Conditional Models for Non-smooth Ranking Loss Functions

Avinava Dubey*[1]   Jinesh Machchhar[†]   Chiranjib Bhattacharyya[‡]   Soumen Chakrabarti[†]
*IBM Research India   [†]IIT Bombay   [‡]IISc Bangalore

Abstract

Learning to rank is an important area at the interface of machine learning, information retrieval and Web search. The central challenge in optimizing various measures of ranking loss is that the objectives tend to be non-convex and discontinuous. To make such functions amenable to gradient based optimization procedures one needs to design clever bounds. In recent years, boosting, neural networks, support vector machines, and many other techniques have been applied. However, there is little work on directly modeling a conditional probability $\Pr(y|x_q)$ *where* y *is a permutation of the documents to be ranked and* x_q *represents their feature vectors with respect to a query* q*. A major reason is that the space of* y *is huge:* $n!$ *if* n *documents must be ranked. We first propose an intuitive and appealing expected loss minimization objective, and give an efficient shortcut to evaluate it despite the huge space of* y*s. Unfortunately, the optimization is non-convex, so we propose a convex approximation. We give a new, efficient Monte Carlo sampling method to compute the objective and gradient of this approximation, which can then be used in a quasi-Newton optimizer like LBFGS. Extensive experiments with the widely-used LETOR dataset show large ranking accuracy improvements beyond recent and competitive algorithms.*

1. Introduction

Search engines use hundreds of complex features from documents and queries to make ranking decisions. Unlike in traditional IR, designing scoring and ranking functions such as TFIDF [15] or BM25 [10] "by hand" is no longer feasible over hundreds of features. Machine learning techniques are increasingly used to design scoring and ranking functions. Learning to rank [13] is now an well-established area of search and machine learning.

1.1. Overview of learning to rank

Algorithms for learning to rank are trained using a set Q of queries. Each query $q \in Q$ has a set of associated documents D_q. Each document has a relevance judgment assigned by a human evaluator. Typically, the relevance of a document is a small integer between 0 and 4, with 0 representing complete irrelevance and 4 representing perfect relevance. For simplicity we will use binary (0/1) relevance in this paper. The relevance of a document i to query q is $z_{qi} \in \{0, 1\}$. We will call relevant documents $D_q^+ \subset D_q$ "good" and irrelevant documents $D_q^- = D_q \setminus D_q^+$ "bad".

From the query q and text of the ith document is constructed a feature vector $x_{qi} \in \mathbb{R}^d$, where d is usually in the dozens to hundreds for commercial search engines. We will use x_{qg} and x_{qb} for good and bad feature vectors.

Virtually all learning to rank algorithms fit a *model* $w \in \mathbb{R}^d$ from the training data. When deployed, a test query t is submitted to the system, along with its document set D_t. The system then assigns a *score* $w^\top x_{ti}$ to each feature vector $x_{ti} \in D_t$, and then the documents are sorted in decreasing order of score.

1.2. Structured learning interpretation

The ranking imposed by the model on the document set may be encoded as a structured label y. We can represent y as a permutation of D_q (breaking score ties arbitrarily). Or, in case of two relevance levels (good and bad), we can represent y as a boolean vector indexed by good-bad pairs. y_{gb} is $+1$ if good document g is ranked better than bad document b, and -1 otherwise. Thus, $y \in \mathcal{Y}_q = \{-1, +1\}^{n_q^+ n_q^-}$, where $n_q^+ = |D_q^+|$ and $n_q^- = |D_q^-|$.

Thus, learning to rank can also be regarded as a classification problem, where the label space $\mathcal{Y}$ is very large. The "correct" label y_q for a query q places each good document ahead of all bad documents. The critical difference between ranking and traditional classification is that we cannot regard all labels other than y_q as equally incorrect. Some rankings are much better than others, even if none of them is perfect. The "defect" of a ranking y wrt the ideal ranking y_q is encoded in a *loss function* [17] $\Delta(y, y_q) \geq 0$, sometimes shorthanded to $\Delta_q(y)$. Naturally, $\Delta_q(y_q) = 0$.

The second piece is a *feature map* [5] $\phi(x_q, y) \in \mathbb{R}^d$ that combines individual document feature vectors x_{qi} into a single vector, using the proposed ranking y. As a simple example, one may add up the vectors at ranks 1 through 10, then subtract all other vectors. (We will see better examples later.) Given a trained model w, the "consistency" of a ranking y is expressed as $w^\top \phi(x_q, y)$: the larger this value, the better the ranking y for query q. Therefore, at test time, the goal is to find $\arg\max_y w^\top \phi(x_q, y)$. This is called *inference*.

1.3. Structured training

During training, we are looking for a w that minimizes $\sum_q \Delta\left(y_q, \arg\max_y w^\top \phi(x_q, y)\right)$ (usually added to some regularization penalty like $\|w\|_2^2$ on the model). The central challenge in learning to rank is that the objective $\sum_q \Delta\left(y_q, \arg\max_y w^\top \phi(x_q, y)\right)$ is highly discontinuous; its gradient is either zero or undefined at any given point w. The vast majority of research on learning to rank is con-

1. Work done while in IIT Bombay.

cerned with approximating the objective with more benign ones that are more tractable for numerical optimization of w. We review a few competitive approaches in recent work.

Some researchers minimize a convex upper bound [17] on the objective above:

$$\min_w \sum_q \max\left\{0, \max_y \Delta_q(y) - w^\top \delta\phi_q(y)\right\}, \quad (1)$$

where $\delta\phi_q(y)$ is shorthand for $\phi(x_q, y_q) - \phi(x_q, y)$. This is a generalization of the well-known hinge loss used in Support Vector Machines [18]. The key is to solve the $\max_y \cdots$ problem without explicitly enumerating through all ys [17]. Solving optimization (1) for commonly used Δ and ϕ functions is nontrivial [20], [11], [5] but test accuracy is often somewhat better than simpler algorithms like RANKSVM [8].

A hinge loss approximation (1) yields upper bounds on the empirical loss [17] but this may not always be the best path to training a good model w. Previous attempts at using structured learning for ranking [5] show that the bound could be loose and may not always yield good models. This motivates the need for alternatives which might be guided by margin-based approaches, and yet better model the empirical risk. In this paper we explore conditional probabilistic models as an alternative, with promising results.

