Large margin optimization of ranking measures

Olivier Chapelle
Yahoo! Research, Santa Clara
chap@yahoo-inc.com

Quoc Le
NICTA, Canberra
quoc.le@nicta.com.au

Alex Smola
NICTA, Canberra
alex.smola@nicta.com.au

Abstract

Most ranking algorithms, such as pairwise ranking, are based on the optimization of standard loss functions, but the quality measure to test web page rankers is often different. We present an algorithm which aims at optimizing directly one of the popular measures, the Normalized Discounted Cumulative Gain. It is based on the framework of structured output learning, where in our case the input corresponds to a set of documents and the output is a ranking. The algorithm yields improved accuracies on several public and commercial ranking datasets.

1 Introduction

Web page ranking has traditionally been based on a hand designed ranking function such as BM25 [12]. However ranking is now considered as a supervised learning problem and several machine learning algorithms have been applied to it [1, 3, 6].

Traditional approaches in learning to rank optimize uniform ranking measures such as number of mis-ordered pairs [9]. However, it is often the case that the users may be more interested in the most relevant items (first page) and ignore other items. Thus it is more appropriate for a ranker to spend effort and get the topmost items right. This fact is reflected in various ranking measures such as the popular Normalized Discounted Cumulative Gain (NDCG) score [10].

While recent attempts have been made to optimize this measure [2], we present here a method which directly optimizes the NDCG. This is made possible by viewing ranking as trying to learn an ordering of the data. For this purpose, we use the framework of learning with structure output [15], where in our case the output is a ranking, i.e. a permutation of the documents. An upper-bound on the empirical loss is minimized and the loss is defined as the difference in NDCG between a given ranking and the optimal one.

We formalize the learning to rank problem as follows. Pairs of (query, document) are available and for each pair a grade measuring the relevance of the document to that query has been assigned by a human editor. We thus have a training set $\{(\mathbf{x}_{qi}, g_{qi})\}$, where $q = 1, \dots n$ indexes the queries on the training set, $i = 1, \dots, m_q$ indexes the documents with each query, and $\mathbf{x}_{qi}$ is a vector describing the i-th (query, document) pair corresponding to the query q. g_{qi} is the categorical grade assigned to that pair. For instance, grades with 5 levels of relevance are typically encoded as integers from 0 for bad to 4 for perfect.

We use the framework of maximum margin for structured output learning [15] to learn a function that takes as as input a set of documents associated to a given query, $\mathbf{x}_q := (\mathbf{x}_{q1} \dots \mathbf{x}_{qm_q})$, and outputs a ranking y_q (i.e. a permutation of $\{1, \dots, m_q\}$) with $y_{qi} = r$ meaning that the i-th document has rank r.

Even though it is possible to optimize other ranking measures, we focus in this paper on the NDCG which is defined for query q as

$$\mathrm{NDCG}_k(y, q) := \frac{1}{N_q} \sum_{i=1}^{k} D(y_i)\varphi(g_{qi}), \tag{1}$$

where D is the discount function $D(r) = \frac{1}{\log_2(1+r)}$, φ is an increasing function often chosen to be $\varphi(g) = 2^g - 1$ and N_q is a normalization constant such that the optimal ranking based on the values of g_{qi} has score 1. k is called a truncation or threshold level. Intuitively the NDCG is a ranking evaluation criterion that puts strong emphasis at the topmost items, and between these items, there is a small decaying factor.

The paper is organized as follows. Section 2 shows how structured output learning can be applied to ranking. The details of the optimization are given in section 3 and in particular how the combinatorial optimization problem turns out to be a linear assignment problem which can be solved efficiently. Section 4 discusses an extension of the algorithm where the convex upper bound on the loss is tightened to a non-convex one. Experimental results are presented in section 5 before finishing with some possible extensions including the non-linear case.

