Direct Optimization of Ranking Measures for Learning to Rank Models

Ming Tan, Tian Xia, Lily Guo and Shaojun Wang

Kno.e.sis Center
Department of Computer Science and Engineering
Wright State University
{tan.6,xia.7,guo.9,shaojun.wang}@wright.edu

ABSTRACT

We present a novel learning algorithm, DirectRank, which directly and exactly optimizes ranking measures without resorting to any upper bounds or approximations. Our approach is essentially an iterative coordinate ascent method. In each iteration, we choose one coordinate and only update the corresponding parameter, with all others remaining fixed. Since the ranking measure is a stepwise function of a single parameter, we propose a novel line search algorithm that can locate the interval with the best ranking measure along this coordinate quite efficiently. In order to stabilize our system in small datasets, we construct a probabilistic framework for document-query pairs to maximize the likelihood of the objective permutation of top-τ documents. This iterative procedure ensures convergence. Furthermore, we integrate regression trees as our weak learners in order to consider the correlation between the different features. Experiments on LETOR datasets and two large datasets, Yahoo challenge data and Microsoft 30K web data, show an improvement over state-of-the-art systems.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation, Performance

Keywords

Learning to rank, supervised learning, direct optimization, ranking measures

1. INTRODUCTION

Learning-to-rank aims to automatically build a ranking model from training data. Training data consists of queries and documents that are matched with human-labeled relevance scores. In testing, the ranking model yields permutations from unseen lists. There are several common ranking measures, including MAP, Precision, MRR and NDCG [21].

Learning-to-rank algorithms can generally be grouped into three categories. The first category is the pointwise approach [13, 16, 32], which assumes that each query-document pair has a numerical or ordinal score. Ranking is formulated as a classification or regression problem, in which the rank value of each document is generally computed independently as an absolute quantity. Methods in this category are fairly similar to conventional machine learning algorithms, and cannot handle pairwise preference and orders.

The second category is the pairwise approach [12, 14, 17, 18, 30], in which the ranked list is decomposed into a set of document pairs. Ranking is treated as a classification problem that can determine which document is better than the other for document pairs. The goal here is to minimize the number of inversions in ranking. For example, RankBoost [14] plugs the exponential loss of document pairs into a framework of Adaboost [31]; FRank [35] defines a new bounded loss function called fidelity; RankNet [5] defines a logistic loss for document pairs and uses cross entropy as the loss function, and RankSVM [17, 18] uses SVM to perform a binary classification on these instances. However, the pairwise approach still ignores the information with respect to partial or total orders of retrieved documents. Some recently proposed algorithms, such as LambdaRank [4] and LambdaMART [6], yield good performance. They tackle the problem by defining a smooth approximation to the gradient of the target cost, instead of searching for a smooth approximation to the target cost itself. McAllester et al. [22] recently proposed a perceptron-like algorithm to directly optimize loss functions. When applied to ranking problems, the method is a pairwise approach that directly optimizes the ranking measure. Bertsimas et al. [2] models the ranking problem by mixed integer programming, which supports many commonly adopted ranking measures. It has shown promising performance on moderate-size data sets, but due to the restriction of current computing ability, this method might be difficult to extend to large data sets.

The third category is the listwise approach. In this approach, the entire ranked list of documents for each query is treated as a training item. Ideally, the listwise methods should directly optimize the ranking measures. However, direct optimization of ranking measures faces a major difficulty: the ranking measures are non-convex, non-differentiable and discontinuous, due to the fact that the ranking measures are determined by the ranked position of documents rather than an explicit value of each document's

KDD'13, August 11-14, 2013, Chicago, Illinois, USA

ranking score function. Previous studies partially solved this problem by using surrogate functions of ranking measures. These surrogate functions are either not directly related to ranking measures [8, 19, 28, 39], or a continuous and differentiable approximation or bounds of ranking measures [9, 11, 20, 26, 34, 37, 40, 41, 42]. Nevertheless, there is still an open question of how to resolve a mismatch between the objective function used in training and the final evaluation criterion used to measure the task performance in testing. The only listwise approach that directly optimizes ranking measure is the coordinate ascent method proposed by Metzler and Croft [23]. However, it uses heuristics to find a good point and fails to locate the optimal point along each coordinate.

Our work is mainly inspired by the minimum error rate training (MERT) algorithm [24] that is widely used in machine translation. We describe a novel algorithm, DirectRank, which trains the ranking model by directly optimizing the same ranking measures as used in the testing phase. Our approach adopts the coordinate ascent method, in which one parameter is chosen for tuning, while others are kept unchanged. We observe that when the selected parameter is posed with a small deviation, the ranking measures would not change unless two documents exchange their ranks. Then, it is possible to speed up the optimization of the chosen parameter by only examining those special points that give rise to a change of ranks of any two documents. We call these special points *jumping points*. Any value for the parameter between two adjacent jumping points corresponds to a constant ranking measure. We therefore propose a novel line search algorithm that could efficiently enumerate all jumping points to obtain the optimal interval. Further, to gain better performance in small datasets, in each iteration we adopt a probabilistic model to help select a point in the optimal interval such that the likelihood of the objective permutation of documents is maximized.

The time complexity of DirectRank is shown to be related to the number of jumping points. By empirical studies, we show that the number of jumping points is usually so limited that DirectRank can efficiently enumerate them, and thus perform very efficiently. As our framework and algorithms can support arbitrary weak learners (or features), we also apply them into a regression tree based model for the nonlinear combination of the features. In each iteration, once a regression tree with respect to the current data distribution is fitted, we call the line search algorithm to assign a suitable weight to maximize the ranking measures. This method proves to be effective in two very large datasets. Last but not least, our algorithms converge into a local coordinatewise optimum, but we empirically verify that our algorithm often finds better solutions than many other systems that use a convex surrogate loss function.

We mainly focus on optimizing the NDCG measure, which is the most commonly used in learning-to-rank tasks, though arbitrary ranking measures can be straightforwardly plugged into our framework. We compare the performance with several state-of-the-art baselines on small data sets provided by LETOR [27], as well as large datasets, including Yahoo Challenge data [10] and Microsoft 30K web data[1].

[1]The Microsoft learning to rank dataset is available at http://research.microsoft.com/en-us/projects/mslr/

2. DIRECTRANK: AN EXACT AND DIRECT OPTIMIZATION METHOD

2.1 General Framework

Suppose that a set of training queries $\mathrm{Q}_s = \{q_1, q_2, \cdots q_n\}$ is given, and a set of documents $\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,m(q_i)}\}$ is retrieved for each q_i. Let $m(q_i)$ denote the number of retrieved documents. We define $\mathbf{y_i} = \{y_{i1}, y_{i2}, \cdots, y_{i,m(q_i)}\}$, where $y_{ij} \in \{r_1, r_2, \cdots, r_l\}$, which represents the relevance judgment. We define the order $r_l \succ r_{l-1} \succ \cdots \succ r_2 \succ r_1$, where $\succ$ means the preference relationship. A T-dimensional feature vector $\mathbf{h}(d_{ij}|q_i) = (h_1(d_{ij}|q_i), \cdots, h_T(d_{ij}|q_i))$, is created for each query-document pair.

