Structured Learning for Non-Smooth Ranking Losses

Soumen Chakrabarti
IIT Bombay
soumen@cse.iitb.ac.in

Rajiv Khanna
IIT Bombay
rajivk@cse.iitb.ac.in

Uma Sawant
IIT Bombay
uma@cse.iitb.ac.in

Chiru Bhattacharyya
IISc Bangalore
chiru@csa.iisc.ernet.in

ABSTRACT

Learning to rank from relevance judgment is an active research area. Itemwise score regression, pairwise preference satisfaction, and listwise structured learning are the major techniques in use. Listwise structured learning has been applied recently to optimize important non-decomposable ranking criteria like AUC (area under ROC curve) and MAP (mean average precision). We propose new, almost-linear-time algorithms to optimize for two other criteria widely used to evaluate search systems: MRR (mean reciprocal rank) and NDCG (normalized discounted cumulative gain) in the max-margin structured learning framework. We also demonstrate that, for different ranking criteria, one may need to use different feature maps. Search applications should not be optimized in favor of a single criterion, because they need to cater to a variety of queries. E.g., MRR is best for navigational queries, while NDCG is best for informational queries. A key contribution of this paper is to fold multiple ranking loss functions into a multi-criteria max-margin optimization. The result is a single, robust ranking model that is close to the best accuracy of learners trained on individual criteria. In fact, experiments over the popular LETOR and TREC data sets show that, contrary to conventional wisdom, a test criterion is often not best served by training with the same individual criterion.

Categories and Subject Descriptors: I.2.6 [Computing Methodologies]: Artifical Intelligence — *Learning*; H.3.3 [Information Systems]: Information Storage and Retrieval — *Information Search and Retrieval*.
General Terms: Algorithms, Experimentation.
Keywords: Max-margin structured learning to rank, Non-decomposable loss functions.

1. INTRODUCTION

Learning to rank is an active research area where supervised learning is increasingly used [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. The main challenge in adapting supervised learning is that for ranking problems, the evaluation criterion or "loss function" is usually defined over the *permutation* induced by the scoring function over response instances (see Section 2.2), and hence the loss function is not decomposable over instances as in regular Support Vector Machines (SVMs) [11]. Moreover, common ways to formulate learning with these loss functions result in intrinsically non-convex and "rough" optimization problems. This is a central challenge in learning to rank.

1.1 Existing algorithms

Algorithms for learning to rank may view instances

Itemwise [10] and regress them to scores, then sort the instances by decreasing score.

Pairwise [2, 3], which is highly suited for clickthrough data, and leads to a simple loss function decomposable over preference pairs; however, it is thereby not sensitive to absolute ranks.

Listwise [4, 9], and use structured learning [12] algorithms, which is our focus here.

The large-margin structured learning framework [12] fits a model to minimize the loss of a whole permutation, not individual or pairs of items. This approach has been used to optimize non-decomposable ranking criteria, like the area under the ROC curve (AUC) [4] and mean average precision (MAP) [9]. However, other widely-used criteria in Information Retrieval and Web search, such as mean reciprocal rank (MRR) [13] or normalized discounted cumulative gain (NDCG) [14], had no efficient direct optimizers.

The advantage of the structured learning approach is that it helps break down the difficult non-convex ranking loss optimization problem into a convex quadratic program and a combinatorial optimization problem which, as we shall see, is often tractable and simple for ranking applications.

The second framework approximates the non-convex and discontinuous ranking loss function with a somewhat more benign (but often still non-convex) surrogate, which is then optimized using gradient descent and neural networks [3, 7, 8, 15]. A very interesting variant is LambdaRank [6] which models only the gradient, with an unmaterialized objective. A potential problem with this family is that non-convex optimization behavior is tricky to replicate accurately in general, requiring many bells and whistles to tide over local optima. In fact, a recent approach using boosted ordinal regression trees (McRank) [10] has proved surprisingly competitive to gradient methods and put itemwise approaches back in the race. In this paper we will not focus on this family, except to compare the best structured learning approaches with McRank, to show that listwise structured learning remains very competitive.

1.2 Our contributions

Our first contribution is to augment the class of non-smooth ranking loss functions that can be directly and efficiently (in near-linear time) optimized for listwise structured learning. Specifically, we propose new, almost linear-time algorithms, SVMndcg for NDCG (Section 3.4) and SVMmrr for MRR (Section 3.6). Therefore, now we can optimize efficiently for AUC, MAP, NDCG and MRR within the structured ranking framework.

Structured ranking requires us to design a *feature map* $\phi(x, y)$ over documents x and a proposed ranking y. Our second contribution is a close look (Section 3.1) at feature map design and feature scaling: all-important but somewhat neglected aspects of structured ranking. Specifically, our feature maps for MRR and NDCG are different, and this affects accuracy (Section 4.3). It also greatly affects the numerical behavior of the optimizer (Section 4.2). We give some guidelines on how to check if a feature map, in conjunction with a loss function, is healthy for the optimizer.

We perform a thorough comparison, using standard public data sets (see Figures 1, 9), of test accuracies in terms of MAP, MRR, and NDCG when trained with structured rank learners that are optimizing for each of these loss functions separately. Conventional wisdom suggests that a system

KDD'08 August 24–27, 2008, Las Vegas, Nevada, USA.

trained to optimize MAP should be best for test MAP, a system optimized for MRR should be best for test MRR, etc. Surprisingly, across five data sets, we see very little evidence of this. Often, the best test loss of a certain type is obtained by training with a different loss function. We conjecture that this is because conventional feature maps for ranking are not well-matched to commonly-used ranking loss functions (Section 4.2).

Web search users have diverse needs. Even if we could, it would be undesirable to ultra-optimize in favor of one criterion. MRR is best for navigational queries ("IBM") and factual questions ("giraffe height"), where only the first URL (`http://www.ibm.com`) or answer (18 feet) matter. In contrast, NDCG is best for collecting information about a topic.

Our third contribution is a robust max-margin optimization (SVMcombo) of a combined loss function (Section 3.7). We show that SVMcombo's test performance on any criterion (MAP, MRR, NDCG) is close to that of the best components of the combination (Section 4.5).

We also report on running time and scalability comparisons between structured rank learners and a prominent recent contender in accuracy (McRank—boosted regression trees) and find, on the public LETOR data set [16], that structured learners are considerably faster (Section 3.8) while also being more accurate in two out of three data sets.

2. BACKGROUND

2.1 Testbed and data sets

Figure 1 summarizes the leading algorithms and data sets on which they have been evaluated. RankSVM [2], StructSVM [12], and SVMmap [9] codes are in the public domain. Implementations of ranking using gradient-descent and boosting are not public.

From Figure 1 it is evident that many of the data sets are proprietary, and many implementations are not readily available. As a result, there are hardly any data sets over which many algorithms have been compared directly. We have implemented SVMauc [4], SVMmap [9], DORM [18], McRank [10], as well as our new proposals SVMmrr and SVMndcg, in a common open-source Java testbed (`http://www.cse.iitb.ac.in/soumen/doc/StructRank/`).