1.4. Probabilistic listwise training

LISTNET [4] is among the best-known listwise training methods. It proposes a very general probability distribution over permutations. To keep training tractable, they restrict to a simpler "top one probability" distribution, and then use cross-entropy for training with a neural network. The optimization is not convex. In contrast, our distribution over permutations is very simple and has a standard log-linear form, which allows polynomial-sized closed forms for some losses and effective sampling techniques for others.

In SOFTRANK [16], the quantity $w^\top x_{qi}$ for each document is regarded as the mean of a normal distribution from which a random score is drawn. It then becomes easy to write down the probability that one score exceeds another, and thereby compute the expected rank of documents as a function of w. This leads to a highly nonconvex optimization for w. Although an intriguing idea, SOFTRANK has not shown consistent gains beyond other approaches.

Very recently, BOLTZRANK [19] proposed an "energy function" $energy(y|\vec{S_q}) = \sum_{i,j}(S_{qi} - S_{qj}) \operatorname{sign}(y_i - y_j)$, where $y_i \in [1, n_q]$ is the rank of document i, and S_{qi} is the score of document i, and tried to reduce a nonconvex function of the energy. Note that the energy function considers all pairs instead of good-bad pairs and seems to be a bad choice: the compatibility between ϕ and Δ is critical for success [5]. BOLTZRANK was evaluated on only two of the seven LETOR [14] data sets; our approach is evaluated on all seven, and we exceed BOLTZRANK accuracy decisively. Another factor may be that, like LISTNET and SOFTRANK, BOLTZRANK leads to non-convex optimizations solved by neural network black-boxes, whereas our best performer is a convex learner.

1.5. Our contributions

Compared to the above approaches, there has been surprisingly little work on extending the paradigm of maximum entropy classification [1] or logistic regression [2] directly to ranking problems.

In Section 2 we propose a parametric conditional probability model $\Pr(y|x; w) \propto \exp(w^\top \phi(x, y))$, and an intuitive minimization of expected ranking loss (equivalently, maximization of a suitably defined expected ranking gain). For a specific but common choice of ϕ and Δ, we give closed form expressions for the objective and gradient that can be efficiently and exactly computed, despite there being an exponential number of rankings y. Unfortunately, the objective is not convex.

In Section 3 we propose a heuristic approximation to the expected gain objective that is convex, but whose computation requires a sum over exponentially many possible rankings y in general. In Section 4 we give a new recipe to replace this exponential sum over ys with a much smaller sum over a sample of ys. We justify why, for typical ranking problems, this approximation is adequate.

In Section 5 we describe experiments with the well-known public ranking data set LETOR, from Microsoft. We compare our new proposals against several competitive systems, including structured max-margin learners and RANKBOOST [6]. The expected gain formulation sometimes beats existing systems, but is plagued by local optima. Interestingly, our convex formulation significantly improves upon the expected gain formulation, and frequently beats existing algorithms significantly.

1.6. Choices of Δ

We will use three definitions of Δ from the literature. Observe that, in all cases, Δ remains unchanged across arbitrary permutations within a relevance level (here, good and bad). Therefore, our feature map ϕ should be designed accordingly.

1.6.1. Pair preference and AUC. For every query q, every good document x_{qg} and every bad document x_{qb}, we want x_{qg} to rank higher than x_{qb}. The number of "satisfied" pairs where this is the case is closely related to the area under the *receiver operating characteristic* (ROC) curve [9].

$$1 - \Delta_{\text{AUC}} = \text{AUC} = \frac{\text{number of satisfied pairs}}{n^+ n^-}$$

A long-standing criticism of pair preference satisfaction is that all violations are not equal [3]; flipping the documents

at ranks 2 and 11 is vastly more serious than flipping #100 and #150. This has led to several global criteria defined on the total order returned by the search engine.

1.6.2. Mean average precision (MAP). For query q, let the ith (counting from zero) relevant or 'good' document be placed at rank r_{qi} (again, counting from zero). Then the precision (fraction of good documents) up to rank r_{qi} is $(1+i)/(1+r_{qi})$. Average these over all good documents for a query:

$$\begin{aligned} &\mathrm{AP}(q) = \textstyle\sum_{i:z_{qi}=1} \frac{1+i}{1+r_{qi}} \\ \text{and define} \quad &\mathrm{MAP} = \textstyle\frac{1}{|Q|}\sum_{q\in Q} \mathrm{AP}(q) \end{aligned} \tag{MAP}$$

The ideal ranking pushes all good documents to the top and ensures a MAP of 1.

1.6.3. Normalized discounted cumulative gain. Of recent interest in Information Retrieval and Machine Learning communities is normalized discounted cumulative gain, abbreviated NDCG. The DCG for a query q and document order is $\mathrm{DCG}(q) = \sum_{0\le i<k} G(q,i)D(i)$ where $G(q,i)$ is the gain or relevance of document i for query q and $D(i)$ is the discount factor given by [7]

$$D(i) = \begin{cases} 1 & 0 \le i \le 1 \\ 1/\log_2(1+i) & 2 \le i < k \\ 0 & k \le i \end{cases} \tag{Discount}$$

Note the cutoff at k. Suppose there are n_q^+ good documents for query q, then the ideal DCG is

$$\mathrm{DCG}^*(q) = \textstyle\sum_{i=0}^{\min\{n_q^+,k\}-1} G(q,i)D(i),$$

pushing all the relevant documents to the top. Now define

$$\mathrm{NDCG}(q) = \frac{\mathrm{DCG}(q)}{\mathrm{DCG}^*(q)} = \frac{\sum_{0\le i<k} z_{qi} D(i)}{\mathrm{DCG}^*(q)} \tag{NDCG}$$

and average $\mathrm{NDCG}(q)$ over queries. $G(q,i)$ is usually defined as $2^{z_{qi}}-1$. Because we focus on $z_{qi}\in\{0,1\}$, we can simply write $G(q,i) = z_{qi}$.

1.7. Feature map ϕ

Recall that for us, $y \in \{-1,+1\}^{n^+n^-}$, where y_{gb} encodes the order between documents g and b. A very commonly used feature map [20] is

$$\phi_{\text{po}}(x_q, y) = \frac{1}{n_q^+ n_q^-} \sum_{g,b} y_{gb}(x_g - x_b). \tag{2}$$

Note that, like Δs above, ϕ is invariant across arbitrary permutations within good or bad documents. (2) is used both with and without the $n_q^+ n_q^-$ scale factor. We omit it, based on cross-validation.