2 Ranking as structured output

The first ingredient of structured output learning is the design of a *joint feature map* $\Psi(\mathbf{x}_q, y_q)$ from a query and a ranking to $\mathbb{R}^d$. We will see shortly how to define this function. For a given weight vector $\mathbf{w}$, the predicted ranking associated to a test query $\mathbf{x}_q$ is [15]

$$\arg\max_y \mathbf{w}^\top \Psi(\mathbf{x}_q, y). \tag{2}$$

We now propose the following choice for Ψ:

$$\Psi(\mathbf{x}_q, y_q) = \sum_i \mathbf{x}_{qi} A(y_{qi}), \tag{3}$$

where $A : \mathbb{N} \to \mathbb{R}$ is a user defined non-increasing function. The reason for choosing such a function Ψ is that one can rewrite (2) as

$$\arg\max_y \sum_i A(y_i)\ \mathbf{w}^\top \mathbf{x}_{qi}. \tag{4}$$

Finding the arg max in (4) over all possible rankings y is straightforward: it is given by the ranking (in decreasing order) of the "scores" $\mathbf{w}^\top \mathbf{x}_{qi}$. For this reason the linear function $\mathbf{x} \mapsto \mathbf{w}^\top \mathbf{x}$ can be interpreted as a relevance function: if a (query,document) $\mathbf{x}$ has a large value for $\mathbf{w}^\top \mathbf{x}$, it means that this document is relevant for this query. So it turns out that at test time our algorithm is not different from any other standard ranking algorithm: it simply ranks the documents according to the relevance function induced by $\mathbf{w}$. Note that we only consider linear functions in this paper, but we will discuss non-linear extension at the end.

When optimizing the NDCG_k, only the first k elements of the ranking matter. So we decide to consider only functions A such that $A(r) = 0$ for $r > k$. Typically, A should be optimized in the model selection part, but in the rest of this paper, we simply take $A(r) = \max(k + 1 - r, 0)$. Also we ignore the rest of ranking after position k and consider as equivalent two rankings which are identical on the first k positions.

For a given query $\mathbf{x}_q$ in the training set and the grades of the corresponding pages, we associate an output y_q which is the perfect ranking for that query. It is often the case that this y_q is not unique and we simply take of one of them at random. The training set has thus been converted to a set $\{(\mathbf{x}_q, y_q)\}$ of queries and ranking and the original grades g_{qi} will only be used indirectly in the loss function (1). Given this training set, we would like

to find a vector $\mathbf{w}$ such that the rule (2) predicts the correct rankings on the training set. This can be written as:

$$\forall q, \forall y \neq y_q, \mathbf{w}^\top \Psi(\mathbf{x}_q, y_q) - \mathbf{w}^\top \Psi(\mathbf{x}_q, y) > 0.$$

As for SVMs, we introduce a margin and slack variables when the training data cannot be learned exactly to arrive at the following optimization problem [15, 13],

$$\min_{\mathbf{w}, \xi_q} \quad \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w} + \sum_q \xi_q \tag{5}$$

under constraints:

$$\forall q, \forall y \neq y_q, \mathbf{w}^\top \Psi(\mathbf{x}_q, y_q) - \mathbf{w}^\top \Psi(\mathbf{x}_q, y) \geq \Delta_q(y) - \xi_q, \tag{6}$$

where $\Delta_q(y)$ is the loss incurred when predicting y instead of the correct output y_q. The intuitive reason for this is that we want to penalize more heavily the very bad rankings. In our case, $\Delta_q(y)$ is the difference between the NDCG with the perfect ranking y_q and the NDCG obtained with the ranking y, i.e. $\Delta_q(y) = 1 - \mathrm{NDCG}_k(y, q)$ (see definition (1)).

It is also interesting to note that ξ_q is an upper bound on the loss incurred on that query [15]. Indeed let us take a query q for which there is a prediction error and let $\hat{y_q} := \arg\max_y \mathbf{w}^\top \Psi(\mathbf{x}_q, y)$ be the predicted ranking. Equation (6) applied to $y = \hat{y_q}$ gives

$$\xi_q \geq \Delta_q(\hat{y_q}) + \underbrace{\mathbf{w}^\top \Psi(\mathbf{x}_q, \hat{y_q}) - \mathbf{w}^\top \Psi(\mathbf{x}_q, y_q)}_{\geq 0}.$$

So $\xi_q \geq \Delta_q(\hat{y_q})$ and minimizing $\sum \xi_q$ can be seen as maximizing a lower bound on the NDCG.

3 Optimization

The optimization of (5) is not straightforward because there is an exponential number of constraints (6). We will explore different possibilities for solving this optimization problem. But before, let us concentrate on a step which is common for all algorithms.