We ran all the algorithms on the well-known LETOR benchmark [16] that is now widely used in research on learning to rank. We also ran all algorithms but one on TREC 2000 and 2001 data prepared by Yue *et al.* [9]. More details are in Section 4.1. In many cases, the behavior of different algorithms differed on the three data sets. This highlights the importance of shared, standardized data to avoid potentially biased conclusions.

2.2 Ranking evaluation criteria

Suppose q is a query from a query set Q. For each document x_i in the corpus, we use q together with x_i to compute a feature vector we call $x_{qi} \in \mathbb{R}^d$ whose elements encode various match and quality indicators. E.g., one element of x_{qi} may encode the extent of match between q and the page title, while another element may be the PageRank of the node corresponding to the ith document in the Web graph. Collectively, these feature vectors over all documents, for fixed query q, is called x_q.

Learning a ranking model amounts to estimating a weight vector $w \in \mathbb{R}^d$. The score of a test document x is $w^\top x$. Documents are ranked by decreasing score.

For evaluation, suppose the exact sets of relevant and irrelevant documents, G_q and B_q, are known for every query q. For simplicity these can be coded as $z_{qi} = 1$ if the ith document is relevant or "good" for query q, and 0 otherwise, i.e., the document is "bad". Let $n_q^+ = |G_q|$ and $n_q^- = |B_q|$. We will drop q when clear from context.

2.2.1 Pair preference and AUC

For every query q, every good document x_{qi} and every bad document x_{qj}, we want x_{qi} to rank higher than x_{qj}, denoted "$x_{qi} \succ x_{qj}$" and satisfied if $w^\top x_{qi} > w^\top x_{qj}$. Pair preferences can be asserted even when absolute relevance values are unknown or unreliable, as with clickthrough data. Usually, learning algorithms [1, 2] seek to maximize over w (a smooth approximation to) the number of pair preferences satisfied. The number of satisfied pairs is closely related to the area under the *receiver operating characteristic* (ROC) curve [4].

A long-standing criticism of pair preference satisfaction is that all violations are not equal [19]; 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.

2.2.2 Mean reciprocal rank (MRR)

In a navigational query, the user wants to quickly locate a URL that is guaranteed to satisfy her information need. In question answering (QA), many questions have definite answers, any one correct answer is adequate. MRR is well-suited to such occasions [13]. Suppose, in the document list ordered by decreasing $w^\top x_{qi}$, the topmost rank at which an answer to q is found is r_q. (Note: For consistency with code all our ranks begin at zero, not one.) The reciprocal rank for a query q is $1/(1 + r_q)$. Averaging over q, we get $\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{1+r_q}$ where Q is the set of all queries. Often a cutoff of k is used:

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \begin{cases} 1/(1 + r_q), & r_q < k \\ 0, & r_q \geq k \end{cases} \qquad \text{(MRR)}$$

The ideal ranking ensures an MRR of 1 (assuming there is at least one relevant document for every query).

2.2.3 Normalized discounted cumulative gain (NDCG)

Of recent interest in Information Retrieval and Machine Learning communities is NDCG, which, unlike MRR, does accumulate credit for second and subsequent relevant documents, but discounts them with increasing rank. The DCG for a specific query q and document order is $\text{DCG}(q) = \sum_{0 \leq i < k} G(q, i) D(i)$ where $G(q, i)$ is the gain or relevance of document i for query q and $D(i)$ is the discount factor given by [14]

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

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

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

pushing all the relevant documents to the top. Now define

$$\text{NDCG}(q) = \frac{\text{DCG}(q)}{\text{DCG}^*(q)} = \frac{\sum_{0 \leq i < k} z_{qi} D(i)}{\text{DCG}^*(q)} \qquad \text{(NDCG)}$$

	Public					Not public					
Algorithm	OHSUMED	TD2003	TD2004	TREC2000	TREC2001	MS Web1	MS Web2	MS Web3	MS Web4	Y! Web1	Y! Web2
RankSVM, public, reimplemented here [2]	•[15, 17]	•	•						[18]	[17]	[17]
SVMauc, public, reimplemented here [4]	•	•	•	• [9]	• [9]				[18]		
SVMmap, public, reimplemented here [9]	•	•	•	• [9]	• [9]						
SVMmrr, proposed here	•	•	•	•	•						
SVMndcg, proposed here	•	•	•	•	•						
DORM, not public, reimplemented here	•[17]	•	•	•	•				[18]	[17]	[17]
RankNet, not public [3]						[6]					
LambdaRank, not public [6]						[6, 10]	[10]	[10]			
SoftRank, not public [8, 15]	[15]										
McRank, not public, reimplemented here [10]	•	•	•	○	○	[10]	[10]	[10]			

Figure 1: Algorithms for learning to rank and data sets where they have been evaluated. "MS Web" data is from Microsoft, "Y! Web" data is from Yahoo. "•" means the evaluation is reported here. "○" means the RAM/CPU requirements prevented evaluation. [2, 3, 4, 5] predate OHSUMED, TD2003, TD2004.

and average 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 $G(q,i) = z_{qi}$.

2.2.4 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:

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

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

2.3 Structured ranking basics

In structured ranking, the input is the set of documents x_q and the output is a total or partial order y over all the documents.

Ranking is achieved through two devices: a *feature map* $\phi(x_q, y)$ and a *model weight vector* w. The score of a ranking y wrt x_q, ϕ and w is $w^\top \phi(x_q, y)$. The intention is, given a trained model w and documents x_q, to return the best ranking $\arg\max_y w^\top \phi(x_q, y)$. $\phi(x,y)$ is usually chosen so that, given w, this maximization amounts to sorting documents by decreasing $w^\top x_{qi}$.

The ideal ranking for query q is called y_q^*. It places all good documents in G_q at top ranks and all bad documents B_q after that. The third component of structured learning is a *loss function*: Any order y incurs a *ranking loss* $\Delta(y_q^*, y) \geq 0$. $\Delta(y_q^*, y_q^*) = 0$ and the worse the ranking y, larger the value of $\Delta(y_q^*, y)$.

In structured rank learning the goal is to estimate a scoring model w via this generic StructSVM optimization [12]:

$$\arg\min_{w;\xi\geq 0} \tfrac{1}{2} w^\top w + \tfrac{C}{|Q|} \sum_q \xi_q \quad \text{s.t.} \qquad (1)$$
$$\forall q, \forall y \neq y_q^* : \quad w^\top \phi(x_q, y_q^*) \geq w^\top \phi(x_q, y) + \Delta(y_q^*, y) - \xi_q.$$

Intuitively, if y is a poor ranking, $\Delta(y_q^*, y)$ is large and we want w to indicate that, i.e., we want $w^\top \phi(x_q, y_q^*) \gg w^\top \phi(x_q, y)$ in that case. If not, w needs to be refined.