The objective of ranking is to construct a ranking function f such that for each query, the retrieved documents can be assigned ranking scores using the function and then be ranked according to the scores. The learning process turns out to be that of optimizing the ranking measure which represents the agreement between the permutation by relevance judgments and the ranking yielded by a ranking function. We first define the ranking function using a linear model,

$$f(\mathbf{h}(d_{ij}|q_i)) = \underline{\alpha} \cdot \mathbf{h}(d_{ij}|q_i))$$

where the weight vector $\underline{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_T)$ is the model parameter. DirectRank can also be applied to non-linear ranking models, introduced in Section 2.4.

In this paper, we use NDCG as the ranking measure. Define $\mathcal{G}(d_{ij}, y_{ij}, q_i, f)$ for each query-document pair and their normalized sum $\mathcal{N}(f, \mathbf{y_i}, q_i)$ for each query q_i,

$$\mathcal{G}(d_{ij}, y_{ij}, q_i, f) = \frac{2^{y_{ij}} - 1}{\log_2(1 + \mathrm{pos}(\pi_i, \mathbf{h}(d_{ij}|q_i), f))}$$

$$\mathcal{N}(f, \mathbf{y_i}, q_i) = \frac{1}{Z_i} \sum_{j=1}^{m(q_i)} \mathcal{G}(d_{ij}, y_{ij}, q_i, f) \quad (1)$$

where Z_i is a normalization factor to guarantee $\mathcal{N}(f, \mathbf{y_i}, q_i) \in [0, 1]$, and $\mathrm{pos}(\pi_i, \mathbf{h}(d_{ij}|q_i), f)$ denotes the position of d_{ij} in the permutation π_i. Given a training query set Q_s, NDCG is denoted by $\hat{L}_{NDCG}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(f, \mathbf{y_i}, q_i)$. Usually, NDCG is trimmed at a certain ranking level τ, so we can change $m(q_i)$ in Equation (1) to a constant value τ. Our goal is to search for a model parameter vector $\underline{\alpha}^\star$ that achieves maximum $\hat{L}_{NDCG}(f)$, i.e.,

$$\underline{\alpha}^\star = \arg\max_{\underline{\alpha}} \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{Z_i} \sum_{d_{ij} \in TOP_\tau(q_i)} \mathcal{G}(d_{ij}, y_{ij}, q_i, f) \right)$$

$$TOP_\tau(q_i) = \arg\max_{top_\tau} f(\mathbf{h}(d_{ij}|q_i))$$

where $TOP\tau(q_i)$ is the top τ documents with respect to the ranking function f. The objective function above is difficult to optimize, as it is non-convex, non-differentiable and discontinuous with respect to $\underline{\alpha}$, thus we cannot directly use gradient ascent algorithms to optimize.

We resort to an iterative coordinate ascent method. For each iteration, there will be only one coordinate parameter updated, denoted as α_k, while others stay unchanged. The rationale of this idea is that the ranking function is written as a one-dimensional linear function,

$$f(\mathbf{h}(d_{ij}|q_i)) = \alpha_k \cdot h_k(d_{ij}|q_i) + \sum_{t \neq k}^{T} \alpha_t h_t(d_{ij}|q_i)$$

Algorithm 1 Coordinate ascent algorithm for direct optimization

Require: $Q_s = \{q_i\}_{i=1}^n$
1: **for** Repeatedly choose a parameter α_k **do**
2: **for** $q_i \in Q_s$ **do**
3: Call Algorithm 2 for q_i
4: // Calculate the NDCG jumping points for q_i
5: **end for**
6: Merge all the jumping points and sort them.
7: Calculate NDCG between the jumping points.
8: **end for**
9: Get the interval $[L_{\alpha_k}, R_{\alpha_k}]$ that maximizes NDCG.
10: Pick the coordinates and corresponding intervals that lead to the largest NDCG increments.
11: Update the parameter giving the highest likelihood of top-τ documents for an ranking by human labels.
12: Repeat 1-11 until convergence.

Since $h_k(d_{ij}|q_i)$ is constant with respect to α_k, and so is the second term, we can re-write these two quantities as a_{ij} and b_{ij}, and convert the equation above to,

$$f(\mathbf{h}(d_{ij}|q_i)) = a_{ij} \cdot \alpha_k + b_{ij}$$

Note that for each document $d_{i,j}$ retrieved by each query q_i, there is a linear function of α_k. Given an input of α_k, each document will get an output score from this linear function. The order of such scores actually reflects the order of the documents which further determines the NDCG value.

This is illustrated in Figure 1, where each of the lines represents a scoring function for a document. At any point of α_k, the rank of the linear function output scores is equivalent to the rank of the documents. Note that a slight change of α_k cannot lead to a jump of NDCG value, unless it is big enough to alter the order of the top-τ documents. Such alternation in the order happens only at the point where two lines intersect. We denote the set of such points as *jumping points*. In Figure 1, we have ten lines corresponding to ten documents, which belong to two queries. Intersections $(p_1, p_2, \cdots p_{12})$ are jumping points. Theoretically, we can

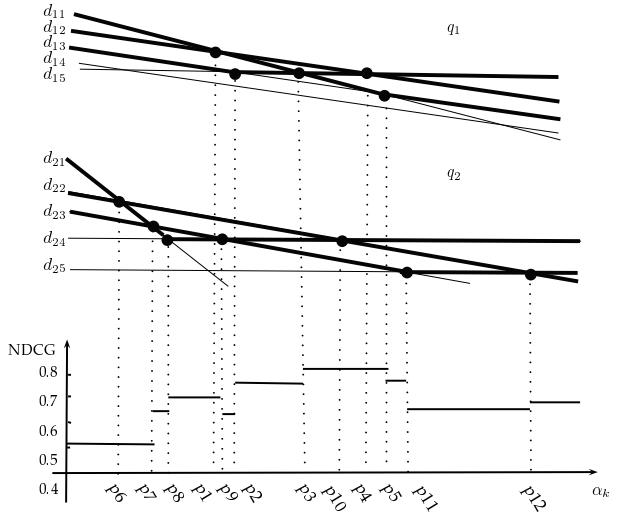

Figure 1: Line search algorithm. The top-τ documents for each of the two queries are bold. Between each two boundaries, the NDCG value is shown. We can see the intervals between p_3 and p_5 achieve the best NDCG.

search all the intersections to acquire all possible snapshots of ranked documents. Because any two non-parallel lines will form an intersection, the total number of intersections then is $m(q_i)^2$. When $m(q_i)$ is large, this effort is time-consuming but unnecessary, because in real-world applications we are merely interested in the rank of the top-τ documents, the NDCG metric is always truncated to a certain level τ, and usually $\tau \leq 10$. As a result, the jumping point size between top-τ documents is quite limited, and increase less than linearly as the document size increases. This will be examined by experiments in section 3.3.