3.1 Assignment problem

From (6), the optimal ξ_q is given by:

$$\xi_q = \max_{y \neq y_q} \ \Delta_q(y) - \mathbf{w}^\top \Psi(\mathbf{x}_q, y_q) + \mathbf{w}^\top \Psi(\mathbf{x}_q, y). \tag{7}$$

All the optimization algorithms descried below rely on being able to compute the $\arg\max$ in (7).

Since $\Delta_q(y) = 1 - \mathrm{NDCG}_k(y, q)$, taking into account (1) and (3), we get the following problem after discarding the terms independent of y in (7):

$$\arg\max_y \ \sum_{i=1}^{k} A(y_i) \mathbf{w}^\top \mathbf{x}_{qi} - \sum_{i=1}^{k} \varphi(g_{qi}) D(y_i). \tag{8}$$

Problem (8) is a linear assignment which can be solved efficiently using for instance the Kuhn-Munkres algorithm [11]. Also we are only searching for the first k assignments and the algorithm is even faster for small k.

Let us now go into the details of the different algorithms.

3.2 Cutting plane

As in [15], one can use a cutting plane method which boils down to iterating between finding the $\arg\max$ in (8) and solving the Quadratic Program (QP) (5); see algorithm 1.

Algorithm 1 Cutting plane method

1: $\mathbf{w} \leftarrow 0$, $\xi_q \leftarrow 0$, $S_q \leftarrow \emptyset$, $\quad i = 1, \ldots, n_q$.
2: **repeat**
3: **for** $q = 1 \ldots n_q$ **do**
4: $\tilde{y} \leftarrow \arg\max$ of (8)
5: **if** (6) is violated with $y = \tilde{y}$ **then**
6: $S_q \leftarrow S_q \cup \{\tilde{y}\}$.
7: Optimize (5) under subset of constraints (6) $\forall q, \ \forall y \in S_q$.
8: **end if**
9: **end for**
10: **until** No S_q has changed.

There are some possible variations around this algorithm. The first one is that the QP does not necessarily need to be resolved after a new constraint is added (step 7 of algorithm 1). Instead, it could be solved after collecting the constraints corresponding to all the queries (i.e. at the end of the for loop). Between these two extreme cases, an intermediate solution is to solve the QP after some fixed number of new constraints have been added. It is also possible to solve the QP only partially, especially in the dual.

We now give the dual formulation for solving (5). For $i = 1, \ldots, \sum_q |S_q|$, we can write the set of constraints as:

$$\forall q, \forall i \in U_q, \ \mathbf{w}^\top \mathbf{v}_i \geq b_i - \xi_q, \tag{9}$$

where $\mathbf{v}_i$ is of the form $\Psi(\mathbf{x}_q, \tilde{y}) - \Psi(\mathbf{x}_q, y_q)$ and U_q is the set of indices corresponding to query q. Note that with these definitions, $|U_q| = |S_q|$.

Introducing the Gram matrix K, with $K_{ij} = \mathbf{v}_i^\top \mathbf{v}_j$, we obtain the following optimization problem:

$$\max_{\boldsymbol{\alpha}} \ \mathbf{b}^\top \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} \tag{10}$$

under constraints

$$\boldsymbol{\alpha} \geq 0 \ \text{ and } \ \forall q, \sum_{i \in U_q} \alpha_i \leq C. \tag{11}$$

It turns out that this optimization problem is the computational bottleneck and the assignment problem (8) is comparatively very fast. There are several "tricks" to have an efficient optimization:

1. Use of "hot-start": the $\boldsymbol{\alpha}$ from the previous iteration is the starting point for the next iteration.
2. After solving the QP, it might be a good idea to get rid off the inactive constraints (i.e. the ones for which $\alpha_i = 0$). This avoids dragging inactive constraints during the entire optimization. A discarded constraint will be "reactivated" if it is picked up by (8). In situation where the cost of (8) dominates the cost of solving (10), this might not be a good idea.
3. In the same line, (10) does not need to be solved exactly. We use an active set method where where once a variable α_i becomes inactive, it is never checked again for inclusion. It will however be checked for inclusion in the outer loop through (8).
4. When a new constraint is added, one can simply optimize the single α_i corresponding to that constraint. The overall optimization on the entire vector $\boldsymbol{\alpha}$ is then only done in the outer loop.