At any step in the execution of a cutting plane algorithm [12], there is a working set of constraints, a current model w and current set of slack variables ξ_q, and we wish to find a *violator* $(\hat{q}, \hat{y} \neq y_{\hat{q}}^*)$ such that

$$w^\top \phi(x_{\hat{q}}, y_{\hat{q}}^*) + \epsilon < w^\top \phi(x_{\hat{q}}, \hat{y}) + \Delta(y_{\hat{q}}^*, \hat{y}) - \xi_{\hat{q}},$$

where $\epsilon > 0$ is a tolerance parameter, and then add the constraint "$w^\top(\phi(x_{\hat{q}}, y_{\hat{q}}^*) - w^\top \phi(x_{\hat{q}}, \hat{y})) \geq \Delta(y_{\hat{q}}^*, \hat{y}) - \xi_q$" to the working set. This means we need to find, for each q,

$$\arg\max_{y\neq y_q^*} H(y; x, w) = \arg\max_{y\neq y_q^*} w^\top \phi(x_q, y) + \Delta(y_q^*, y) \qquad (2)$$

If w were "perfect", maximizing $w^\top \phi(x_q, y)$ would give us an ideal y with very small Δ. Intuitively, the maximization (2) finds weaknesses in w where a large $w^\top \phi(x_q, y)$ can coexist with a large $\Delta(y_q, y)$. Then the next step of the cutting plane algorithm proceeds to improve w and adjust ξ suitably.

Applying StructSVM to a problem [9, 18] amounts to designing $\phi(x,y)$ and giving an efficient algorithm for (2). The critical property of the cutting plane algorithm is that, for any fixed ϵ, a constant number of violators are considered before convergence. Therefore, if each invocation of (2) takes linear time, the overall training algorithm is also linear-time. The details, which are now standard, can be found in [12, 4, 5].

3. ALGORITHMS AND ANALYSIS

3.1 Feature map design

The representation of x_{qi} as a feature vector comes from domain knowledge, but the design of the feature map $\phi(x,y)$ is an integral part of learning to rank. Here we review two known feature maps and propose one.

3.1.1 Partial order ϕ_{po}

For pair preferences and AUC, the partial order feature map is a natural choice. If y encodes a partial order, it is indexed as y_{ij} where $z_{qi} = 1$ and $z_{qj} = 0$. $y_{ij} = 1$ if the partial order places x_{qi} before x_{qj}. $y_{ij} = -1$ if the partial order (mistakenly) places x_{qi} after x_{qj}. If $y_{ij} = 0$, x_{qi} and x_{qj} are incomparable in the partial order. Note that $y_{ij}^* = 1$ for all i, j. With this coding of y, a common feature function used by Joachims and others [4, 9] is

$$\phi_{\mathrm{po}}(x_q, y) = \tfrac{1}{n_q^+ n_q^-} \textstyle\sum_{g\in G_q, b\in B_q} y_{gb}(x_{qg} - x_{qb}) \qquad (3)$$

where g indexes a good document and b indexes a bad document.

In practice, the pair-averaging scale factor $1/(n_q^+ n_q^-)$ is absolutely critical for any learnability of w; without it, we get almost zero accuracy of all kinds (AUC, NDCG, MAP, MRR). Therefore, proper scaling across queries is also an integral part of feature map design.

3.1.2 New insights and feature map ϕ_{mrr}

With ϕ_{po} defined as above, consider

$$\delta\phi_x(y^*, y) = \phi(x, y^*) - \phi(x, y).$$

Note that the optimization (1) sees only $\delta\phi_x(y^*, y)$, never $\phi(x, y^*)$ or $\phi(x, y)$ separately.

Fact 3.1. *$\delta\phi_x(y^*, y)$ can be written as*

$$2\frac{1}{n_q^+ n_q^-} \sum_g \sum_{b:b\succ g} (x_g - x_b), \tag{4}$$

where "$b \succ g$" means that y places the bad document indexed by b before the good document indexed by g.

Proof. Consider a good document (index) g with two bad documents, $b_1 \succ g$ and $g \succ b_2$, in partial order y. Using (3), $\phi(x, y)$ will include terms $x_{b_1} - x_g$ and $x_g - x_{b_2}$. In y^* we will have $g \succ b_1$ and $g \succ b_2$, so $\phi(x, y^*)$ will include terms $x_g - x_{b_1}$ and $x_g - x_{b_2}$. □

This shows that, despite the global "all pairs" feel to (3), ϕ_{po} carries no information to the optimizer from documents lower than the lowest-ranked good document. We can use (4) to define an equivalent feature map that exposes the local nature of ϕ_{po}:

$$\psi(x, y) = 2\frac{1}{n_q^+ n_q^-} \sum_g \sum_{b:b\succ g} (x_b - x_g).$$

Note that $\psi(x, y^*) = \vec{0}, \quad \because \nexists b \succ g \text{ in } y^*,$

and therefore $\delta\psi_x(y^*, y) = \delta\phi_x(y^*, y)$.

Now consider MRR. Δ_{MRR} depends not on all g, but only the top-ranking good document $g_0(y)$. So the sum over all g seems out of place. Accordingly, we will define

$$\phi_{\text{mrr}}(x, y) = \sum_{b:b\succ g_0(y)} (x_b - x_{g_0(y)}). \tag{5}$$

Again, $\phi_{\text{mrr}}(x, y^*) = \vec{0}$. There is no need to scale down by n_q^+, because only one good document is contributing to the sum. We are just soft-counting the number of bad documents ahead of $g_0(y)$, so there is also no need to scale down by n_q^-.

3.1.3 Permutation feature map ϕ_{dorm}

Instead of expressing a partial order involving good-bad pairs, y may also encode a total order. A natural encoding of a total order is to set $y(i)$ to the rank of x_{qi}, where $y(i) \in \{0, \ldots, n_q - 1\}$. Given x_q and a permutation y, the feature function is

$$\phi_{\text{dorm}}(x_q, y) = \sum_{0\le i<n} A(y(i)) x_{qi} \tag{6}$$

where $A(r)$ is a heuristically designed *decay profile* [18]. E.g., the ranking evaluation measures we study here pay more attention to the documents in the top ranks. For NDCG and MRR, our attention is limited to the top k documents. To embed this knowledge in the feature function, one can set $A(r)$ to various decay functions, like $1/\sqrt{r+1}$. Thus, $\phi_{\text{dorm}}(x_q, y)$ increases the representation of top-ranked documents. Unfortunately, there is no theoretical guidance to design A. It is naturally of interest to see how $\phi_{\text{po}}, \phi_{\text{mrr}}, \phi_{\text{dorm}}$ perform at various tasks; to our knowledge such comparisons have not been done before.