2. Minimizing aggregated expected loss

We begin by proposing a parametric conditional probability

$$\Pr(y|x_q; w) = \frac{\exp(w^\top \phi(x_q, y))}{Z_q} \tag{3}$$

where $Z_q = \sum_{y'} \exp(w^\top \phi(x, y'))$. Note that computing Z_q is not easy for an arbitrary ϕ as it involves summing over exponential number of terms.

Two-class logistic regression [2] with 0/1 loss seeks $\arg\max_w \prod_q \Pr(y_q|x_q; w) = \arg\max_w \sum_q \log\Pr(y_q|x_q;w)$. One can also put a homoscedastic normal prior of the form $w^\top w/C$ over w, inducing regularization in the solution. (C is set by cross validation.) However it is not clear how to extend this setup to general loss functions.

A reasonable quantity to minimize wrt w is the *aggregate expected loss* which is essentially the sum of expected loss per query,

$$\sum_q \sum_y \Pr(y|x_q; w)\Delta(y_q, y) \tag{4}$$

or the log of expected loss

$$\sum_q \log\left(\sum_y \Pr(y|x_q; w) f(\Delta(y_q, y))\right) \tag{5}$$

where f is a monotonic increasing function. It is interesting to note that if we use $f(x) = e^x$ one can show that (4) is a lower bound to (5). Minimizing either of them can yield similar ranking measures.

2.1. Switching from loss to gain

Typically, $n_q^+ \ll n_q^-$. Therefore, most rankings y have $\Delta_q(y) \approx 1$. Even if each of them have small probability, their collective probability may be considerable. Suppose we initialize $w = \vec{0}$, then the initial objective for each query is close to 1. Ranking optimizations tend to be ill-conditioned [5], so, even as w is progressively optimized, the objective (per query) will likely drop from 1 by a very small quantity.

The solution is to write $\Delta_q(y) = 1 - G_q(y)$, where $G(\cdot)$ is a *gain* function. E.g., we can directly use MAP, AUC, or NDCG instead of the corresponding losses. We modify objective (5) to

$$\textbf{ExpGain:} \quad \max_w \sum_q \log\left(\sum_y \Pr(y|x_q; w) G_q(y)\right). \tag{6}$$

(We also have a standard $\|w\|_2^2/C$ regularization term where C is tuned by cross validation.) Now, $G_q(y) \approx 0$ for most ys, and training w lifts terms in the innermost sum from zero, thus increasing numerical sensitivity. We introduce some

notation:

$$S_{qi} = w^\top x_{qi}$$
$$h^q_{gb}(y_{gb}) = \exp\left(\frac{y_{gb}(S_{qg} - S_{qb})}{n_q^+ n_q^-}\right)$$
$$f^q_{gb}(y)[k] = \frac{y_{gb}(x_{qg}[k] - x_{qb}[k])}{n_q^+ n_q^-}$$

With this notation, the objective can be written as

$$\sum_q \underbrace{\log \underbrace{\left[\sum_y G(y) \prod_{g,b} h^q_{gb}(y_{gb})\right]}_{A_q} - \log Z_q}_{L_q} \tag{7}$$

Another benefit of using log-gain is that the $\exp(\cdots n^+ n^- \cdots)$ in the expression above does not create numerical difficulty.

With A_q and L_q defined as in (7), the gradient for a particular query q and the kth dimension is

$$\frac{dL_q}{dw_k} = \frac{1}{A_q} \sum_y G(y) \exp(w^T \phi(x,y)).\phi(x,y)[k]$$
$$- (1/Z_q)(dZ_q/dw_k), \quad \text{where} \tag{8}$$
$$Z_q = \textstyle\sum_y \prod_{g,b} h^q_{gb}(y_{gb})$$
$$\frac{dZ_q}{dw_k} = \textstyle\sum_y \exp(w^T \phi(x_q, y))\phi(x_q, y)[k]$$

In the end it is easy to see that the gradient w.r.t w_k, as defined in (8), is essentially the expectation $-E_Y(\Delta(Y, y_q)\phi(x_q, Y)[k])$ where Y is a random variable defined by (3). We would like to leverage this observations later for efficient algorithms.

2.2. Nonconvexity

In standard logistic regression with two classes and 0/1 loss $\Delta_q(y) = [\![y \neq y_q]\!]$, it is well known [2] that the objective $\max_w \prod_q \Pr(y_q|x_q; w)$ is log-concave, and therefore, there is no danger of getting stuck in a local optimum. Unfortunately, the generalizations $\min_w \sum_q \sum_y \Pr(y|x_q; w)\Delta_q(y)$ or $\min_w \sum_q \log\left(\sum_y \Pr(y|x_q; w)\Delta_q(y)\right)$ may not be convex optimizations for arbitrary probability distributions. In our experiments, we implemented a gradient method with multiple restarts to guard against this problem, and diagnostic tests suggest that these restarting strategies were adequate.

2.3. Polynomial form for AUC

Perhaps more serious than demanding a nonconvex optimization is the problem that both (7) and (8) involve a sum over all y, which cannot be implemented as-is.

For the specific case where Δ_{AUC} is used with ϕ_{po}, we provide algebraic identities that enable us to rewrite (7) and (8) in a way that lets them be evaluated in polynomial time, by exploiting the special structure of y, ϕ, and Δ. This may be of independent interest.

Let $\#1_y = \sum_{g,b} [\![y_{gb} = 1]\!]$. (7) can be written as

$$\sum_q \left\{ \log\left[\sum_y \#1_y \left(\prod_{gb} h^q_{gb}(y_{gb})\right)\right] - \log Z_q \right\}$$

Using the identities in Figure 1, the objective $\sum_q \log(A_q - \log Z_q)$ can be written as

$$\sum_q \log\left(\sum_{g,b} h^q_{gb}(-1) \prod_{\langle i,j\rangle \neq \langle g,b\rangle} \left(h^q_{ij}(-1) + h^q_{ij}(1)\right)\right)$$

Note that there is no sum over y in the above expression. Z_q is completely factored and easily expressed as

$$Z_q = \prod_{g,b} \big(h_{gb}(-1) + h_{gb}(1)\big).$$

The identities in Figure 1 also allows us to manipulate $\nabla_w Z_q$ into a form that can be evaluated in polynomial time (q dropped for clarity):

$$\sum_{g,b} \Big(f_{gb}(-1)h_{gb}(-1) + f_{gb}(1)h_{gb}(1)\Big) \prod_{\langle i,j\rangle \neq \langle g,b\rangle} \big(h_{ij}(-1) + h_{ij}(1)\big)$$

We omit the tedious details. The above setup works by carefully exploiting the similarity of the expression for ϕ_{po} and Δ_{AUC}. But we also need to give a general solution for Δ_{NDCG} and Δ_{MAP}, where a simple and efficient closed form is not possible. As we shall see in Section 5, the expected gain formulation, with one of AUC, NDCG or MAP gains, beats prior art in a large majority of cases. In the next section we give our final formulation which is even better.

