Efficient Coordinate Descent for Ranking with Domination Loss

by

Mark A. Stevens

S.B. EECS, M.I.T., 2009

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May, 2010

Author [illegible]

Department of Electrical Engineering and Computer Science

May 17, 2010

Certified By [illegible]

Yoram Singer, Google Research

VI-A Company Thesis Supervisor

Certified By

Professor Michael Collins

M.I.T. Thesis Supervisor

Accepted By [illegible]

Dr. Christopher J. Terman

Chairman, Department Committee on Graduate Theses

Abstract. We define a new batch coordinate-descent ranking algorithm based on a domination loss, which is designed to rank a small number of positive examples above all negatives, with a large penalty on false positives. Its objective is to learn a linear ranking function for a query with labeled training examples in order to rank documents. The derived single-coordinate updates scale linearly with respect to the number of examples. We investigate a number of modifications to the basic algorithm, including regularization, layers of examples, and feature induction.

The algorithm is tested on multiple datasets and problem settings, including Microsoft's LETOR dataset, the Corel image dataset, a Google image dataset, and Reuters RCV1. Specific results vary by problem and dataset, but the algorithm generally performed similarly to existing algorithms when rated by average precision and precision at top k. It does not train as quickly as online algorithms, but offers extensions to multiple layers, and perhaps most importantly, can be used to produce extremely sparse weight vectors. When trained with feature induction, it achieves similarly competitive performance but with much more compact models.

1 Introduction

Ranking is the task of ordering a set of inputs according to how well they match some label (or set of labels). The learning task is to find a function $F(x)$, where for some set of inputs x, $F(x_i) > F(x_j)$ if and only if x_i should be ranked above x_j. Such a ranking function can be used in a wide variety of applications, from improving upon the results of other learning problems (reranking another algorithm's output) to finding the best matches between complex inputs like images and a given label. The output from ranking may be used in a variety of ways – the total ordering may be useful or the selection of the highest-ranked elements may be useful. Lebanon and Lafferty show that ranking algorithms may be used as input to ensemble classifiers [9] and find that such ensembles are competitive with those based on traditional classifiers.

Ranking methods are commonly developed by modifying traditional classifiers to output a ranking function $F(x)$, where the output of $F(x)$ is roughly analagous to the distance (margin) between an input and a decision boundary in the original classification problem. In such a classifier, points with a larger margin are classified with a greater confidence than those with smaller margins and could be ranked higher. For instance, Grangier et al's PAMIR method for ranking image-label matches is loosely based on classification with SVMs and the kernel trick. [7] In the most general case, training instances may be labeled according to a directed graph, where each edge in the graph indicates that its starting vertex should be ranked above its terminal vertex, with no assumptions that the graph is acyclic or the ranking preferences consistent. In practice, most existing ranking algorithms assume one set of positive examples and one set of negatives. While our algorithm does not efficiently compute the most general case, it does extend to cover training instances that are ordered consistently, i.e. those where the "ranked above" relationship is transitive.

Much prior work has been done on ranking problems. Grangier et al [7] introduced a ranking method used to retrieve images based on input text queries. Their method, Passive-Aggressive model for Image Retrieval (PAMIR) was adapted from kernel-based classifiers. This method exceeds other retreival techniques in experiments with the Corel dataset. One of the strengths of this algorithm is that it does not rely on an intermediate tagging of images with contained concepts, but instead trains directly for ranking performance – a benefit which the new ranking algorithm shares. PAMIR's ranking function F is based on text retrieval, and sets $F(q,p) = q \cdot f(p)$, where q is the query, p is a picture, and $f(p)$ is a mapping from the image space to the text space; finding this function is the learning problem solved by PAMIR. Much like an SVM, the method first redefines $F(q,p) = \sum_t q_t w_t \cdot (p)$, so that w_t defines a hyperplane separator learned for each query term t with some weight q_t. They also recast the problem with kernels. w is then calculated using the Passive-Aggressive algorithm to minimize the hinge loss $L(w) = \sum_{x,y^+,y^-} max(0, 1 - w \cdot q_t p^+ + w \cdot q_t p^-)$. By combining models for different queries, they were able to provide rankings for queries which were not directly trained on. One major advantage of this approach is that the algorithm is online – training examples are randomly drawn, which leads to extremely fast training.

Elisseeff and Weston [6] also developed a SVM-based label ranking learning system. Noting past research showing the merit of controlling model complexity, they use an all-pairs approach to develop a large-margin ranker. Their loss function is based on the average fraction of misordered pairs, and they use regularization to reduce overfitting.

Shashua and Levin developed another SVM-based ranking system and investigated separating instances into classes, with margin between each class, similarly to how we define the multiple layer ranking problem in Sec. 3. Whereas they try two methods for the margin, e.g. maximizing either the smallest margin or the sum of margins, the Domination Rank does both, with more focus on the smaller margin at any step.