3.2 Loss functions

It is easy to translate the ranking criteria reviewed in Section 2.2 into loss functions. $\Delta_{\text{AUC}}(y^*, y)$ is the fraction of pair preferences that are violated by y (y^* violates none). This can be written as

$$\Delta_{\text{AUC}}(y^*, y) = \tfrac{1}{2}\frac{1}{n^+ n^-} \sum_{g,b} (1 - y_{gb}) \tag{7}$$

Next we consider MRR, NDCG and MAP. For any y, MRR is a number between 0 and 1; $\Delta_{\text{MRR}}(y^*, y)$ is simply one minus the MRR of y. Note that the MRR of y^* is 1 if there is at least one good document for every query, which we will assume. $\Delta_{\text{NDCG}}(y^*, y)$ and $\Delta_{\text{MAP}}(y^*, y)$ are defined similarly.

3.3 Review of SVMAUC and SVMMAP

Consider optimization (2) using ϕ_{po} and Δ_{AUC} [4]. For the current fixed w in a cutting plane algorithm, let us shorthand the current score $s_{qi} = w^\top x_{qi}$, and omit q when fixed or clear from context. Then observe that

$$\begin{aligned} &\arg\max_y w^\top \phi_{\text{po}}(x_q, y) + \Delta_{\text{AUC}}(y_q^*, y) \\ &= \arg\max_y \frac{1}{n_q^+ n_q^-} \sum_{g,b} y_{gb} \left(s_g - s_b - \tfrac{1}{2}\right). \end{aligned} \tag{8}$$

In this case the best choice of y is obvious: $y_{gb} = \text{sign}(s_g - s_b - \frac{1}{2})$, and therefore the elements y_{gb} can be optimized independently. Other ranking criteria, such as MAP, MRR and NDCG lead to more non-trivial optimizations.

SVMAUC with ϕ_{po} and Δ_{AUC} admits a very efficient optimization of (2). Learning for MAP with ϕ_{po} and Δ_{MAP} is not as simple, but the following insight can be exploited to design a greedy algorithm [9].

Fact 3.2. *There is an optimal total order y for (2) with ϕ_{po} and $\Delta = \Delta_{MAP}$ such that the scores (wrt the current w) of good documents are in non-increasing order, and the scores of bad documents are in non-increasing order.*

The proof is via a swap argument. The SVMMAP algorithm of Yue *et al.* [9] greedily percolates score-ordered bad documents into the score-ordered sequence of good documents, and this is proved to be correct.

3.4 New algorithm SVMNDCG

For ϕ_{po} and Δ_{NDCG}, we present a solution to optimization (2) that takes $O\left(\sum_q (n_q \log n_q + k^2)\right)$ time, where $n_q = n_q^+ + n_q^-$. It can be verified that Fact 3.2 holds for ϕ_{po} and Δ_{NDCG} as well. However, the details of merging good and documents are slightly different from SVMMAP.

Fix a query q and consider the good documents in a list $G = x_0, \ldots, x_g, \ldots, x_{n^+-1}$ sorted by decreasing current score wrt the current w. Also let $B = x_0, \ldots, x_b, \ldots, x_{n^--1-1}$ be the bad documents sorted likewise. We will insert good documents, in decreasing score order, into B (the opposite also works). Initially, it will be easiest to visualize this as a dynamic programming table, shown in Figure 2.

Cell $[g, b]$ in the table will store a solution up to the placement of good document x_g just before bad document x_b, which means its rank in the merged list is $g + b$ (counting from 0). The contribution of the cell $[g, b]$ to the objective H comes in two parts:
CellScore(g, b) from $w^\top \phi_{\text{po}}$ is

$$\frac{\sum_{\ell=0}^{b-1}(s_{b_\ell} - s_g) + \sum_{\ell=b}^{n^--1}(s_g - s_{b_\ell})}{n_q^+ n_q^-};$$

this can be found in $O(1)$ time by precomputing prefix and suffix sums of B.

CellLoss(g, b) is $-D(g+b)/\operatorname{DCG}^*(q)$: this is the negated contribution of the gth good document to the NDCG (Discount). After the last row, we will add up (negative) loss contributions from each row and add to 1 to get Δ_{NDCG}.

S	$b \rightarrow$					
	0	...	$k-1$	$\geq k$	...	$(n^- - 1)$
0		...				
⋮						
g		•				
$k-1$		○	$[g, b]$	○		○
⋮						
$n^+ - 1$						

```
1: obtain sorted good list G and bad list B
2: initialize objective matrix S to zeros
3: for g = 0, 1, ..., n+ − 1 do
4:    for b = 0, 1, ..., n− − 1 do
5:       {g good and b bad before x_g}
6:       find cellLoss ← CellLoss(g, b) (see text)
7:       find cellScore ← CellScore(g, b) (see text)
8:       cellValue ← cellLoss + cellScore
9:       if g = 0 then
10:         {first good doc}
11:         S[g, b] ← cellValue
12:      else if g > 0 and b = 0 then
13:         {several good docs before first bad}
14:         S[g, b] ← S[g − 1, 0] + cellValue
15:      else
16:         {general recurrence with g, b > 0}
17:         p* ← arg max_{0≤p≤b} {S[g − 1, p] + cellValue}
18:         S[g, b] ← S[g − 1, p*] + cellValue
```

Figure 2: Generic SVMndcg pseudocode.

For clarity, Figure 2 shows a generic procedure that may also be useful for other loss functions. For the specific case of NDCG, the following observations simplify and speed up the algorithm considerably.

Fact 3.3. *The optimal solution can be found using a reduced table of size* $\min\{n^+, k\} \times (k+1)$ *instead of the* $n^+ \times n^-$ *table shown.*

Proof. We keep the first k columns $0, \ldots, k-1$ as-is, but columns k through n^- can be folded into a single column representing the best solution in row g for '$b \geq k$'. This is possible because, right of column k, *cellLoss* becomes zero, so the best *cellValue* is the best *cellScore*, which can be obtained by binary searching bad documents $b_k, \ldots, b_{n^- - 1}$ with key s_g. This reduces our table size to $n^+ \times (k+1)$. However, there is also no benefit to considering more than k good documents, so we can further trim the number of rows to k if $k < n^+$. □

Fact 3.4. *Instead of the general recurrence*

$$S[g, b] \leftarrow \max_{0 \leq p \leq b} \{S[g-1, p] + \text{cellValue}\},$$

which takes $\Theta(k^2)$ *time per cell and* $O(k^3)$ *time overall, we can first find the best column for the previous row* $g-1$*:*

$$b^*_{g-1} = \arg\max_b S[g-1, b], \qquad S^*_{g-1} = \max_b S[g-1, b],$$

and then set $S[g, b] = \max_{b^*_{g-1} \leq b} \{S^*_{g-1} + \text{cellValue}(g, b)\}$,

which will take k^2 *time overall.*

Proof. This involves a swap argument similar to Yue *et al.* [9, Lemma 1] to show that $b^*_{g-1} \leq b^*_g$, i.e., even though the optimal column in each row is being found greedily, these columns will monotonically increase with g. This follows the same argument as Yue *et al.* [9] and is omitted. □

Therefore, we can execute each 'argmax' step of SVMNDCG in $O(k^2 + n \log n)$ time, using ϕ_{po} and Δ_{NDCG}. The initial sorting of good and bad documents by their current scores s_{qi} dominates the time.