3.3 Unconstrained optimization

We now look at methods which optimize the unconstrained formulation of the problem obtained by combining equations (5) and (7):

$$\min_{\mathbf{w}} \ \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w} + \sum_q \max_{y \neq y_q} \ \Delta_q(y) - \mathbf{w}^\top \Psi(\mathbf{x}_q, y_q) + \mathbf{w}^\top \Psi(\mathbf{x}_q, y). \tag{12}$$

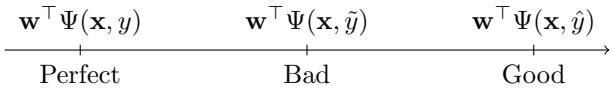

Figure 1: Let $(\mathbf{x}, y)$ be a training example. Imagine a situation where there is no $\mathbf{w}$ learning the data well, i.e. such that $\mathbf{w}^\top \Psi(\mathbf{x}, y)$ is large, but there is a $\mathbf{w}$ for which the prediction $\hat{y}$ on $\mathbf{x}$ is good (but maybe not perfect). In that case, we would like to pay a small loss for that training example. However, in the standard formulation of [15], the loss might be very large if there is a "bad" prediction $\tilde{y}$ whose output is larger than the perfect one y.

The function in equation (12) is nondifferentiable. We now describe two commons methods for *nonsmooth optimization*; see [5, chapter 14] for more details.

3.3.1 Steepest subgradient minimization

This is the analogous of steepest descent except that at a non-differentiable point a subgradient is used. A non-differentiable point is such that there is a tie in of the max of (12). A subgradient is obtained by breaking it at random. The next point is found by a standard line search with backtracking. Instead of steepest descent, a conjugate gradient technique can also be used [16].

3.3.2 Bundle minimization

This is an iterative method which works as follows [14]. Let $\mathbf{w}_k$ be the current value of $\mathbf{w}$ at each iteration k. Consider the rhs of (12) and compute its value f_k as well as a subgradient $\mathbf{g}_k$. Because of convexity, we have that

$$\frac{\lambda}{2}\mathbf{w}^\top \mathbf{w} + \max_k \left((\mathbf{w} - \mathbf{w}_k)^\top \mathbf{g}_k + f_k \right) \tag{13}$$

is a lower bound on (12). Equation is minimized using a similar dual formulation as in section 3.2 and the new minimum is $\mathbf{w}_{k+1}$.

4 Tighter upper bound

On some datasets, we found that the optimal solution of (12) is simply $\mathbf{w} = 0$. The problem is related to underfitting; see figure 1 for an intuitive explanation.

One way to fix this problem is the following. Instead of trying to enforce that the target output y is larger than the other outputs, we will try to enforce it for another target $\hat{y}$, which is not far from y. More precisely, we replace the original loss,

$$\max_{\tilde{y}} \ \Delta(\tilde{y}) - \mathbf{w}^\top \Psi(\mathbf{x}, y) + \mathbf{w}^\top \Psi(\mathbf{x}, \tilde{y})$$

by

$$\begin{aligned} \min_{\hat{y}} \quad & \max_{\tilde{y}} \ \Delta(\tilde{y}) - \mathbf{w}^\top \Psi(\mathbf{x}, \hat{y}) + \mathbf{w}^\top \Psi(\mathbf{x}, \tilde{y}) \\ = \quad & \max_{\tilde{y}} (\Delta(\tilde{y}) + \mathbf{w}^\top \Psi(\mathbf{x}, \tilde{y})) - \max_{\hat{y}} \ \mathbf{w}^\top \Psi(\mathbf{x}, \hat{y}). \end{aligned} \tag{14}$$

In the scenario pictured in figure 1, that would lead to a much smaller loss. The maximum over $\hat{y}$ is simply given by the prediction rule (2) at the current $\mathbf{w}$. Several comments about this new loss:

1. It is smaller than the original loss. This can be seen by taking $\hat{y} = y$.
2. It still an upper bound on the loss: take $\tilde{y} = \hat{y}$.
3. It is non-convex.