Therefore, we can efficiently find all the jumping points on one coordinate. The coordinate ascent algorithm is described in Algorithm 1. For each coordinate, we exploit such a line search algorithm as following: For each query, we obtain all of its jumping points (Line 2 $\sim$ 5 in Algorithm 1). For instance, in Figure 1, $(p_1, p_2, \cdots, p_5)$ are the jumping points of q_1, while $(p_6, p_7, \cdots, p_{12})$ are the jumping points of q_2. Next, the jumping points of all queries are merged and sorted (Line 6). Then, NDCG can be exactly computed, because the objective is a stepwise function. Between any two adjacent jumping points, the top-τ documents of any query stay unchanged, so does the NDCG value (For example, in Figure 1, the NDCG will not change between p_3 and p_{10}).

In section 2.2, we introduce an efficient line search algorithm, which can quickly enumerate all intervals with optimal NDCG values in each coordinate ascent iteration. In section 2.3, to alleviate the instability on small training datasets, we adopt a probabilistic model to maximize the likelihood of top-τ documents given a set of ranking by human judgment labels. We progress to DirectRank with nonlinear ranking functions, by exploiting regression trees in section 2.4. Finally, in section 2.5, we present a series of theoretical analysis and proofs on the proposed method with respect to time complexity, convergence and consistency.

2.2 Searching for Jumping Points

We again use Figure 1 to illustrate the key ideas in the line search algorithm. Suppose there are only two queries in the training set, q_1 and q_2, each of which consists of five documents that are denoted as $\mathbf{d_1} = \{d_{11}, d_{12}, \cdots, d_{15}\}$ and $\mathbf{d_2} = \{d_{21}, d_{22}, \cdots, d_{25}\}$ respectively. We assign their relevance judgments as $\{1, 2, 0, 1, 2\}$ and $\{0, 0, 2, 1, 1\}$. Suppose we intend to maximize $NDCG@3$, and obtain the $TOP_3(q_i)$ on the direction of the k-th coordinate. This means we apply the coordinate ascent method and search the optimal value of the k-th parameter α_k along the k-th coordinate, while fixing all other parameters.

Interval	$< p_1$	$> p_1$	$> p_2$	$> p_3$	$> p_4$	$> p_5$
Rank 0	d_{11}	$\mathbf{d_{12}}$	d_{12}	d_{12}	$\mathbf{d_{15}}$	d_{15}
1	d_{12}	$\mathbf{d_{11}}$	d_{11}	$\mathbf{d_{15}}$	$\mathbf{d_{12}}$	d_{12}
2	d_{13}	d_{13}	$\mathbf{d_{15}}$	$\mathbf{d_{11}}$	d_{11}	$\mathbf{d_{13}}$
3	d_{14}	d_{14}	d_{14}	d_{14}	d_{14}	d_{14}
4	d_{15}	d_{15}	$\mathbf{d_{13}}$	d_{13}	d_{13}	$\mathbf{d_{11}}$

Table 1: documents of q_1 along α_k. $< p_1$ means the left side of p_1, while $> p_1$ means the right side of p_1, but on the left side of the next point p_2. Bold items denote any changes on the order of ranked documents.

As shown in Figure 1, $(p_1, p_2, \cdots, p_5)$ are the jumping points of q_1, while $(p_6, p_7, \cdots, p_{12})$ are the jumping points of q_2. Table 1 and 2 respectively show the top-τ documents selected between those jumping points, and the changes of rank are noted as bold. Clearly, for a given query, a change

	$< p_6$	$> p_6$	$> p_7$	$> p_8$	$> p_9$	$> p_{10}$	$> p_{11}$	$> p_{12}$
0	d_{21}	$\mathbf{d_{22}}$	d_{22}	d_{22}	d_{22}	$\mathbf{d_{24}}$	d_{24}	d_{24}
1	d_{22}	$\mathbf{d_{21}}$	$\mathbf{d_{23}}$	d_{23}	$\mathbf{d_{24}}$	$\mathbf{d_{22}}$	d_{22}	$\mathbf{d_{25}}$
2	d_{23}	d_{23}	$\mathbf{d_{21}}$	$\mathbf{d_{24}}$	$\mathbf{d_{23}}$	d_{23}	$\mathbf{d_{25}}$	$\mathbf{d_{22}}$
3	d_{24}	d_{24}	d_{24}	$\mathbf{d_{21}}$	d_{21}	d_{21}	d_{21}	d_{21}
4	d_{25}	d_{25}	d_{25}	d_{25}	d_{25}	d_{25}	$\mathbf{d_{23}}$	d_{23}

Table 2: Ranked documents of q_2 along α_k. $< p_6$ means the left side of p_6, while $> p_6$ means the right side of p_6, but to the left side of the next point p_7. Bold items denote any changes on the order of ranked documents.

of document rank order happens when two linear functions of documents intersect at a jumping point. For example, for q_1, its linear functions of d_{11} and d_{12} intersect at p_1, which causes a switch of rank between d_{11} and d_{12} in Table 1.

Algorithm 2 shows how to precisely compute these points. First, we use selection sort to determine top-τ documents with smallest a_{ij}, which is the slope of the lines in Figure 1. This order is actually the rank when α_k takes negative infinity (Line 2 $\sim$ 4 in Algorithm 2). Here we just make sure the top-τ documents are ranked by the scores. Next, we repeat the following procedure in order to find the next intersection that leads to a change of top-τ documents (Line 6 $\sim$ 25). Since the top-τ documents have already been sorted, for each of the top τ-1 documents, we just calculate their intersection with the candidate right below it (Line 8 $\sim$ 13). On the other hand, for the candidate at the rank of τ, all the documents below it have to be scanned, because they might not be sorted, which means any of these documents might be just below the τ-th candidate (Line 14 $\sim$ 19). We choose the minimum one from these intersections, which is the next jumping point (Line 21). We update the rank according to the two intersected documents on this jumping point (Line 23). Standing on the current jumping point, we repeat the procedure above to locate the next jumping point. Such procedure does not terminate until all the jumping points are obtained. For example, as for q_1 in Figure 1, its jumping points will generate from left to right as a sequence: $p_1 \to p_2 \to p_3 \to p_4 \to p_5$. The jumping points of q_1 and q_2 are then merged and sorted, and NDCG between them can be easily calculated. The stepwise NDCG values are shown at the bottom of Figure 1.