Because MAP needs to optimize over the location of *all* good documents, the 'argmax' (2) step in SVMMAP took time $\sum_q O(n^+_q n^-_q + n_q \log n_q)$. Because NDCG is clipped at rank k, SVMNDCG can be faster, although, in practice, the $O(n_q \log n_q)$ term tends to be the dominating term.

Not clipping at k***:*** Comparing (MRR), (NDCG) and (MAP), we see that in case of MRR and NDCG, no credit accrues for placing a good document after rank k, whereas in case of MAP, a good document will fetch some credit no matter where it is placed. This means that SVMMRR and SVMNDCG get no signal when if improves the position of good documents placed beyond rank k. We can give SVMNDCG the same benefit by effectively setting $k = \infty$. The dynamic programming or greedy algorithms can be adapted, like SVMMAP, to run in $O\left(\sum_q (n^+_q n^-_q + n_q \log n_q)\right)$ time. We will call this option SVMNDCG-NC, for "no clip".

3.5 Review of the DORM algorithm

A different feature encoding, ϕ_{dorm} described in Section 3.1.3, was used very recently by Le *et al.* [17, 18] to perform a Direct Optimization of Ranking Measures (DORM). In this case, optimization (2) takes the form

$$\begin{aligned} &\arg\max_y \textstyle\sum_i A(y(i))(w^\top x_i) + \Delta_{\text{NDCG}}(y^*, y) \\ &= \arg\max_y \textstyle\sum_i A(y(i))(w^\top x_i) - \sum_i \frac{D(y(i))}{\text{DCG}^*} z_i \end{aligned}$$

This is equivalent to filling in a permutation matrix (a square 0/1 matrix with exactly one 1 in each row and column) π to optimize an assignment problem [20] of the form

$$\arg\max_\pi \textstyle\sum_{i,j} \pi_{ij}(s_i A(j) - z_i d_j).$$

The Kuhn-Munkres assignment algorithm takes $O(n^3)$ time in the worst case, which is much larger than the time taken by SVMNDCG. The time can be reduced by making A very sparse. E.g., we might force $A(r) = 0$ for $r > k$, but the resulting accuracy is inferior to a smooth decay such as $A(r) = 1/\sqrt{1+r}$ [18], which needs $O(n^3)$ time.

An important limitation of DORM, thanks to using the assignment paradigm, is that it cannot "count good documents to the left of a position", and so cannot deal with MAP or MRR at all.

As we shall see in Section 4.4, SVMNDCG is substantially faster than DORM while having quite comparable accuracy.

3.6 New algorithm SVMMRR

Because of the change in feature map from ϕ_{po} to ϕ_{mrr}, we have to redesign the 'argmax' routine. The pseudocode for solving (2) in SVMMRR is shown in Figure 3. Below we explain how it works.

```
inputs: current w, x_1, ..., x_n, clip rank k
obtain sorted good list G = g_0, ..., g_{n+-1} and bad list B = b_0, ..., b_{n--1}
maxObj ← −∞, argMaxOrder ← null
for r = 0, 1, ..., k − 1 do
    initialize empty output sequence o
    append b_0, ..., b_{r−1}; g_{n+−1} to o
    merge b_r, ..., b_{n−−1} and g_0, ..., g_{n+−2} in decreasing score order to o
    obj(o) ← 1 − 1/(1+r) + w^⊤ φ_mrr(x, o)
    if obj(o) > maxObj then
        maxObj ← obj(o)
        argMaxOrder ← o
build remaining output sequence o with r ≥ k and Δ_MRR = 1 as described in text
if obj(o) = 1 + w^⊤ φ_mrr(x, o) > maxObj then
    maxObj ← obj(o)
    argMaxOrder ← o
return optimal order o for generating new constraint
```

Figure 3: SVMmrr pseudocode for one query. In an implementation we do not need to materialize o.

Fact 3.5. *With ϕ_{mrr} and Δ_{MRR}, (2) can be solved for a query q in time $O(n_q \log n_q + k^2 + k \log n_q)$.*

Proof. With ϕ_{mrr} and Δ_{MRR}, instead of using Fact 3.2, we collect all solutions y for the objective $w^\top \phi_{\text{mrr}}(x, y) + \Delta_{\text{MRR}}(y^*, y)$ into clusters, each having a common value of $\Delta_{\text{MRR}}(y^*, y)$.

Note that $\Delta_{\text{MRR}}(y^*, y)$ can only take values from the set $\{0, 1/k, 2/k, \ldots, 1\}$, so we can afford to first optimize the objective within each cluster and take the best of these $k+1$ solutions.

Consider all solutions y with $\Delta_{\text{MRR}}(y^*, y) = 1 - 1/(1+r)$, i.e., the MRR of ordering y is $1/(1+r)$ $(0 \le r < k)$ because the first good document is at position r (beginning at 0). Inspecting ϕ_{mrr} in (5), it is easy to see that within this cluster of solutions, the y that maximizes $w^\top \phi_{\text{mrr}}(x, y)$ is the one that fills ranks $0, \ldots, r-1$ with bad documents having the largest scores, and then places the good document with the smallest score at rank r. What documents are placed after the first good document is immaterial to ϕ_{mrr}, and therefore we can save the effort.

The last cluster of orderings is where there is no good document in any of ranks $0, \ldots, k-1$ and the MRR is 0 and $\Delta_{\text{MRR}} = 1$. In this case, clearly the k bad documents with the largest scores should occupy ranks $0, \ldots, k-1$. Now consider the good document x_g with the smallest score s_g. We should now place all bad documents with score larger than s_g, after which we place x_g. Again, how other documents are placed after x_g does not matter. □

3.7 SVMcombo: Multicriteria ranking

Conventional wisdom underlying much work on learning to rank is that it is better to train for the loss function on which the system will be evaluated. As we have argued in Section 1.2, search systems typically face a heterogeneous workload. It may not be advisable to ultra-optimize a ranking system toward one criterion. Moreover, our experiments (Section 4.5) suggest that a test criterion is not reliably optimized by training with the associated loss function.

A related question of theoretical interest is, *must* one necessarily sacrifice accuracy on one criterion to gain accuracy in another, or are the major criteria (AUC, MAP, MRR and NDCG) sufficiently related that there can be a common model serving them all reasonably well?

Once a model w is trained by optimizing (1), during testing, given x_q we return $f(x_q, w) = \arg\max_y w^\top \phi(x_q, y)$. Define the following empirical risk as

$$R(w, \Delta) = \tfrac{1}{|Q|} \textstyle\sum_q \Delta(y_q^*, f(w, x_q)).$$

In presence of multiple kinds of loss functions $\Delta_l, l = 1, \ldots, L$, we can modify learning problem (1) in at least two ways.