3. Probabilistic structured ranking

As described in Section 1.2, structured learning can be a powerful tool for listwise learning to rank [20], [11], [5]. In this section, we upper bound the aggregated expected loss expression developed in Section 2 with an expression that is closely related to loss expressions in structured learning. Interestingly, using our upper bound in a quasi-Newton optimizer beats all other baselines, including structured learning and BoltzRank, in a large majority of cases. (Note: We keep all our objectives well-posed by adding a $\|w\|_2^2/C$ term and tune C by cross validation as usual. Here we show only the training loss part for simplicity.)

Define $\delta\phi_q(y) = \phi(x_q, y_q) - \phi(x_q, y)$ and consider the distribution

$$\Pr(y|x_q, y_q; w) = \frac{\exp(-w^\top \delta\phi_q(y))}{\sum_{y'} \exp(-w^\top \delta\phi_q(y'))} \tag{9}$$

$$\sum_{a\pm 1^n} \prod_{1\le i\le n} h_i(a_i) = \prod_{1\le i\le n} (h_i(-1)+h_i(1))$$

$$\frac{1}{2^{n-2}} \sum_{a\pm 1^n} \left[\sum_{1\le i\le n} f_i(a_i)\right]\left[\sum_{1\le j\le n} h_j(a_j)\right] = 2\sum_{1\le i\le n} (f_i(-1)h_i(-1)+f_i(1)h_i(1)) + \sum_{i\ne j}(f_i(-1)+f_i(1))(h_j(-1)+h_j(1))$$

$$\sum_{a\in\pm 1^n} \left[\sum_{1\le i\le n} f_i(a_i)\right]\left[\prod_{1\le j\le n} h_j(a_j)\right] = \sum_{1\le i\le n} (f_i(-1)h_i(-1)+f_i(1)h_i(1)) \prod_{i\ne j}(h_j(-1)+h_j(1))$$

$$\begin{aligned}&\sum_{a\in\pm 1^n} \left(\sum_{1\le i\le n} f_i(a_i)\right)\left(\sum_{1\le j\le n} g_j(a_j)\right)\left(\prod_{1\le k\le n} h_k(a_k)\right)\\ &= \sum_{\substack{1\le i,j\le n\\ i\ne j}} \Big(f_i(-1)h_i(-1)+f_i(1)h_i(1)\Big)\Big(g_j(-1)h_j(-1)+g_j(1)h_j(1)\Big) \prod_{k\ne i, k\ne j}(h_k(-1)+h_k(1))\\ &\quad+ \sum_{1\le i\le n}\Big(f_i(-1)g_i(-1)h_i(-1)+f_i(1)g_i(1)h_i(1)\Big)\prod_{j\ne i}\Big(h_j(-1)+h_j(1)\Big)\end{aligned}$$

Figure 1. Identities used to evaluate expected gain and its gradient efficiently.

This distribution is actually the same as (3), but restating it like this helps in relating it to structured learning scenario. For the sake of brevity, we overload $Z_q(w)$ as the denominator in (9) above. One can find the maximum likelihood estimate the model w by minimizing the following objective:

$$\textbf{MLE:}\quad L_1(w) = \sum_q L_{1q}(w) = \sum_q \log Z_q(w) \qquad (10)$$

This is a reasonable objective as it leads to positive values of $w^\top \delta\phi_q(y')$ at optimum, which is the case in structured learning. However, it does not exploit information from Δ.

To this end, we can design an expected loss per query, along the lines of BoltzRank [19] and Section 2, i.e., $E_{Y\sim(3)}(\Delta_q(Y))$, where the subscript means that distribution (3) is used to compute the expectation. Instead, we consider the following alternative: $\min_w L_2(w)$, where

$$L_2(w) = \sum_q L_{2q}(w) = \sum_q E_{Y\sim(9)}(\Delta_q(Y)), \qquad (11)$$

now using (9) as the distribution of Y. This will result in direct optimization of the ranking measures. We might further boost the performance by combining (10) and (11):

$$\min_w L_3(w) = \min_w L_1(w) + L_2(w), \qquad (12)$$

so as to reinforce the property that ys having low loss should have large $w^\top\phi_q(y)$. The problem is that L_3 will in general be non-convex because of L_2.

Now consider the final modification to (9) and (11) to combine these two objectives:

$$\Pr(y|x_q, y_q) = \frac{\exp(-w^\top\delta\phi_q(y)+\Delta_q(y))}{\sum_{y'}\exp(-w^\top\delta\phi_q(y')+\Delta_q(y'))} \qquad (13)$$

For given training data, the MLE for w under distribution (13) will be equivalent to $\min_w L(w)$ where

ConvexLoss:

$$L(w) = \sum_q \log\sum_{y'} \exp(-w^\top\delta\phi_q(y')+\Delta_q(y')) \qquad (14)$$

Next, using distribution (9), write

$$L(w) = \sum_q \log Z_q \sum_{y'} \Pr_{Y\sim(9)}(y'|x_q,y_q)\exp(\Delta_q(y'))$$

Finally, using Jensen's Inequality $E(e^X) \ge e^{E(X)}$, we obtain

$$L(w) \ge L_1(w) + L_2(w).$$

Summarizing, we have arrived at a convex upper bound to $L_1(w)+L_2(w)$ that, unlike L_1, takes Δ into account, but does not have the non-convexity disadvantage of L_2.

The objective functions are not easy to optimize, as they involve summation of a large number of terms. To circumvent this problem, we devise sampling schemes which give gradient estimates that can be used with quasi-Newton procedures.

4. Sampling the space of rankings

At this point our main remaining problem is to evaluate expressions of the form $E_Y(a_Y) = \sum_y a_y \Pr(y|x_k)$, where the sum ranges over all rankings. Given our encoding of y there are $2^{n^+n^-}$ values of y, although not all of them correspond to valid total orders. Approximating the above sum is needed to compute the value of the objectives and gradients designed in Sections 2 and 3. We do this by drawing some m samples $\hat{Y}$ from the subset of $\{\pm 1\}^{n^+n^-}$

that is consistent with total orders, and computing the empirical average $(1/m)\sum_{y\in\hat{Y}} a_y$.