Note that when calculating the NDCG values between the jumping points, it is completely unnecessary to re-calculate all the training queries. Instead, since at each jumping point there are only a very small number of candidates (usually two) in one query switching their rank order, we can update NDCG values in an incremental manner. We store the initial NDCG values for each query when $\alpha_k \leftarrow \infty$. Whenever a jumping point is found (Line 21), the NDCG of the corresponding query is updated only by the rank switch of the candidates. For example, as shown in Figure 1 and Table 1, when $p_2 < \alpha_k < p_3$, the NDCG of q_1 is 0.889. At p_3, only d_{11} and d_{15} are switched between the rank 1 and 2, and the NDCG for q_1 can simply be updated as $0.889 + \frac{1}{Z_1}(2^1-1) \times \left[\frac{1}{\log_2(1+2)} - \frac{1}{\log_2(1+1)}\right] + \frac{1}{Z_1}(2^2-1) \times \left[\frac{1}{\log_2(1+1)} - \frac{1}{\log_2(1+2)}\right] = 1$.

There are two ways to choose the coordinate in each iteration. One is cyclic, which repeatedly update every coordinate in the cycle. The other is greedy, which within a single iteration, all coordinates are tried ahead, and then DirectRank updates the parameter with the maximum NDCG.

Algorithm 2 Line search algorithm

Require: q, $\mathbf{d} = \{d_i\}$, and τ
1: $\ell \leftarrow$ {lines for each d_i following Formula 5}
2: $\mathcal{S}_\tau \leftarrow$ {$\ell_i | \ell_i \in \ell$, with top-τ smallest slope} ▷ top-τ documents
3: $\mathcal{S}_b \leftarrow \{\ell_i | \ell - \mathcal{S}_\tau\}$ ▷ the documents out of top-τ
4: $CurrP \leftarrow -\infty$ ▷ Current jumping point
5: $JumpS \leftarrow \{\}$ ▷ All jumping points
6: **repeat**
7: $CandS \leftarrow \{\}$
8: **for** $i = 1$ to $|\mathcal{S}_\tau| - 1$ **do**
9: $p \leftarrow$ Intersection$(\mathcal{S}_\tau^i, \mathcal{S}_\tau^{i+1})$
10: **if** $p > CurrP$ **then**
11: $CandS \leftarrow CandS + \{p\}$
12: **end if**
13: **end for**
14: **for** $i = 1$ to $|\mathcal{S}_b|$ **do**
15: $p \leftarrow$ Intersection$(\mathcal{S}_\tau^{|\mathcal{S}_\tau|}, \mathcal{S}_b^i)$
16: **if** $p > CurrP$ **then**
17: $CandS \leftarrow CandS + \{p\}$
18: **end if**
19: **end for**
20: **if** $CandS \neq \{\}$ **then**
21: $Currp = \min\{\text{CandS}\}$ ▷ get next jumping point
22: $JumpS \leftarrow JumpS + \{Currp\}$
23: Update $\mathcal{S}_\tau$, $\mathcal{S}_b$ ▷ exchange the lines associated with the current jumping point
24: **end if**
25: **until** $CandS = \{\}$
return $JumpS$

We observe that the greedy approach shows no obvious improvement compared to the cyclic way, and is much slower. So we choose the cyclic pattern for experiments.

2.3 Determining the Optimal Value of Model Parameter

Since NDCG is a stepwise function with a single weight α_k, thus there are infinite alternative points in a NDCG-optimal interval. We found that, in small data sets a suitably chosen point improves the stability of our framework, while in big data sets, e.g. Yahoo challenge and Microsoft 30K web, simply taking the middle point of the best interval runs very well. Thus, our system adds the technique in this subsection for small data sets by default, and removes it for big data sets to gain a higher speed.

We exploit a surrogate to choose a point within the interval. DirectRank adopts a probabilistic model in which the only parameter is α_k, and then maximizes the likelihood over an objective permutation. Note that the objective permutation is defined merely based on the human-labeled judgment scores and fixed during the training. Here we use the Plackett-Luce model [25], although any continuous surrogate of NDCG can be used. The Plackett-Luce model treats the ranking procedure as a random selection sequence without replacement. Similar to [8], we define the log probability of the top-τ documents for a specific permutation as,

$$\log P_{\alpha_k}(m(\pi_i); q_i, f) = \sum_{j=1}^{\tau} \log \frac{\exp(f_{\alpha_k}(d_{i,\mu(\pi_i,j)}))}{\sum_{l=j}^{m(q_i)} \exp(f_{\alpha_k}(d_{i,\mu(\pi_i,l)}))}$$

where j and l are the rank indices and $\mu(\pi, j)$ denotes the document of the position j in a permutation π. The log

probability is continuous, differentiable, and concave [3]. We maximize the log likelihood of the objective permutation by the binary search method [1]. This surrogate function optimization is still performed within the best-NDCG interval. Therefore, it is basically a supplementary step for the direct NDCG optimization in section 2.2, in order to distinguish the points with the same NDCG.

2.4 Regression Trees as Weak Learners

In this section, we describe how to integrate regression trees into our framework effectively and conveniently. We follow a *stage-wise* strategy. That is, Algorithm 2 is invoked to tune the optimal weight after each new tree is generated. Here, we use a least-square regression tree, called the MART tree, which is mainly introduced in [15].

MART, one of powerful regression tree based models, aims to construct an ensemble of tree-based weaker learners h_{tree} such that the resulting new scoring function $f(d_i) = \sum_{t=1}^{N} h_{tree}^{t}(d_i) \cdot \alpha_t$ is as close to the objective r_i as possible with the measure of square loss. Here d_i denotes a document, and r_i is the relevance by human judge of a document in learning-to-rank task. The MART method bypasses the difficulties of combinatory optimization of ranking by regression. When weak learners are rich enough, such as with deeper depth of trees, generated models from MART are indeed capable of approaching the regression objective very well, and hence lead to better performance than other linear models.

However, MART was not designed to optimize the objective, and Friedman [15] suggests the tree's weight α_t be tuned towards specific goals. A similar approach also appeared in [43], where another type of regression tree is constructed and combined with a brute search line search. In that work, no significant improvements have been observed. In our framework, we use MART trees as our weak learners, and in order to enhance the stability, we follow the trick in [43] to restrict the new weight α_k in range $[a, b]$, where we empirically set the hyper-parameters between $[0.1, 0.5]$ in our experiments. If the output of Algorithm 2 is beyond the range, we just take the border values.

2.5 Complexity, Consistency and Convergence

We give an empirical analysis on the complexity of the line search algorithm (Alg. 2). Line 8∼13 and Line 14∼19 show that to locate each jumping point, the line search algorithm goes through all the documents once. For a query with m retrieved documents, the proposed algorithm enumerates all jumping points along one coordinate in $O(m \cdot v)$, where v is the number of jumping points. v is related to the data distribution, which is hard to be denoted as a function of m. But we can study the empirical relation between them in Table 12. In LETOR, when m equals 100, v/m is around 0.5; when m increases to 1000, this ratio decreases to around 0.062. The runtime of DirectRank is in Table 13.