Shared slacks: We define an aggregate loss $\Delta(y, y') = \max_l \Delta_l(y, y')$, and then assert the same constraint as in (1). This is done by simply asserting more constraints, on behalf of each Δ_l:

$$\forall l, \forall q, \forall y \neq y_q^* : w^\top \delta\phi_{x_q}(y_q^*, y) \ge \Delta_l(y_q^*, y) - \xi_q.$$

At optimality, we can see that $\frac{1}{|Q|} \sum_q \xi_q \ge R(w, \max_l \Delta_l)$.

Separate slacks: The other option is to aggregate the empirical risk, as $\sum_l R(w, \Delta_l)$, in which case, we have to declare separate slacks ξ_q^l for each query q and loss type l. These slacks have different "units" and should be combined as $(1/|Q|) \sum_l C_l \sum_q \xi_q^l$. For simplicity we set all $C_l = C$; learning C_ls is left for future work.

$$\arg \min_{w; \xi \ge \vec{0}} \tfrac{1}{2} w^\top w + \tfrac{C}{|Q|} \sum_q \sum_l \xi_q^l \quad \text{s.t.}$$

$$\forall l, q, \forall y \neq y_q^* : w^\top \delta\phi(y_q^*, y; x_q) \ge \Delta_l(y_q^*, y) - \xi_q^l.$$

As before, we can see that $\frac{1}{|Q|} \sum_q \xi_q^l \ge R(w, \Delta_l)$, therefore $\sum_l \frac{1}{|Q|} \sum_q \xi_q^l \ge \sum_l R(w, \Delta_l)$. Because $\max_l \Delta_l(y, y') \le \sum_l \Delta_l(y, y')$, we have

$$\textstyle\sum_l R(w, \Delta_l) \ge R(w, \max_l \Delta_l).$$

Therefore, if there is a mix of queries that benefit from different loss functions, such as some navigational queries that are served well by MRR and some exploratory queries served better by NDCG, separate slacks may perform better, which is indeed what we found in experiments.

3.8 Review of McRank

For completeness, we compare the structured learning approaches (SVMauc, SVMmap, SVMmrr, SVMndcg, DORM, SVMcombo) against McRank, which is among the best of the lower half of Figure 1. Li *et al.* [10] have found McRank to be generally better than LambdaRank [6] and FRank [21]; SoftRank is comparable to LambdaRank [8] and both are generally better than RankNet [3].

McRank uses a boosted ensemble of regression trees [22] to learn a non-linear itemwise model for $\Pr(z|x_{qi})$, i.e., the probability of falling into each relevance bucket (in this paper we have mostly considered $z \in \{0, 1\}$). The interesting twist is that, instead of assigning relevance $\arg\max_z \Pr(z|x_{qi})$, McRank assigns a score $\sum_z z \Pr(z|x_{qi})$ to the ith document responding to query q, and then sorts the documents by decreasing score. Note that McRank has no direct hold on true loss functions like MAP, MRR or NDCG. We implemented the boosting code in Java, taking advantage of the WEKA [23] REPTree implementation.

4. EXPERIMENTS

4.1 Data preparation

Inside the LETOR distribution [16] there are three data sets, OHSUMED (106 queries, 11303 bad documents, 4837

good documents), TD2003 (50 queries, 48655 bad documents, 516 good documents) and TD2004 (75 queries, 73726 bad documents, 444 good documents). Each document has about 25–45 numeric attributes. These are scaled to $[0, 1]$ within each query as specified in the LETOR distribution. We observed that LETOR has many queries for which the same feature vector is marked as both good and bad. In addition there are feature vectors with all elements exactly equal to zero. A robust training algorithm is expected to take these in stride, but test accuracy falls prey to breaking score ties arbitrary. This can give very unstable results especially given the modest size of LETOR. Therefore we eliminated all-zero feature vectors and good and bad vectors whose cosine similarity was above 0.99. Although this further reduced the number of queries, the comparisons became much more reliable. In our other data set obtained from Yue *et al.* [9], TREC 2000 has 50 queries, 218766 bad documents and 2120 good documents. TREC 2001 has 50 queries, 203507 bad documents and 2892 good documents.

4.2 ϕ, Δ and ease of optimization

Obviously, formulating a structured learning approach to ranking does not guarantee healthy optimization. The purpose of this section is to highlight that structured ranking algorithms suffer from various degrees of distress during training, and offer some analysis.

Average slack vs. C: Figure 4 shows, for different algorithms, the value of $(1/|Q|)\sum_q \xi_q$ (an upper bound on the training loss) when the optimizer terminates, against C. DORM, SVMmrr, and SVMauc show the most robust reduction in average slack with increasing C. Note that DORM and SVMmrr use custom feature maps. Also, ϕ_{po} is ideally suited for Δ_{AUC}. When constraints are added in SVMauc, each term $(1/n^+n^-)y_{gb}(s_g - s_b)$ on the lhs $w^\top \delta\phi$ is matched to one term $(1/n^+n^-)(1-y_{gb})$ on the rhs Δ_{AUC}.

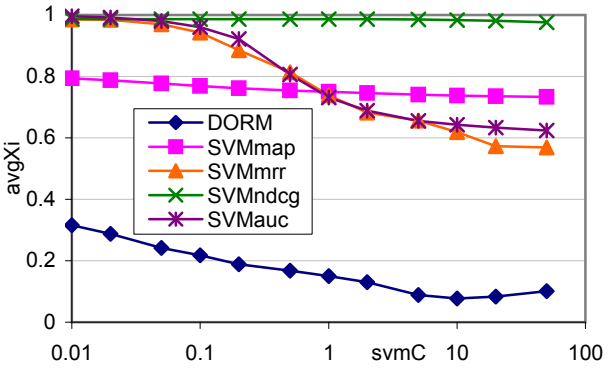

Figure 4: Different behavior of $(1/|Q|)\sum_q \xi_q$ at termination for different algorithms as C increases.

SVMmap and SVMndcg have a harder time. We conjecture that this is caused by a mismatch between Δ_{MAP}, Δ_{NDCG}, and ϕ_{po}. The lhs of constraints now consist of sums of (variable numbers of) score differences, while the rhs have a much more granular loss Δ not sufficiently sensitive to the variation on the lhs.

Training objective: Let $\text{obj}(w = \vec{0})$ be the value of the objective in (1) for $w = \vec{0}$, and obj be the optimized training objective. For $C = 0$, $w = \vec{0}$ is the optimum. A plot of $\text{obj}/\text{obj}(w = \vec{0})$ against C (Figure 5) is an indication of how well w is adapting to increasing C. The results are related to Figure 4: DORM, SVMauc and SVMmrr adapt best, while SVMmap and SVMndcg stay close to $w = \vec{0}$, but, luckily, not quite at $w = \vec{0}$—see Figure 6.

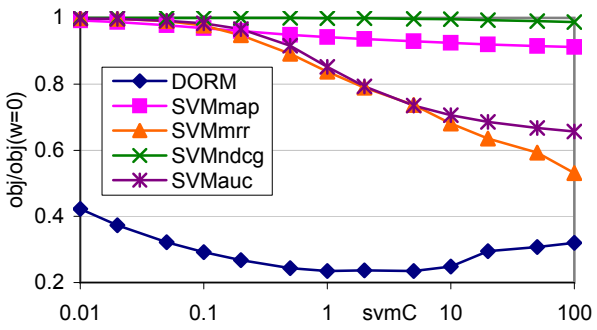

Figure 5: obj/obj$(w = \vec{0})$ for different algorithms as C increases.