In theory, one should use $\Pr(y|x_k)$ to draw the samples. Given the enormous space of y (and therefore numerically infinitesimal probability for any y) doing this directly is infeasible. The standard way to explore such large spaces is Markov Chain Monte Carlo (MCMC) sampling [2]:

1: **while** not enough samples **do**
2: (re)start a random walk at a well-chosen state y^0
3: **for** some number of steps $t = 1, 2, \ldots$ **do**
4: transition from y^{t-1} to y^t
5: make an accept/reject decision on y^t
6: collect some subset of $y^0, y^1, y^2, \ldots$ as samples

Getting the underlined details right is, in most applications, a practised art.

In standard MCMC sampling, because $\Pr(y^t|x_q; w) \propto \exp(w^\top \phi(x_q, y^t))$, the acceptance probability would be a function of w. This is a big problem for quasi-Newton optimizers like LBFGS [12], because as we draw samples every Newton iteration, we would be giving LBFGS numerically inconsistent views of the objective and gradient, and line search would fail. (This was observed by us in practice; to our knowledge this issue has never been reported.)

Therefore, we cannot use a standard MCMC recipe. Instead, we draw the sample $\hat{Y}$ just once before we begin optimizing w, but we draw $\hat{Y}$ using the following strategy:

- Choose restart states to span a variety of Δs.
- In each walk, make local changes in y so as to stay near to the restart Δ.

1: **Input:** $y_{in} \in \{\pm 1\}^{n^+ n^-}$
2: **Output:** $y_{out} \in \{\pm 1\}^{n^+ n^-}$
3: Draw a random number idx between 0 and $n^+ n^-$.
4: idx gives a position indexed by gb in y_{in}.
5: Let the number of bad documents that g beats in y_{in} be n_g and the number of good documents that b beats be n_b.
6: **if** $(y_{in}[idx] = +1)$ **then**
7: $\theta = \dfrac{n^- - n_g + n_b + 1}{2 + n^+ + n^-}$
8: **else**
9: $\theta = \dfrac{n^+ + n_g - n_b + 1}{2 + n^+ + n^-}$
10: With probability θ, flip the bit at $y_{in}[idx]$ to get candidate next state $y_?$.
11: Check $y_?$ to see if it is a valid ranking, reject if not.
12: Repeat above steps until there is a flip and some $y_?$ is accepted.
13: **return** $y_?$ as y_{out}.

Figure 2. Swap method used for sampling.

4.1. Restart states

Both a_y and $\Pr(y|x_k; w)$ are skewed, and this must be taken into account while designing the sampler. If w already fits the data reasonably well, as is the case some way into the optimization, then high-quality rankings (with many good documents at top ranks) have much larger probability than poor-quality ones. a_y is usually a ranking gain or loss. Ranking gain is vanishingly small for all but a small minority of ys. However, it is not clear ab initio if and how the magnitudes of a_y and $\Pr(y|x_k; w)$ line up.

Therefore, a reasonable strategy would be to pick restart states with a variety of $\Pr(y|x_k; w)$. Assuming a reasonable current value of w, this means the restart states (rankings) should be picked to have a variety of losses Δ.

Sampling at the extremes is easy: the best ranking ($\Delta = 0$) has all good documents followed by all bad documents and the worst ranking ($\Delta = 1$) has all bad documents followed by all good documents. At first we used only these two restart points, and tuned the probability of sampling one vs. the other. Later, we also created a bigger set of restart rankings with a variety of Δs, and used the distribution $\Pr(y) \propto \exp(k\Delta(y))$, with a flexible skew parameter k, to sample a restart state.

4.2. Transitions

A well-trained w must distinguish very good rankings from very bad ones, but also make fine distinctions between good and excellent rankings. Having started from ys of diverse quality, we will design each transition to mutate y_{in} to y_{out} while keeping the quality y_{out} close to that of y_{in}, as judged by all the Δs that are commonly used. Together, these two strategies should collect a reasonable sample $\hat{Y}$.

Elaborating upon the mutation step, if we want to flip a good-bad pair, then the change in Δ will be large if the bad document is close to the worst possible and the good document is close to the best possible. We want to reduce the probability of such flips, as compared to flips that change Δ in small amounts.

For the current w, the goodness of a good document g is reflected by the number of bad document it beats (n_g). Similarly, the badness of a bad document b is reflected by the number of good document that beat it ($n^+ - n_b$). Continuing the above line of thought, we should encourage a flip of the bit indexed by g, b if g is not that good (small n_g) and b is not that bad (large n_b). This is arranged using the flip probability θ in Figure 2. If $y_{in}[idx] = +1$ and $n^- - n_g + n_b$ is large and we flip, then the resultant y_{out} will have gain close to that of y_{in}. A symmetric argument holds when $y_{in}[idx] = -1$.

5. Experiments

5.1. Testbed

All algorithms were coded in Java 1.6 and run on 64-bit JVMs on 2.4GHz Xeon processors with 8–16GB RAM.

5.1.1. Data sets. For our accuracy studies we primarily use the well-known **LETOR** benchmark [14], version 3. It consists of seven different data sets (TD2003, TD2004, HP2003, HP2004, NP2003, NP2004, OHSUMED) with a total of 575 queries. We clean the dataset in standard ways [5] by removing documents of a particular query that has conflicting relevance labels, and removing queries that have no relevant document. 5-fold cross-validation as prescribed by LETOR was used throughout, in particular, to tune C in the regularizing term $\|w\|_2^2/C$ in all objectives.

Because the data sets have different number of queries each, we also include a weighted average ("microaverage") of the accuracies across all datasets. The weight of accuracy for a particular dataset is given by the number of queries in that dataset.

5.1.2. Baseline algorithms and evaluation. We compare our proposed system LogRank against these competitive baseline algorithms:

SVMMAP: Based on structured learning [17], this directly optimizes a convex upper bound to a loss function that reflects mean average precision (MAP).

RANKBOOST: We implemented the standard RANKBOOST algorithm [6].

RANKSVM: We implemented standard RankSVM [8].

In addition, we quote accuracies of BOLTZRANK and LISTNET reported in [19] for two data sets, with the caveat that they were run on unclean data.