We compare our algorithm to the exhaustive line search. For simplicity, we just consider one query. The latter enumerates all the intersections and sorts them, then computes ranking measures from left to right incrementally. Its time complexity is $O(m^2 + m^2 \log(m^2) + m^2) = O(m^2 \log(m^2))$. It is easy to see that the exhaustive algorithm runs faster than DirectRank when v is approximately $O(2m \log m)$.

According to [38], the principle of empirical risk minimization is consistent if it provides a sequence of loss functions for which both expected risk and empirical risk converge to the minimal possible value of the expected risk. Apparently directly optimizing ranking performance measures such as NDCG is an example of empirical risk minimization. To make our presentation easier, let $\bar{\mathcal{X}}$ be the space of the feature vectors in which the documents are represented and are typically derived from the query-document pairs. Let $\bar{\mathcal{Y}}$ be the space of the relevance scores each document receives. Thus for any query q that sampled from the query space $\mathcal{Q}$, we have a list $\mathbf{X} = (\mathbf{X_1}, \cdots, \mathbf{X_m}) \in \mathcal{X} := \bar{\mathcal{X}}$ of document feature vectors, and a corresponding list $\mathbf{Y} = (\mathbf{Y_1}, \cdots, \mathbf{Y_m}) \in \mathcal{Y} \in \bar{\mathcal{Y}}$ of document relevance scores. Then the expected NDCG measure can be written as

$$L_{\mathrm{NDCG}}(f) = \int_{\mathcal{Q}} \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathcal{N}(f(\mathbf{X}), \mathbf{Y}, \mathbf{q}) \mathbf{dP}(\mathbf{X}, \mathbf{Y}|\mathbf{q}) \mathbf{dP}(\mathbf{q})$$

Following the standard proof of empirical risk minimization [38], it can be shown that directly optimizing the empirical NDCG measure is consistent as stated below.

Theorem 1. *Denote f_n^* as the ranking function that maximizes the empirical NDCG $\hat{L}_{NDCG}(f)$ over n query-document pairs, and f^* as the ranking function that maximizes the expected NDCG measure $L_{\mathrm{NDCG}}(f)$ for a fixed but unknown distribution over the probability space of query-document pairs. Then $L_{\mathrm{NDCG}}(f_n^*) \to L_{\mathrm{NDCG}}(f^*)$ as $n \to \infty$.*

The empirical NDCG is non-convex, non-differentiable and discontinuous, but we can prove the convergence of the proposed coordinate ascent algorithm as stated below.

Theorem 2. *The proposed coordinate ascent algorithm converges to a set of local coordinatewise maximum solutions.*

The proofs for both theorems are given in [33].

Even though consistent, DirectRank has a hard time finding the ranking function that is the solution of the global maximum of the empirical NDCG measure. The consistency of a family of listwise surrogate functions for the NDCG measure is proved in [29]. Optimizing the empirical concave surrogate objective turns out to be a convex programming problem with considerable computational advantages and the guarantee of the global optimal empirical surrogate objective (but not the optimal empirical NDCG measure), and for which learned ranking functions remain consistent. However, in practice, the size of the training dataset is always limited and finite, the ranking function learned by optimizing listwise surrogate functions do not correspond to even a local coordinatewise maximum solution of empirical NDCG measure, and the ranking function learned by directly optimizing the empirical NDCG measure does correspond to a local coordinatewise maximum solution.

The experiments we conducted below show that DirectRank not only always reaches a higher NDCG measure on training data than other baselines whose surrogate objectives are concave, but also gives a higher NDCG measure on test data. This shows that the local optimal problem of directly optimizing ranking performance measures is less serious than the mismatch between the training objective and the test objective introduced by a surrogate objective.

3. EXPERIMENTAL RESULTS

We study the performance of the proposed algorithm in both small and large datasets. We first evaluate DirectRank with a linear ranking function. We try to clarify two issues: (1) DirectRank often reaches higher ranking measures

Algorithm	@1	@3	@5	@8	@10
DirectRank	**33**	**44**	**44**	**42**	**48**
SmoothRank	30	31	35	33	31
ListNet	27	29	28	33	30
AdaRank-MAP	25	22	23	23	24
AdaRank-NDCG	21	14	14	14	13
SVM-MAP	18	20	20	17	16
RankBoost	15	17	15	19	18
RankSVM	14	18	17	15	16

Table 3: Winning numbers of NDCG on LETOR 3.0

on training data than other baselines with surrogate objectives. (2) DirectRank proves to be efficient in runtime and stable in handling large datasets. Moreover, we compare the regression-tree version of DirectRank with LambdaMART, the champion of Yahoo Learning-to-rank Challenge.

3.1 Experiments with linear ranking functions

In this subsection, we restrict the ranking function of DirectRank to a linear model and conduct a series of experiments on Microsoft LETOR datasets, which are relatively small, as well as two large datasets, including Yahoo Challenge data and Microsoft 30K web data.

3.1.1 Experiments on Small Datasets

The LETOR datasets website released several benchmark datasets, baselines and evaluation tools. We evaluate DirectRank on all nine datasets in LETOR. For each datasets, five fold partitions for cross validation and their baseline results have been published.

We compare DirectRank with several state-of-the-art learning to rank algorithms in LETOR, as shown in Figure 2. We train our models on NDCG@5 only and evaluate them on NDCG@1∼NDCG@10. We repeat our training procedure 50 times with different initial parameters. Similar to other LETOR baselines, we choose the iteration with the best MAP of validation sets for testing.

Figure 2 provides the the average results of five folds for different learning to rank algorithms in terms of NDCG at each of the first 10 truncation levels. Since these algorithms perform differently on different datasets, to evaluate the overall performance, we use the concept of *winning number* W_i that is introduced by Liu [21] for each algorithm, which counts the number of how many times the algorithm beats other algorithms over all datasets.

$$W_i = \sum_{j=1}^{n} \sum_{k=1}^{m} I(NDCG_i(j) > NDCG_k(j))$$

where n is the number of datasets, k is the indexes of compared algorithms. The larger W_i is, the better the algorithm performs. Due to the different numbers of baseline algorithms in LETOR 3.0 and 4.0, we present winning numbers separately in Table 3 and Table 4. For LETOR 3.0, $n = 7$ and $m = 7$; for LETOR 4.0, $n = 2$ and $m = 5$.

From Table 3, we observe that SmoothRank [11] and ListNet [8] are the best two baseline algorithms. And generally speaking, listwise algorithms outperform pairwise algorithms, except AdaRank-NDCG. On LETOR 3.0 datasets, DirectRank ranks highest on all NDCG levels. For LETOR 4.0 in Table 4, DirectRank wins except on NDCG@1.