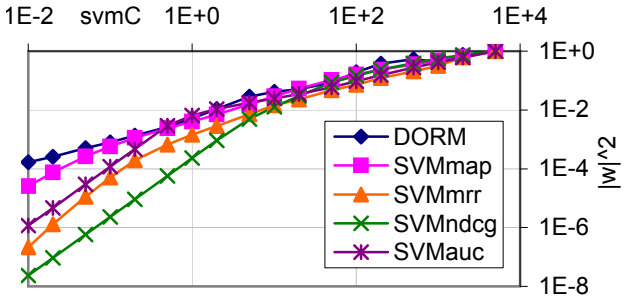

Figure 6: Different behavior of $|w|$ at termination for different algorithms as C increases. Rightmost point arbitrarily scaled to 1 for comparison.

4.3 SVMmrr evaluation

Indirect support for our conjecture comes from Figure 7. It shows the benefits of using ϕ_{mrr} instead of ϕ_{po} for optimizing MRR. Over all data sets, there is a consistent large gain in MRR when ϕ_{mrr} is used, compared to ϕ_{po}. However, from Figure 9, we see that training with some criterion other than MRR is almost always best for test MRR scores. Specifically, SVMcombo almost always beats SVMmrr. Similar to Taylor *et al.* [8], we conjecture that this is because the "true" loss function Δ_{MRR} and ϕ_{MRR} are losing information from multiple good documents. SVMcombo "hedges the bet" in a principled manner.

4.4 SVMndcg scalability and accuracy

The three data sets inside the LETOR distribution have different sizes, which makes it easy to do scaling experiments. OHSUMED has a total of 16140 documents, TD2003 has 49171, and TD2004 has 74170; this is roughly 1:3:4.6. OHSUMED has 106 queries, TD2003 has 50 and TD2004 has 75.

In these experiments we gave DORM the benefit of a sparse $A(\cdot)$ decay function decaying to zero after rank 30, which was what was required to approach or match the accuracy of SVMndcg. From Figure 8 we see that the total time taken by DORM is substantially larger than SVMndcg, and scales much more steeply than 1:3:4.6, which is expected from the nature of the assignment problem.

In contrast, the total time taken by SVMndcg is much smaller. For TD2004, SVMndcg took only 19 seconds while

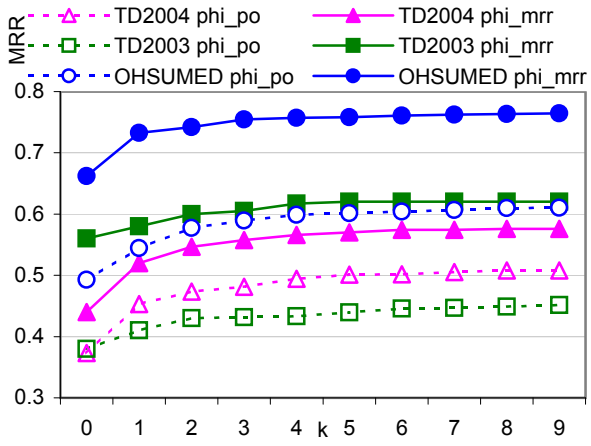

Figure 7: Comparison of ϕ_{mrr} against ϕ_{po} for the MRR criterion.

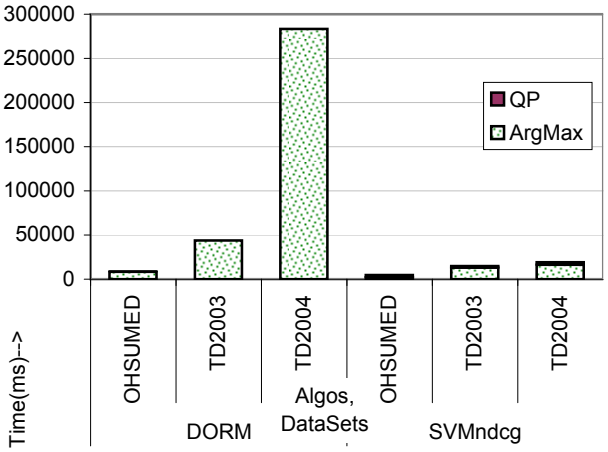

Figure 8: Breakdown of DORM time into assignment [20] and quadratic optimization; compared with breakdown of SVMndcg time into time taken by the 'argmax' calculations (Figure 2) and quadratic optimization.

DORM needed 283 seconds. Obviously the gap will only grow with increasing data sizes. Proprietary training data mentioned in the literature [10] have millions of documents.

Also noteworthy is the very small time taken in the QP optimizer (invisible for DORM, barely visible for SVMNDCG). We used a very recent and fast implementation of LaRank [24]. This shows that solving the 'argmax' problem (2) quickly for large data sets is important, because the QP solver is not the bottleneck.

Figure 9 compares test NDCG at rank 10 for different training criteria. SVMNDCG and DORM come out about even, but SVMNDCG-NC is consistently better than DORM.

4.5 SVMCOMBO evaluation

At this point, it is of interest to complete a table where each row corresponds to a training criterion, and each column is a test criterion. Conventional wisdom suggests that the trainer that gives the best test NDCG will be the one that uses Δ_{NDCG} and so on. Figure 9 shows that this is rarely the case! Specifically,

- Δ_{MRR} is never best for test MRR.
- SVMNDCG, which uses the "true" Δ_{NDCG} loss, consistently loses to SVMNDCG-NC, which uses only an approximation to Δ_{NDCG}.
- Often, SVMMAP does not give the best test MAP.
- SVMCOMBO (using MAP, NDCG, NDCG-NC, and MRR components variously) is most consistently among the top two performers in each column. Sometimes SVMCOMBO's accuracy is greater than any of its constituents.
- Despite spending $O(n^3)$ time in optimization (2), DORM never tops the chart in any column.
- Similarly, MCRANK rarely wins over SVMCOMBO and SVMMAP (two of nine columns).

4.6 Comparison with MCRANK

We finally consider the training speed and test accuracy of MCRANK. Our WEKA-based implementation exceeded 2 GB of RAM for each of TREC 2000 and TREC 2001, and was unreasonably slow. So we limit our study to the LETOR data. In only two of the nine columns pertaining to LETOR does MCRANK show substantial advantage; in the remaining seven, one of the list-wise structured learning approaches is better.

MCRANK's occasional lead comes at a steep RAM and CPU cost. The CPU time is dominated by the time to induce CART [22] style regression trees. The number of rounds of boosting was set between 1500 and 2000 by Li *et al.* [10]; we found this too slow (corroborated elsewhere [19]) and also unnecessary for accuracy. On LETOR more than 30–40 rounds sometimes hurt accuracy, so we set the number of boosting rounds to 30; this only tips the scales against us wrt performance. Even so, we find in Figure 10 that MCRANK can be computationally more expensive that structured learners by two orders of magnitude.