In summary, we have a tighter upper bound on the loss, but the convexity is lost. Because of the non-convexity, we simply minimized this loss by gradient descent as in section 3.3.1. Another possibility would have been to use the technique described in [8]

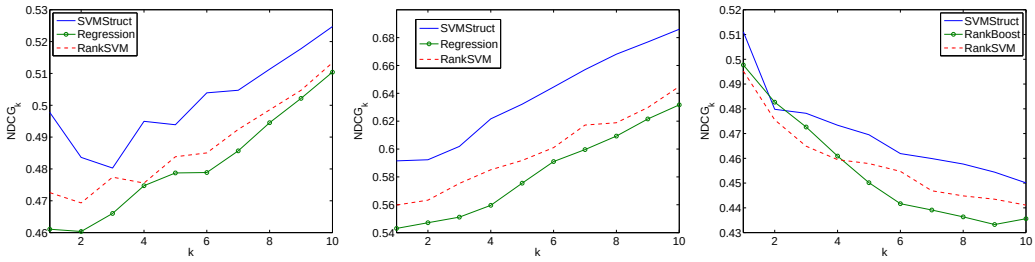

Figure 2: Results on the web search 1 dataset (left), web search 2 (center) and Ohsumed (right).

5 Experiments

We compare our proposed algorithm on three different datasets: the Ohsumed dataset, part of the Letor package[1] and two datasets from different web search engines. The reason for choosing Ohsumed over Trec (the other dataset part of Letor) is because Trec has less queries and has only 2 relevance levels and is thus less interesting from a ranking perspective.

In both cases, we compare the NDCG at different truncation levels with some baseline algorithms. Note that we only consider linear algorithms.

5.1 Web search 1

In this first experiment, we present results on a ranking dataset coming from a commercial search engine. It consists of about 1500 queries and 50k (query,url) pairs. For each of these pairs, a relevance grade has been assigned out of 5 possible relevance levels. Each (query,url) pair is made of several hundred features.

A fifth of the data is held out as a test set and a fifth as a validation set. The regularization parameter λ is tuned on this validation set. For each k from 1 to 10, the NDCG_k is independently optimized. The function A is chosen to be $A(r) = \max(k+1-r, 0)$. We also tried to take A to be the discount function D but the results were not as good.

Results are shown in figure 2. Even though there is no clear state-of-the-art ranking algorithm, we have identified based on the literature and our own experience two popular methods in web search ranking. They are the baseline methods that we are including for comparison. The first one simply consists in performing regression to fit the output relevance levels $0, 1, 2, 3, 4$. The second one, RankSVM [9], is based on pairwise classification: for a given query, if a page $\mathbf{x}_1$ is more (resp less) relevant than $\mathbf{x}_2$, then $\mathbf{x}_1 - \mathbf{x}_2$ is added to the training set with label +1 (resp -1). Then a standard linear SVM classifier is trained on this new training set.

The gain in performance is about 2%, which is considered very satisfactory in the web search ranking community. For each query, we add the NDCGs from $k = 1$ to 10 and perform a Wilcoxon signed-rank test. The p-value corresponding to the difference between our method and regression (resp RankSVM) is 3% (resp 7%).

From a computational point of view, the training is relatively fast. For the three different methods described in sections 3.2, 3.3.1 and 3.3.2, the training time is of the order of 15 minutes.

5.2 Web search 2

This second experiment is similar to the first one except that the data comes from another search engine. It has 1000 queries for training 1000 queries for test and 1000 queries for

[1] Available at `http://research.microsoft.com/users/tyliu/LETOR/`

validation. The number of documents per query is typically of the order of 50. It also has 5 levels of relevance. For that experiment, a different function A is used, $A(r) = \frac{1}{\sqrt{r}}$.

The results are shown in figure 2. Our algorithm clearly outperforms the other ones and the improvement is statistically significant (p-value $< 1\%$).

5.3 Ohsumed dataset

The Ohsumed dataset has 106 queries and we used the same 5 splits training / validation / test as provided in the Letor distribution. Each (query,document) pair has 25 features and 3 possible relevance scores.