From Figure 2, we can see that out of the nine LETOR datasets, there are six datasets for which DirectRank generally gives the best results, including TD2003, NP2004, TD2004, OHSUMED, MQ2007 and MQ2008.

Algorithm	@1	@3	@5	@8	@10
DirectRank	8	**10**	**10**	**10**	**10**
ListNet	3	6	5	7	6
AdaRank-MAP	1	3	3	2	2
AdaRank-NDCG	4	5	5	5	5
RankSVM	4	4	3	2	3
RankBoost	**9**	4	4	4	4

Table 4: Winning numbers of NDCG on LETOR 4.0

	Yahoo	Microsoft
Feature sizes	519	136
Training queries	19k	31k
Total retrieved doc.	473k	2M
Avg. retrieved doc.	about 23	about 72

Table 5: Statistics of Yahoo and Microsoft Data

Compared with other algorithms, DirectRank performs more stably. For example, AdaRank-MAP is the best baseline on HP2004, but it does not fit well in TD2003 and NP2004. SmoothRank generally performs better than other baselines except on TD2003, in which SmoothRank is ranked in the middle. In comparison, DirectRank generally achieves relatively good performance on most of the datasets.

3.1.2 Experiments on Large Datasets

Using Yahoo Challenge Data and Microsoft 30K web data (Table 5), we evaluate DirectRank on accuracy, efficiency and stability. To be consistent, we follow the evaluation approach of Yahoo Challenge, adopting its NDCG formula on both datasets and using NDCG@10 to be the main evaluation criterion. Microsoft dataset releases five cross validation folds, so we report the averaged results.

DirectRank randomizes the initial points 20 times, and picks up the iteration with the best NDCG on the validation dataset. The winner of the Yahoo Challenge is LambdaMART [6], which combines MART [15] and LambdaRank [4]. Since we use a linear function in DirectRank in this subsection, to have a fair comparison, we compare it with LambdaRank, whose ranking function is also linear. Table 6 shows that DirectRank outperforms LambdaRank on NDCG@10. And Tables 6 and 7 report our algorithm performance on Yahoo and Microsoft data. We miss the results of RankBoost in Table 7 because of the 64G memory limitation. We also compare DirectRank with other baselines, such as SmoothGrad [20], AdaRank, simple coordinate ascent [23], and RankBoost [14]. In addition, we compare the proposed algorithm with consistent-RankCosine [29], whose objective is proved to be a surrogate consistent with NDCG. For LambdaRank, we adopt the result on the testing set reported on Yahoo Challenge Workshop. SmoothGrad's code is from BMRM (http://users.cecs.anu.edu.au/%7Echteo/BMRM.html). The results of consistent-RankCosine are generated from the MATLAB program provided by the author of [29]. The rest of the baselines are acquired from RankLib, an open source learning-to-rank package written in Java (http://www.cs.umass.edu/%7Evdang/ranklib.html). We modified the code to adopt the NDCG formula used in Yahoo Challenge. DirectRank yields a general best performance. Consistent-RankCosine has a performance close to DirectRank on Yahoo data, and DirectRank's advantage looks better on Microsoft data. This might reflect the claim in section 2.5, in which even a large scale of queries is insufficient to guarantee

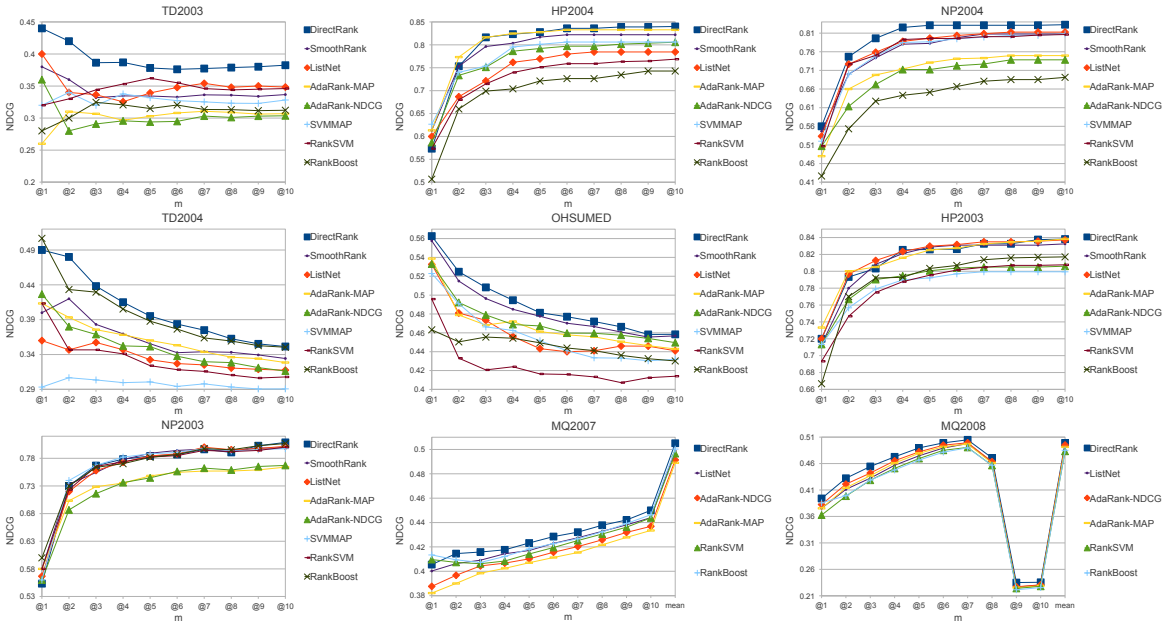

Figure 2: The experimental results in terms of NDCG for Letor 3.0 and 4.0

	DR	LR	SG	AR	CA	RB	CRC
train	**.762**	–	.741	.728	.750	.734	.762
valid	**.757**	–	.738	.723	.744	.730	.755
test	**.760**	.757	.739	.729	.745	.732	.761

Table 6: NDCG@10 on Yahoo dataset for DirectRank(DR), LambdaRank(LR), SmoothGrad (SG), AdaRank (AR), Coordinate Ascent (CA), RankBoost (RB) and consistent-RankCosine (CRC)

Microsoft	DR	SG	AR	CA	CRC
train	**.467**	.460	.368	.462	.451
valid	**.465**	.455	.365	.460	.439
test	**.459**	.450	.365	.457	.437

Table 7: Average NDCG@10 on Microsoft datasets for DirectRank (DR), SmoothGrad (SG), AdaRank (AR), Coordinate Ascent (CA), consistent-RankCosine(CRC)

the consistency for those methods with surrogate objectives, so an algorithm whose optimized objective is the exact ranking measure is more likely to yield a better performance.