5. CONCLUSION

Using the structured learning framework, we proposed novel, efficient algorithms for learning to rank under the MRR and NDCG criteria. The new algorithms are comparable to known techniques in terms of accuracy but are much faster. We then presented SVMCOMBO, a technique to optimize for multiple ranking criteria. SVMCOMBO may be preferable for real-life search systems that serve a heterogeneous mix of queries. Our exploration revealed that structured ranking often suffers from a mismatch between the feature map $\phi(x, y)$ and the loss function $\Delta(y, y')$. Designing loss-specific feature maps for better training optimization remains a central problem that merits further investigation.

Acknowledgment: Thanks to Thorsten Joachims and Yisong Yue for the TREC 2000 and TREC 2001 data, to Sundar Vishwanathan for discussions, and to Sunita Sarawagi for discussions and the LARank implementation.

Dataset	MCRANK tree	MCRANK boost	MCRANK total	SVMNDCG	SVMMRR
OHSUMED	1034	67	1102	4.8	30.6
TD2003	9730	383	10113	14.9	125
TD2004	8760	548	9308	19.1	148

Figure 10: Break-up of McRank running time and comparison with SVMndcg and SVMmrr (times in seconds).

	OHSUMED			TD2003			TD2004			TREC2000			TREC2001		
	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP
AUC	.799	.635	.582	.510	.349	.256	.639	.501	.420	.607	.448	.267	.632	.441	.264
MAP	.808	**.642**	**.586**	.618	.411	*.314*	.614	.496	*.412*	**.696**	**.469**	**.277**	.636	**.450**	**.272**
NDCG	.790	*.636*	.581	.587	*.372*	.302	.631	*.457*	.374	.517	*.323*	.175	.608	*.356*	.171
NDCG-NC	**.818**	.640	.582	.595	.404	.306	.611	.486	.404	.685	.455	.265	.624	.443	.264
MRR	*.795*	.623	.570	*.628*	.405	.330	*.629*	.441	.383	*.670*	.410	.244	*.643*	.426	.230
COMBO	.813	.635	.578	**.667**	**.434**	**.345**	**.647**	.458	.384	.695	.465	**.277**	**.647**	.449	**.272**
DORM	.807	.637	.583	.587	.362	.290	.474	.340	.297	.662	.413	.243	.621	.435	.250
McRank	.701	.565	.527	.650	.403	.232	.588	**.529**	**.453**						

Figure 9: Test accuracy vs. training loss function used. Contrary to conventional wisdom, the best test accuracy on a given loss function may be the result of training on some other loss function. SVMcombo and SVMmap are among the most robust trainers.

6. REFERENCES

[1] R. Herbrich, T. Graepel, and K. Obermayer, "Support vector learning for ordinal regression," in *International Conference on Artificial Neural Networks*, 1999, pp. 97–102. http://www.research.microsoft.com/~rherb/papers/hergraeober99b.ps.gz

[2] T. Joachims, "Optimizing search engines using clickthrough data," in *SIGKDD Conference*. ACM, 2002. http://www.cs.cornell.edu/People/tj/publications/joachims_02c.pdf

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005. http://research.microsoft.com/~cburges/papers/ICML_ranking.pdf

[4] T. Joachims, "A support vector method for multivariate performance measures," in *ICML*, 2005, pp. 377–384. http://www.machinelearning.org/proceedings/icml2005/papers/048_ASupport_Joachims.pdf

[5] ——, "Training linear SVMs in linear time," in *SIGKDD Conference*, 2006, pp. 217–226. http://www.cs.cornell.edu/people/tj/publications/joachims_06a.pdf

[6] C. J. C. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *NIPS*, 2006. http://research.microsoft.com/~cburges/papers/LambdaRank.pdf

[7] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136. http://www.machinelearning.org/proceedings/icml2007/papers/139.pdf

[8] M. Taylor, J. Guiver, S. Robertson, and T. Minka, "SoftRank: Optimising non-smooth rank metrics," in *WSDM*. ACM, 2008. http://research.microsoft.com/~joguiver/sigir07LetorSoftRankCam.pdf

[9] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *SIGIR Conference*, 2007. http://www.cs.cornell.edu/People/tj/publications/yue_etal_07a.pdf

[10] P. Li, C. J. C. Burges, and Q. Wu, "McRank: Learning to rank using multiple classification and gradient boosting," in *NIPS*, 2007, pp. 845–852. http://books.nips.cc/papers/files/nips20/NIPS2007_0845.pdf

[11] 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.

[12] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *JMLR*, vol. 6, no. Sep, pp. 1453–1484, 2005. http://ttic.uchicago.edu/~altun/pubs/TsoJoaHofAlt-JMLR.pdf

[13] E. Voorhees, "Overview of the TREC 2001 question answering track," in *The Tenth Text REtrieval Conference*, ser. NIST Special Publication, vol. 500-250, 2001, pp. 42–51. http://trec.nist.gov/pubs/trec10/t10_proceedings.html

[14] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *SIGIR Conference*, 2000, pp. 41–48. http://www.info.uta.fi/tutkimus/fire/archive/KJJKSIGIR00.pdf

[15] E. Snelson and J. Guiver, "SoftRank with gaussian processes," in *NIPS 2007 Workshop on Machine Learning for Web Search*, 2007. http://research.microsoft.com/CONFERENCES/NIPS07/papers/gprank.pdf

[16] 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. http://research.microsoft.com/users/LETOR/

[17] O. Chapelle, Q. Le, and A. Smola, "Large margin optimization of ranking measures," in *NIPS 2007 Workshop on Machine Learning for Web Search*, 2007. http://research.microsoft.com/CONFERENCES/NIPS07/papers/ranking.pdf

[18] Q. V. Le and A. J. Smola, "Direct optimization of ranking measures," Feb. 2008, arXiv:0704.3359v1. http://arxiv.org/pdf/0704.3359

[19] C. Burges, "Learning to rank for Web search: Some new directions," Keynote talk at SIGIR Ranking Workshop, July 2007. http://research.microsoft.com/users/LR4IR-2007/LearningToRankKeynote_Burges.pdf

[20] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*. Prentice Hall, 1993.

[21] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "FRank: A ranking method with fidelity loss," in *SIGIR Conference*, 2007, pp. 383–390. http://portal.acm.org/ft_gateway.cfm?id=1277808&type=pdf

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth & Brooks/Cole, 1984, iSBN: 0-534-98054-6.

[23] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005. http://www.cs.waikato.ac.nz/ml/weka/

[24] A. Bordes, L. Bottou, P. Gallinari, and J. Weston, "Solving multiclass support vector machines with LaRank," in *ICML*. ACM, 2007, pp. 89–96. http://www.machinelearning.org/proceedings/icml2007/papers/381.pdf