	**54.08**	**51.83**	50.23	**49.10**	**46.04**
ADARANK.NDCG	53.30	49.22	47.90	46.88	46.73	44.98
ADARANK.MAP	53.88	47.89	46.82	47.21	46.13	44.87
FRANK	53.00	50.08	48.12	46.94	45.88	44.39
LISTNET	53.26	48.10	47.32	45.61	44.32	44.57
BOLTZRANK1	45.33	40.00	37.77	36.16	35.91	22.36
BOLTZRANK2	47.67	**41.33**	**39.02**	**37.57**	**36.35**	**23.90**
ADARANK.NDCG	42.67	38.00	36.88	35.24	35.14	19.36
ADARANK.MAP	41.33	39.33	37.57	36.83	36.02	21.89
FRANK	**49.33**	40.67	38.75	35.81	36.29	23.88
LISTNET	36.00	34.67	35.73	34.69	33.25	22.31

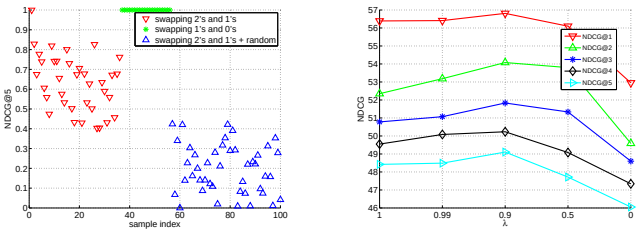

Figure 3. (a). NDCG@5 values for a rank sample set of 100 samples. The first sample is the target score vector, with NDCG value of 1. Every other point is a rank vector obtained by swapping the scores (here, relevance levels of $0, 1$, or 2) and randomly resolving ties. (b). NDCG vs. λ for the OHSUMED data set.

uation metrics place a lot of importance on the top ranked documents. Thus, if for example NDCG is used, then swapping documents with relevance levels 2 and 1 will less significantly affect NDCG than swapping the documents with relevance levels 2 and 0. Using this, our procedure for computing the rank sample set for NDCG was as follows. 35% of the sample were obtained by only swapping subsets of documents with relevance levels 2 and 1, 20% were obtained by swapping subsets of documents with relevance levels 1 and 0, and the rest were obtained by swapping subsets of documents with relevance levels 2 and 0 and randomly permuting the entire set. Once these permutations were computed we then used the relevance levels to sort the documents (randomly resolving ties) and stored the resulting ranks. NDCG@5 values for an example rank sample set resulting from this procedure are shown in Figure 3(a). We see that the first 55% of the sample has very high NDCG, with values mostly above 0.5, and the remainder contains samples with low NDCG values, mostly below 0.4. Such diversity of NDCG values and sample structures should provide a lot of useful information for learning to the system. Note that this procedure can be tailored readily to any set of relevance levels. In general, throughout our experiments we found that as long as the samples were balanced and contained a wide range of target performance measure the actual procedure used to compute them had little effect on the results.

We experimented with various settings for λ in Equation 15 including 0 and 1 and found that $\lambda = 0.9$ performed best. Figure 3(b) shows the behavior of NDCG for different values of λ. From this figure we see that as λ decreases the NDCG increases at first and then drops off. The initial increase in NDCG supports the argument of (Taylor et al., 2008) that NDCG places too much emphasis on top-ranked documents, making learning difficult. On the other hand the significant decrease in NDCG when λ approaches 0 suggests that NDCG contains crucially important ranking information and thus should not be discarded.

We also experimented with different g functions including linear and logistic. We found that as long as g is sign preserving and bounded its exact form had little effect on the results. Thus for all experiments we used a simple linear form: $g_q(x) = 2x/(m_q - 1)$. For each fold we conducted ten random restarts each time retaining the model that gave best validation results. Finally, we experimented with various sample sizes ranging from 20 to 1000, and found no significant improvement by increasing the sample size be-

yond 100.