As explained in Section 2.1, the NDCG value is usually truncated by the top-τ documents. Therefore, the proposed algorithm performs especially efficiently on optimizing NDCG, where the number of jumping points is very small. Here we also examine the performance of DirectRank on expected reciprocal rank (ERR) and mean average precision (MAP), in which documents are usually not truncated by a certain level. However, as shown in section 2.5, when there are too many jumping points, DirectRank will be slower than a simple brute force search. We have two choices to adapt DirectRank to optimize ERR and MAP. (1) We replace Algorithm 2 with the brute force search. Table 8 indicates a competitive performance compared to LambdaRank and better performance than other baselines on Yahoo Data. Particularly, this work shared the coordinate ascent framework with [23], in which it fails to find the exact optimum in each coordinate iteration. This difference might be indicated

	DR	DR@10	LR	AR	CA	RB
ERR	**.454**	.452	.456	.430	.444	.424
MAP	**.853**	.850	–	.841	.849	.848

Table 8: ERR and MAP on Yahoo Challenge test data. DirectRank(DR), DirectRank which optimizes ERR/MAP@10 (DR@10), LambdaRank(LR), AdaRank (AR), Coordinate Ascent (CA), RankBoost (RB)

	@1	@10
MT	.7084	.7768
LM	.7167	.7791
DirectRank	**.7199**	**.7810**

Table 9: NDCG scores of tree models on Yahoo Challenge, including MART (MT), LambdaMART (LM).

from the performance of DirecRank (DR) and coordinate ascent (CA) in Table 8. (2) In order to speed this up, we can also approximate ERR and MAP with a truncation (eg. 10), for which we observe a performance moderately better than coordinate ascent (CA) on Yahoo data.

3.2 Experiments with Regression Trees

As tree-based models generally outperform linear models, we compare our system with two state-of-the-art systems, MART and LambdaMART on two large datasets. The maximum number of trees is set to 1000. In Yahoo data, the number of leaf nodes is set to 10, and more leaves do not contribute to the final performance significantly with respect to the official measure NDCG@10. On Microsoft 30K web data, we adjust the number of leaf nodes as 10, 30 and 50.

Since our DirectRank is constructed based on MART, the consistent improvements in two large datasets verify our motivation. Wu et al. [43] did a similar trial and received no improvement; we conjecture one of the reasons may be that the regression trees in MART are less prone to overfit data than trees in LambdaMART, because we observe that LambdaMART does boost objective measures quite quickly.

	@1			@10		
	MT	LM	DR	MT	LM	DR
10	.4582	.4602	**.4894**	.4887	.4943	**.4985**
30	.4823	.4830	**.4917**	.4994	.4997	**.5055**
50	.4744	.4883	**.4911**	.5022	.5006	**.5061**

Table 10: NDCG score of tree models on Microsoft 30K web data with varying number of leaf nodes, including MART(MT), LambdaMART (LM), DirectRank (DR).

	DR	SM	AR	LN	RB
TD2003	**.453**	.427	.313	.438	.395
TD2004	**.415**	.366	.337	.354	.407
HP2003	**.916**	.848	.813	.847	.914
MQ2007	**.446**	–	.409	.420	.442

Table 11: Training NDCG@5 of DirectRank (DR), SmoothRank (SR), AdaRank (AR), ListNet (LN) and RankBoost (RB) on different datasets.

Also, DirectRank shows significant superiority to NDCG@1 over Microsoft 30K web data, especially when the number of leaf nodes is quite small, and in other cases (Table 9 [2] and 10) DirectRank still performs slightly better. We find the average number of documents per query is greatly different in the two datasets, about 23 in Yahoo dataset and 72 in Microsoft data. Since MART treats all documents equally, more documents may in some sense have a negative influence on the objective NDCG@1; thus, it would be more likely to acquire improvement by adopting an accurate objective. When the number of leaf nodes increases, the two baselines improve significantly in NDCG@1, while our DirectRank is more stable and effective in performance. Moreover, even for a small number of leaf nodes, DirectRank works very well. Finally, MART in [36] gets a higher performance by using a complete binary tree with different depths, and all tree-based algorithms here are implemented in a fair manner by restricting the maximum number of leaf nodes.

3.3 Analysis of Experimental Results

In this section, we use linear models as an example to analyze DirectRank from the aspects of running efficiency and stability. We first show that as the document size of each query increases, the total number of jumping points does not increase to the same extent. For example, in the two datasets illustrated by Table 12, each query has about 1000 documents. If we fix the truncation level at 5, and increase the document size from 50 to 1000, the average jumping points per query mildly increase, and converge after the document size is larger than 500. This indicates that a large amount of documents will never appear in the top 5 list, and do not affect the value of NDCG@5.

Illustrated by Table 6, 7 and 11, DirectRank often achieves better training NDCG than other baselines using convex surrogates on both small- and large-scale datasets, which suggests a local optimum of direct measure is not necessarily worse than the global optimum of convex surrogates. This point is also suggested by McAllester et al. [22]. However, in the LETOR dataset, in order to prevent overfitting, we choose parameters using cross validation data. In the larger datasets, the algorithms in comparison show a consistent performance on both training and testing data.

[2]LambdaMART does not outperform 0.786, officially released by Yahoo Challenge, maybe because the latter concatenates five training sets subsampled at 70% with original training data, while this paper uses the entire training dataset.

Avg. J. P.	$m = 50$	100	200	500	1000
HP2003	44.20	50.42	53.74	64.60	66.32
HP2004	40.76	52.05	59.18	63.09	65.11

Table 12: Average jumping point (J.P.) number per query with different m. The coordinate is the first feature in the four datasets. The truncation level is 5.

	DR	SG	AR	CA	RB	CRC
Hours	3.3	0.3	11.8	45.3	24.5	5 days

Table 13: Runtime on Yahoo Challenge data for DirectRank (DR), SmoothGrad (SG), AdaRank (AR), Coordinate Ascent (CA), RankBoost (RB) and consistent-RankCosine(CRC)

We implement DirectRank with C++ and run on a single core with a 2.5GHz Opteron. It takes about 10 minutes for a single round, which enumerates all 519 features. As shown in Table 13, given a random starting point, DirectRank converges after around 20 rounds and takes 3.3 hours. SmoothGrad [20] is the fastest, but it does not perform as well as DirectRank in large datasets.

Also, we examine the influence brought by different starting points. We run our program for 100 random initial points on the two large datasets, and obtain the standard deviation of NDCG@10, 0.756±0.0023 on Yahoo data, and 0.457±0.0026 on Microsoft data. It shows that our algorithm might not be that sensitive to the initial point, when the data is large enough.

4. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel line search algorithm DirectRank in the coordinate ascent framework to exactly optimize non-smooth ranking measures. In contrast to other listwise methods, DirectRank can exactly optimize any ranking measures instead of resorting to upper bounds or approximations. DirectRank offers several advantages: efficiency in training, convergence/consistency guarantees, and high accuracy in ranking. Combined with regression-trees, DirectRank proves more powerful. Experiments on small- and large-scale datasets show DirectRank generally outperforms several state-of-the-art baseline systems.