5.2. Results

Figure 3 summarizes the results from all baselines and all our proposed approaches. There are four subtables, for MAP, NDCG@1, NDCG@5, and NDCG@10. Each column corresponds to a data set; the last column is the microaverage. In each column, the best three cells are shaded green, yellow and tan. We immediately see that, often, the top three cells are all within the family of new algorithms presented in this paper. Also, clearly the baselines are rarely the best. The gains we see are at par with, or larger than, typical gains seen with new algorithms for learning to rank. The following sections provide detailed commentary on the relative performance of all competitors.

5.2.1. MLE. Our first comparison is between the baselines and the ranking achieved by the MLE of w obtained by using the distribution (3). From Figure 3, there is nothing too special about the MLE w, although it does beats *all* baselines 16 out of 32 times (There are 8 columns evaluated on 4 evaluation measures). This is not surprising, because MLE makes no use of Δ.

5.2.2. Expected Gain (ExpGain). Next we compare the performance of the non-convex objective given in equation 7 against the baselines. Note that ExpGain can be trained with different gain functions: AUC, MAP, or NDCG. Overall, ExpGain is much better than MLE. More specifically,

- ExpGain is the best 26 out of 32 times.
- ExpGain_MAP beats SVMMAP (both trained on MAP) 23 out of 32 times.
- ExpGain_AUC beats RANKSVM (both trained on AUC) 30 out of 32 times.
- Our "Combined" (microaveraged) accuracy is always greater than that of the baselines.
- ExpGain beats MLE 21 out of 32 times.

5.2.3. Convex loss (ConvexLoss). Like ExpGain, ConvexLoss can be trained with three loss functions corresponding to AUC, MAP and NDCG. The convex upper bound formulation gives the best result overall. More specifically,

- ConvexLoss is the absolute best in 26 out of 32 times.
- ConvexLoss_MAP beats SVMMAP (both trained on MAP) in 26 out of 32 times.
- ConvexLoss_AUC beats RANKSVM (both trained on AUC) in 25 out of 32 times.
- ConvexLoss_NDCG is the best method across all datasets and is also the best in terms of "Combined" (microaveraged) accuracy.
- ConvexLoss beats MLE in 29 out of 32 times, clearly showing the worth of information from $\Delta(y)$.
- ConvexLoss beats ExpGain in 25 out of 32 times.
- ConvexLoss_NDCG beats ExptGain_NDCG in 25 out of 32 times.
- Even the *worse* of ConvexLoss_MAP and ConvexLoss_NDCG is better than the *best* of the baselines in 23 out of 32 times.

We also observe that L_3 also appears as the best in each column some number of times, but not as often as ConvexLoss.

5.3. Sensitivity to sampling variations

Given the basis of our accuracy is the sampling scheme discussed in Section 4, we present some diagnostic tests to understand how the sampler affects the optimizer and learner.

5.3.1. Saturation. Our first concern regarding sampling is whether there is a natural saturation of accuracy as $|\hat{Y}|$ is increased. Figure 4 shows the result. We observe that accuracy increases with increasing sample size, although, at the sample sizes shown, we are close to saturation.

5.3.2. Choice of restarts. In the results above, we used two restart states: the best possible ranking y_q for query q ($\Delta = 0$), and the exact opposite: the worst possible ranking

Metric	Method	TD2003	TD2004	HP2003	HP2004	NP2003	NP2004	OHSU-MED	Microavg
NDCG@1	SVMmap	0.364	0.493	0.765	0.665	0.591	0.573	0.676	0.621
	Rankboost	0.360	0.453	0.714	0.653	0.636	0.530	0.617	0.600
	RankSVM	0.242	0.413	0.765	0.695	0.622	0.600	0.667	0.615
	ListNet		*0.360*					*0.530*	
	BoltzRank2		*0.478*					*0.568*	
	MLE	0.444	0.493	0.786	0.665	0.629	0.596	0.734	0.652
	ExpGain_AUC	0.302	0.480	0.779	0.695	0.637	0.600	0.696	0.638
	ExpGain_MAP	0.467	0.453	0.779	0.680	0.660	0.613	0.696	0.652
	ExpGain_NDCG	0.467	0.453	0.765	0.680	0.653	0.613	0.696	0.647
	L_3 AUC	0.467	0.480	0.779	0.695	0.629	0.584	0.714	0.649
	L_3 MAP	0.467	0.520	0.786	0.709	0.653	0.582	0.706	0.661
	L_3 NDCG	0.427	0.520	0.800	0.681	0.653	0.579	0.705	0.657
	ConvexLoss_AUC	0.322	0.453	0.778	0.682	0.661	0.568	0.676	0.633
	ConvexLoss_MAP	0.427	0.507	0.771	0.724	0.660	0.541	0.705	0.651
	ConvexLoss_NDCG	0.447	0.507	0.772	0.709	0.667	0.598	0.724	**0.662**
NDCG@5	SVMmap	0.368	0.363	0.829	0.835	0.801	0.830	0.621	0.706
	Rankboost	0.325	0.349	0.854	0.821	0.816	0.768	0.597	0.698
	RankSVM	0.312	0.336	0.842	0.852	0.811	0.823	0.618	0.705
	ListNet		*0.332*					*0.443*	
	BoltzRank2		*0.363*					*0.491*	
	MLE	0.367	0.365	0.874	0.854	0.822	0.817	0.614	0.720
	ExpGain_AUC	0.347	0.365	0.849	0.862	0.813	0.833	0.618	0.715
	ExpGain_MAP	0.372	0.339	0.859	0.875	0.838	0.825	0.570	0.714
	ExpGain_NDCG	0.370	0.332	0.860	0.872	0.836	0.834	0.569	0.714
	L_3 AUC	0.385	0.368	0.860	0.880	0.824	0.836	0.610	0.724
	L_3 MAP	0.396	0.362	0.866	0.883	0.827	0.829	0.603	0.724
	L_3 NDCG	0.370	0.366	0.876	0.885	0.830	0.825	0.606	0.726
	ConvexLoss_AUC	0.319	0.354	0.860	0.890	0.820	0.813	0.616	0.716
	ConvexLoss_MAP	0.389	0.371	0.866	0.874	0.826	0.831	0.605	0.724
	ConvexLoss_NDCG	0.385	0.368	0.878	0.892	0.843	0.828	0.608	**0.732**
NDCG@10	SVMmap	0.385	0.343	0.840	0.845	0.821	0.847	0.601	0.712
	Rankboost	0.347	0.340	0.868	0.845	0.836	0.806	0.582	0.711
	RankSVM	0.314	0.315	0.850	0.877	0.830	0.856	0.598	0.712
	MLE	0.387	0.338	0.883	0.872	0.843	0.830	0.581	0.724
	ExpGain_AUC	0.353	0.346	0.862	0.885	0.832	0.862	0.600	0.723
	ExpGain_MAP	0.372	0.323	0.870	0.895	0.855	0.847	0.558	0.722
	ExpGain_NDCG	0.366	0.309	0.867	0.890	0.859	0.850	0.553	0.719
	L_3 AUC	0.405	0.341	0.874	0.898	0.844	0.851	0.582	0.729
	L_3 MAP	0.402	0.335	0.871	0.900	0.848	0.848	0.576	0.727
	L_3 NDCG	0.400	0.337	0.884	0.900	0.842	0.843	0.582	0.729
	ConvexLoss_AUC	0.331	0.334	0.872	0.905	0.838	0.842	0.602	0.724
	ConvexLoss_MAP	0.390	0.338	0.873	0.885	0.848	0.853	0.577	0.726
	ConvexLoss_NDCG	0.397	0.336	0.884	0.904	0.858	0.853	0.578	**0.733**
MAP	SVMmap	0.296	0.259	0.781	0.746	0.707	0.709	0.563	0.626
	Rankboost	0.277	0.263	0.782	0.739	0.740	0.664	0.545	0.624
	RankSVM	0.244	0.233	0.794	0.764	0.726	0.718	0.563	0.629
	ListNet		*0.223*					*0.440*	
	BoltzRank2		*0.239*					*0.460*	
	MLE	0.330	0.258	0.820	0.755	0.732	0.695	0.543	0.639
	ExpGain_AUC	0.274	0.259	0.802	0.774	0.732	0.715	0.557	0.637
	ExpGain_MAP	0.314	0.232	0.808	0.775	0.754	0.723	0.516	0.638
	ExpGain_NDCG	0.294	0.220	0.804	0.772	0.752	0.718	0.517	0.633
	L_3 AUC	0.342	0.259	0.807	0.771	0.739	0.704	0.544	0.641
	L_3 MAP	0.340	0.259	0.813	0.786	0.749	0.703	0.540	0.646
	L_3 NDCG	0.323	0.259	0.825	0.773	0.745	0.705	0.541	0.645
	ConvexLoss_AUC	0.262	0.252	0.809	0.778	0.750	0.696	0.558	0.640
	ConvexLoss_MAP	0.334	0.261	0.807	0.791	0.744	0.689	0.542	0.642
	ConvexLoss_NDCG	0.325	0.260	0.816	0.793	0.762	0.720	0.543	**0.651**