4.3. Results

Table 2 shows test set NDCG results for the OHSUMED and TD2004 data sets. We compared BoltzRank to four baseline methods: AdaRank.NDCG and AdaRank.MAP (Xu et al., 2007), ListNet (Cao et al., 2007), and FRank (Tsai et al.,2007)[2]; these methods are considered the state-of-the-art in the pairwise and listwise categories described above. From the OHSUMED data set results we see that both BoltzRank methods significantly outperform other methods, especially for small truncation levels. Furthermore, BoltzRank2 consistently outperforms BoltzRank1 on all truncation levels except four. This indicates that higher order document interactions do contain helpful information for ranking. The results for the TD2004 data set further support this conclusion, as BoltzRank1 performs well while BoltzRank2 beats all the baseline methods for all truncation levels except for one.

Experimental results using MAP as the performance metric are also shown in Table 2. We see that the results are similar to those of NDCG, as BoltzRank2 outperforms the other methods on both data sets, and BoltzRank1 is only outperformed by one baseline method on one of the data sets.

5. Conclusion

In this paper, we have proposed a new energy-based, listwise approach to learning to rank. In this approach potentials that depend on individual documents and pairs of documents are combined to define a probability distribution over possible rank assignments to a set of documents retrieved for a given query. Pairwise potentials allow us to explore document interactions beyond individual scoring functions; these have not been previously explored at test time in this domain. We are able to optimize non-continuous listwise loss functions, based on the expectation of the objective under the probability distribution given by our model.

Future work includes studying the relationship between the rank samples we use to approximate the expectation, and the local optimum of the target performance measures that the algorithm finds. An important issue is how well our approximated ranking distribution will scale if there are more relevance levels. Finally, we are exploring applications of this approach to ranking in collaborative filtering.

[2]We omit SoftRank because it is not listed in LETOR's baselines, making standardized comparison impossible.

References

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

C. Burges (2006). Ranking As Function Approximation, *In Algorithms for Approximation*, Springer.

C. Burges, R. Rango and Q. V. Le (2006). Learning to rank with nonsmooth cost functions. *Proceedings of the Neural Information Processing Systems*, 395-402.

C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender (2005). Learning to rank using gradient descent. *Proceedings of the International Conference on Machine Learning*, 89-96.

Z. Cao, T. Qin, T. Y. Liu, M. F. Tsai and H. Li (2007). Learning to rank: From pairwise approach to listwise approach. *Proceedings of the International Conference on Machine Learning*, 129-136.

K. Crammer and Y. Singer (2001). Pranking with ranking. *Proceedings of the Neural Information Processing Systems*, 641-647.

K. Jarvelin and J. Kekalainen (2000). IR evaluation methods for retrieving highly relevant documents. *Proceedings of the Special Interest Group on Information Retrieval*, 41-48.

Y. Freund, R. D. Iyer, R. E. Schapire and Y. Singer (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933-969.

T. Qin, T. Y. Liu, X. D. Zhang, D. S. Wang, H. Li (2008). Global Ranking Using Conditional Random Fields. *Proceedings of the Neural Information Processing Systems*, 1281-1288.

Y. T. Liu, T. Qin, J. Xu, Y. W. Xiong and H. Li (2007). Letor: Benchmark dataset for research on learning to rank for information retrieval. *Proceedings of the Special Interest Group on Information Retrieval*.

M. Taylor, J. Guiver, S. Robertson and T. Minka (2008). SoftRank: Optimizing non-smooth rank metrics. *Proceedings of the Web Search and Data Mining*, 77-86.

M. F. Tsai, T. Y. Liu, T. Qin, H. H. Chen, and W. Y. Ma (2007). FRank: A ranking method with delity loss. *Proceedings of the Special Interest Group on Information Retrieval*, 383-390.

J. Xu and H. Li (2007). AdaRank: A boosting algorithm for information retrieval. *Proceedings of the Special Interest Group on Information Retrieval*, 391-398.