On this dataset, the algorithm presented in section 3 (i.e. the minimization of (12)) returns the solution $\mathbf{w} = 0$ even for very small values of λ. We believe that it is because of the underfitting problem mentioned in section 4. To solve this problem, we instead perform a gradient descent optimization on the tighter non-convex upper bound (14). Since this optimization problem is non-convex, the starting point is important. We set it to $\mathbf{w}_0$, where $\mathbf{w}_0$ is found by regression on the grades. Also the regularizer is changed from $||\mathbf{w}||^2$ to $||\mathbf{w} - \mathbf{w}_0||^2$.

Finally we observed that optimizing the NDCG_k, as done above, gave unstable results in particular during the model selection phase. This is probably because the dataset if very small. Instead we optimize the NDCG_{10} which gives a more stable indicator of the quality of a solution.

Figure 2 shows the performance of our method compared to the two baseline algorithms, RankBoost and RankSVM. These results are included in the Letor distribution. Even though there is around 2% improvement in the NDCG, this difference is not statistically significant. This is again probably due to the fact that there are only 106 queries in this dataset.

6 Extensions

Non-linear functions There are several ways of achieving a non-linear extension:

"Kernel trick" Simply replace $\mathbf{x}_{qi}$ in (3) by $\Phi(\mathbf{x}_{qi})$ where Φ a mapping to high dimensional feature space such $K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$. The evaluation of one "kernel" entry in (10) involves a linear combination of kernel evaluations.

Non-linear architecture Starting from the unconstrained formulation (12), one can replace the dot products $\mathbf{w}^\top \mathbf{x}_{qi}$ by $f_{\mathbf{w}}(\mathbf{x}_{qi})$ where $f_{\mathbf{w}}$ could for instance be a neural network with weights $\mathbf{w}$. For a gradient optimization, we simply need that $f_{\mathbf{w}}(\mathbf{x}_{qi})$ is a differentiable with respect to $\mathbf{w}$. See [1] for ranking with neural networks.

Functional gradient boosting When the class of functions cannot be optimized by gradient descent (that is the case of decision trees for instance), one can still apply the gradient boosting framework of [7]. Indeed in this framework we only need to compute the gradient of the loss with respect to the function values. This is possible in (12) if we ignore the regularization term. The regularization would have to be imposed in the boosting algorithm, for instance by limiting the number of iterations and/or the number of leaves in a decision tree.

Rescaled slacks In [15], it was also proposed to rescale the slacks instead of rescaling the margin as in (6). This means changing the rhs of (6) to $1 - \frac{\xi_q}{\Delta_q(y)}$. The benefit of this formulation is that it does not waste ressources in trying to enforce a very large margin for bad rankings. The problem is that the arg max problem (8) becomes a quadratic assignment problem:

$$\arg\max_y \left(C_1 - \sum g_{qi} D(y_i)\right) \left(C_2 + \sum A(y_i) \mathbf{w}^\top \mathbf{x}_{qi}\right),$$

where C_1 and C_2 are two constants independent of y. Since this problem is NP-Hard, efficient heuristics have to be used for solving it.

Model selection Instead of fixing the function A, it is possible to try to learn it. In the optimization method of section 3.2, A only appears in the kernel matrix (10). So a natural way to learn A is to consider it as a kernel parameter and apply for instance the methods described in [4].

7 Conclusion

By viewing the ranking problem as a structured output learning problem, we have been able to propose an algorithm that directly optimizes a complex quality measure such as the NCDG. Since most other rankings algorithm optimize simpler loss functions, our proposed algorithm could improve the ranking quality on several datasets.

We also proposed a modification of the original structured output learning algorithm [15] that leads to a tighter upper bound on the training loss. We believe that this novelty might not only be useful in ranking, but also in a variety of structured output learning problems, especially the ones where underfitting could be an issue.

References

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[2] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems 19*, 2007.

[3] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[4] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[5] R. Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, 1987.

[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] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

[8] A. Fuduli, M. Gaudioso, and G. Giallombardo. Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. *SIAM Journal on Optimization*, 14(3):743–756, 2004.

[9] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In Smola, Bartlett, Schoelkopf, and Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[10] K. Jarvelin and J. Kekalainen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[11] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–87, 1955.

[12] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994.

[13] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: a large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, 2005.

[14] C. H. Teo, Q. Le, A. Smola, and S.V.N. Vishwanathan. A scalable modular convex solver for regularized risk minimization. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[15] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[16] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:143–175, 1975.