Figure 3. Comparison of all the proposed formulations and algorithms. Each column corresponds to a data set; the last column is a microaverage (see text). In each column, the best three cells are shaded green, yellow and tan.

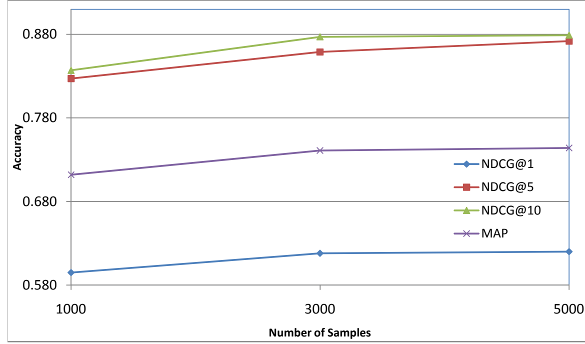

Figure 4. Effect of $|\hat{Y}|$ on test accuracy on HP2004 for ConvexLoss_NDCG.

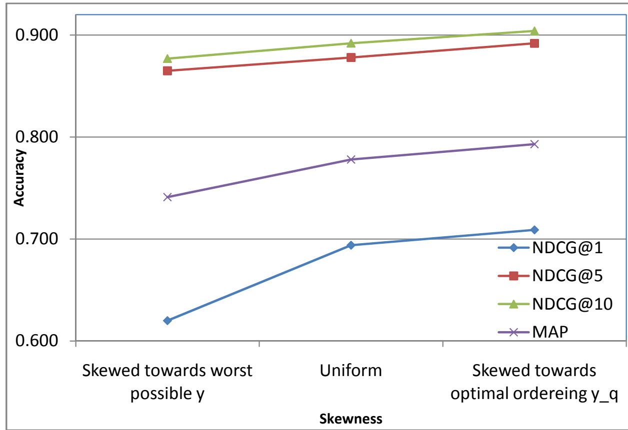

Figure 5. Effect of restart skew between $\Delta = 0$ and $\Delta = 1$ on accuracy on HP2004 for ConvexLoss_NDCG.

($\Delta \to 1$). Here we study the restart process in more detail.

First, we study the effect of setting the mix of restarts in three ways: skewed (probability 0.9) toward $\Delta = 0$, likewise skewed toward $\Delta = 1$, and balanced between the two. Figure 5 shows that skewing toward $\Delta = 0$ generally increases accuracy. (The number of restarts and samples were kept fixed.)

In the second experiment, we handcrafted a number of restarts with diverse Δ values in $[0, 1]$. Then we used ths distribution $\Pr(y) \propto \exp(k\Delta(y))$ to sample restarts. For $k > 0$, this means we skew toward $\Delta = 1$. For $k < 0$, we skew toward $\Delta = 0$. Representative results are shown in Figure 6: $k \ll 0$ seems uniformly better.

$k \to$	5	0	-5	-10	-20
MAP	.033	.089	.706	.774	.827
NDCG@10	.041	.124	.815	.905	.918

Figure 6. Effect of restart skew with more than two restart states.

5.3.3. Δ smear. We were surprised by the consistent signal that skewing toward the $\Delta = 0$ restart is the best policy. From considerations of sampling accuracy as well as presenting both good and bad rankings to the learner, we had anticipated that a more even mix of restart Δs would work better.

To investigate the unexpected observations, we started from three restarts: $\Delta = 0, \Delta \approx 0.5, \Delta = 1$, and plotted the density of Δs in the final sample. Results are shown in Figure 7.