The current algorithm allows only one parameter for optimization per iteration. We are considering combining the Powell algorithm with our line search to realize a fast multi-dimensional optimization. When taking regression trees as weak learners, it is interesting to compare the *stage-wise* and *leveraging* strategies, the latter generating all regression trees first and then leveraging their weights. We are also exploring the possibility that the regression trees are constructed by optimizing ranking measures directly, instead of borrowing from an extra regression model. Finally, since it is now understood that no algorithm that learns a scoring function using a convex surrogate can be consistent with respect to ERR and MAP [7], we will further study the consistency of DirectRank on those measures.

5. ACKNOWLEDGMENTS

This research is partially supported by Air Force Office of Scientific Research under grant FA9550-10-1-0335, the National Science Foundation under grant IIS RI-small 1218863 and a Google research award.

6. REFERENCES

[1] M. Bazaraa, H. Sherali and C. Shetty. *Nonlinear Programming: Theory and Algorithms*, 3rd Edition. Wiley-Interscience, 2006

[2] D. Bertsimas, A. Chang and C. Rudin. Integer Optimization Methods for Supervised Ranking. *Technical Report*, 388-11, 2011

[3] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University, 2004

[4] C. Burges, R. Ragno and Q. Le. Learning to Rank with Nonsmooth Cost Functions. *NIPS*, 19:193-200, 2007

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank Using Gradient Descent. *ICML*, 89-96, 2005

[6] C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. *Microsoft Research Technical Report*, MSR-TR-2010-82.

[7] C. Calauzenes, N. Usunier, P. Gallinari. On the (Non-)existence of Convex, Calibrated Surrogate Losses for Ranking. *NIPS*, 197-205, 2012

[8] Z. Cao, T. Qin, T. Liu, M. Tsai and H. Li. Learning to Rank: From Pairwise to Listwise Approach. *ICML*, 129-136, 2007

[9] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured Learning for Non-smooth Ranking Losses. *KDD*, 88-96, 2008

[10] O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *JMLR-Proceedings Track*, 14:1-24, 2011

[11] O. Chapelle and M. Wu. Gradient Descent Optimization of Smoothed Information Retrieval Metrics. *Information Retrieval*, 13(3):216-235, 2010

[12] W. Cohen, R. Schapire, and Y. Singer. Learning to Order Things. *JAIR*, 10:243-270, 1999

[13] K. Crammer and Y. Singer. Pranking with Ranking. *NIPS*, 14:641-647, 2001

[14] Y. Freund, R. Iyer, R. Schapire and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *JMLR*, 4:933-969, 2003

[15] J. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189-1232, 2001

[16] E. Harrington. Online Ranking/Collaborative Filtering Using the Perceptron Algorithm. *ICML Workshop then Conference*, 20(1):250, 2003

[17] R. Herbrich, T. Graepel, and K. Obermayer. Support Vector Learning for Ordinal Regression. *ICANN*, 97-102, 1999

[18] T. Joachims. Optimizing Search Engines Using Clickthrough Data. *KDD*, 1:7-102, 2002

[19] J. Kuo, P. Cheng, and H. Wang. Learning to Rank from Bayesian Decision Inference. *CIKM*, 827-836, 2009

[20] Q. Le and A. Smola. Direct Optimization of Ranking Measures. *NICTA Tech report*, 2007

[21] T. Liu. *Learning to Rank for Information Retrieval.* Springer, 2011

[22] D. McAllester, T. Hazan, and J. Keshet. Direct Loss Minimization for Structured Prediction. *ICML*, 23:1594-1602, 2010

[23] D. Metzler and W. Croft. Linear Feature-Based Models for Information Retrieval. *Information Retrieval*, 10(3):257-274, 2007

[24] F. Och. Minimum Error Rate Training in Statistical Machine Translation. *ACL*, 311-318, 2003

[25] R. Plackett. The Analysis of Permutations. *Applied Statistics*, 193-202, 1975

[26] T. Qin, T. Liu, and H. Li. A General Approximation Framework for Direct Optimization of Information Retrieval Measures, *Information Retrieval*, 13(4):375-397, 2010

[27] T. Qin, T. Liu, J. Xu, and H. Li. LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Information Retrieval*, 13(4):346-374, 2010

[28] T. Qin, X. Zhang, M. Tsai, D. Wang, T. Liu, and H. Li. Query-level Loss Functions for Information Retrieval. *Information Processing and Management*, 44(2):838-855, 2008

[29] P. Ravikumar, A. Tewari, and E. Yang. On NDCG Consistency of Listwise Ranking Methods. *AISTATS*, 618-626, 2011

[30] C. Rudin. The P-Norm Push: A Simple Convex Ranking Algorithm that Concentrates at the Top of the List. *JMLR*, 10:2233-2271, 2009

[31] R. Schapire and Y. Freund. *Boosting: Foundations and Algorithms.* MIT Press, 2012

[32] A. Shashua and A. Levin. Ranking with Large Margin Principle: Two Approaches. *NIPS*, 937-944, 2003

[33] M. Tan, T. Xia, L. Guo and S. Wang. Direct Optimization of Ranking Measures for Learning to Rank Models. *Technical Report*, 2013

[34] M. Taylor, J. Guiver, S. Robertson and T. Minka. Softrank: Optimizing Non-smooth Rank Metrics. *WSDM*, 77-86, 2008

[35] M. Tsai, T. Liu, Q. Tao, H. Chen and W. Ma. Frank: A Ranking Method with Fidelity Loss. *SIGIR*, 383-390, 2007

[36] S. Tyree, K. Weinberger, K. Agrawal, and J. Paykin. Parallel Boosted Regression Trees for Web Search Ranking. *WWW*, 387-396, 2011

[37] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to Rank by Optimizing NDCG Measure. *NIPS*, 1883-1891, 2009

[38] V. Vapnik. *Statistical Learning Theory.* John Wiley, 1998

[39] F. Xia, T. Liu, J. Wang, W. Zhang, and H. Li. Listwise Approach to Learning to Rank: Theory and Algorithm. *ICML*, 1192-1199, 2008

[40] J. Xu, T. Liu, M. Lu, H. Li, and W. Ma. Directly Optimizing Evaluation Measures in Learning to Rank. *SIGIR*, 107-114, 2008

[41] J. Xu and H. Li. AdaRank: A Boosting Algorithm for Information Retrieval. *SIGIR*, 391-398, 2007

[42] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A Support Vector Method for Optimizing Average Precision. *SIGIR*, 271-278, 2007

[43] Q. Wu, C. Burges, K. Svore, and J. Gao. Adapting Boosting for Information Retrieval Measures. *Information Retrieval*, 13(3):254-270, 2010