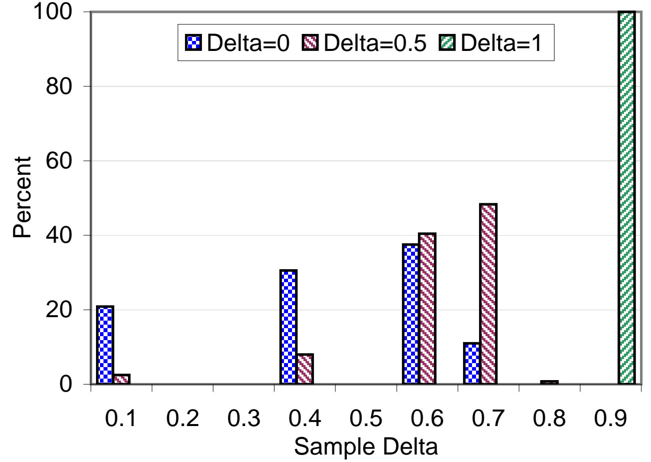

Figure 7. Smear of the sampled Δ density for three restart seeds.

Note that if n^+ is small, Δ changes quite discretely and not all Δ values are realizable by rankings. E.g., the slightest perturbation from $\Delta = 0$ may raise Δ to a minimum of 0.5 in case of MAP. Hence the empty buckets.

The difference in the spread of sampled Δs immediately stands out: the smear of sampled Δs around the seed $\Delta = 1$ is much smaller than that around the seed $\Delta = 0$. If we are at a state with $\Delta \approx 1$, flipping a random g, b pair is unlikely to perturb Δ much. On the contrary, given that, for most queries, $n^+ \ll n^-$, a random g, b flip on a good ranking can turn it very bad.

In other words, the sampler shown in Figure 2 results in different spreads of sampled Δs for different seed Δs. Therefore, a skew toward $\Delta = 0$ does not deprive the learner from rankings with a variety of Δs. In fact, if w is already good enough, it may be better to present great and good rankings to the learner than to waste samples on really bad rankings.

5.3.4. Optimization objective dynamics. We also traced the evolution of the exact and sampled objectives as w evolved through iterations of the optimizer. While these values were not very close together, each improved as iterations progressed. Based on these studies, we hypothesized that

- w quickly evolves to a reasonably good model.

- This means that for any y with small $\Delta(y)$, $w^\top \phi(x, y)$ is large.
- Figure 8 shows that a large $w^\top \phi(x, y)$ usually trumps a small $\Delta(y)$.
- This means that, in our sampled estimation of $\sum_{y'} \exp(-w^\top \delta\phi_q(y') + \Delta_q(y'))$, preferably picking y' with large $w^\top \phi_q(x, y')$ (i.e., small $\Delta_q(y')$, is adequate.

(Figure 8 shows, for 10 random queries and 10 arbitrary y rankings per query, a scatter of the values of $w^\top \phi(x, y)$, averaged over the first 10 iterations, as w evolves against the fixed $\Delta \in [0, 1]$. The latter is always quite small compared to the former, except at the initialization $w = \vec{0}$.)

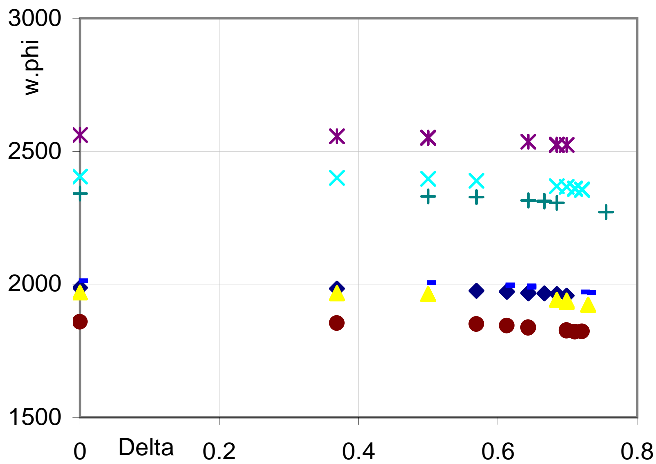

Figure 8. As w evolves, $w^\top \phi(x, y)$ quickly begins to dominate $\Delta(y)$.

Summarizing, our sensitivity study on the sampling process explains the dependency between sampling policies and system accuracy, and gives a clear recipe for sampling rankings based on the template in Figure 2.

6. Conclusion

The central problem in learning to rank is to approximate the loss function with tractable surrogates. One way is to parametrically model a conditional probability, and then minimize the expected loss under this distribution. Compared to other approaches such as neural networks and structured max-margin learners, the direct conditional probability approach was not well-explored before our work. This was probably because evaluating the partition function is tricky. We gave closed form for one common setting, and gave a new Monte Carlo sampling technique for other general settings. Despite its simplicity, our new approach shows significant accuracy gains compared to recent, competitive algorithms. A natural future direction would be to guide the ConvexLoss optimizer with some combination of all the loss functions.

References

[1] A. L. Berger, S. A. D. Pietra, and V. J. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):3971, 1996.
[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
[3] C. Burges. Learning to rank for Web search: Some new directions. Keynote talk at SIGIR Ranking Workshop, July 2007.
[4] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007.
[5] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *SIGKDD Conference*, pages 88–96. ACM, 2008.
[6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.
[7] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR Conference*, pages 41–48, 2000.
[8] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD Conference*, pages 133–142. ACM, 2002.
[9] T. Joachims. A support vector method for multivariate performance measures. In *ICML*, pages 377–384, 2005.
[10] K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments (parts 1 and 2). *Information Processing and Management*, 36(6):779–840, 2000.
[11] Q. V. Le and A. J. Smola. Direct optimization of ranking measures. arXiv:0704.3359v1, Feb. 2008.
[12] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528, 1989.
[13] T.-Y. Liu. Learning to rank for information retrieval. Tutorial at SIGIR, 2008.
[14] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR Workshop*, 2007.
[15] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
[16] M. Taylor, J. Guiver, S. Robertson, and T. Minka. SoftRank: Optimising non-smooth rank metrics. In *WSDM Conference*. ACM, 2008.
[17] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6(Sep):1453–1484, 2005.
[18] V. Vapnik, S. Golowich, and A. J. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems*. MIT Press, 1996.
[19] M. N. Volkovs and R. S. Zemel. BoltzRank: Learning to maximize expected ranking gain. In *ICML*, 2009.
[20] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR Conference*, pages 271–278, 2007.