Whereas most ranking methods minimize a surrogate loss function rather than maximizing some actual performance measure, Xu and Li introduced a ranking method called AdaRank [8] directly based on performance measures NDCG (Normalized Discounted Cumulative Gain) and MAP (Mean Average Precision). Their algorithm is similar to AdaBoost, in that at each step t they select a weak ranker h_t (like the weak classifier in AdaBoost) based on its performance against the most influential training instances, as determined by the performance measure, instead of some margin. It is designed for the query - multiple document structure used in the Learning To Rank dataset, LETOR, which we use in experiments in Sec. 8.1.

Cao et al [15] introduce a ranking algorithm ListNet based again on the LETOR structure, where inputs are in the form of a list, which allows them to avoid the slow all-pairs formulation of ranking problems – their learning objective is to reproduce this list ordering. This formulation is similar to the transitive layers we will use for multilayer domination rank. They use Neural Networks as the actual learning method. We compare Domination Rank to both ListNet and AdaRank (among other algorithms), and find that it often exceeds these, despite not being targeted to the LETOR dataset as these are.

Schapire and Singer [11] investigated using ranking to solve a multiclass, multiple label problem. In this setting, correct labels should be ranked above incorrect labels. Their algorithm, called AdaBoost.MR, trained a ranker given a set of correct labels and incorrect labels, with errors occuring only when $f(x, l_1) \leq f(x, l_0)$, where l_1 was a correct label and l_0 an incorrect one. Freund et al (2003) [14] introduced an algorithm called RankBoost which was also used to apply the AdaBoost algorithm to ranking. This method outperformed regression and nearest neighbors for the task of ranking movie preferences. Its loss and updates were similar to those used in Collins (2000) [4] and Schapire and Singer (1999) [11]. RankBoost is also included in the LETOR experiments in Sec. 8.1.

Collins (2000, 2002) [3, 4] applied this boosting-based ranker to the problem of reranking outputs from a probabilistic method to improve the selection of a best match. Their algorithm reranked the output of a probabilistic text parser in order to achieve greater accuracy than the parser alone. They investigate two loss functions for ranking, LogLoss $= \sum_i \log\left(1 + \sum_{j=2}^{n_i} e^{F(x_{i,j}) - F(x_{i,1})}\right)$ and ExpLoss $= \sum_i \sum_{j=2}^{n_i} e^{F(x_{i,j}) - F(x_{i,1})}$. Both maximize the margin between the best parsing (a single human-verified correct sentence parsing) and all others.

Additionally, Dekel et al (2005) [10] introduce ranking with boosting using a preference graph and domination error. In their setting, the target ranking is specified by a directed graph $G = (V, E)$. Each vertex v_i corresponds to an input x_i, and each edge e indicates that its initial vertex should be ranked above its terminal vertex. They define a domination error ε_{Dom}, which is the fraction of dominating vertices (instances or vertices with an outgoing edge) that are ranked below at least one of their dominated vertices. Mathematically,

$$\varepsilon_{Dom} = \frac{\sum_{v_p \in V} [[\max_{v_c}(F(v_c)) > F(v_p)]]}{Z} \tag{1}$$

where v_c is a child of v_p and Z is the number of vertices with outgoing edges. $[[\max_{v_c}(F(v_c)) > F(v_p)]] = 1$ only if a dominating vertex v_c is ranked below at least one dominated vertex. See Fig. 1 for a visual

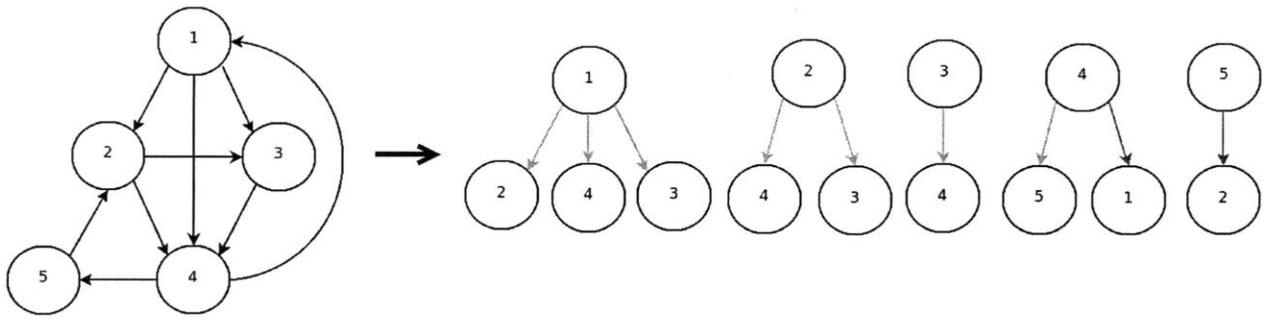


Figure 1: Given a graph of desired labels and rankings $1 > 2 > 3 > 4 > 5$, the subgraphs G_k on the right correspond to partitioning the graph to calculate the domination loss. Here, the domination error $\varepsilon_{Dom} = 2/5$, since the two graphs on the right are not satisfied.

explanation of the domination error.

Similarly, Dekel et al (2005) defined the dominated error ε_{dom}, which instead counts the number of vertices whose incoming edge preferences are not all satisfied. More generally, they define generalized error ε_{gen} for a set of subgraphs, where ε_{gen} is the fraction of subgraphs whose edges are not all satisfied. For instance, the domination error can be derived from a set of subgraphs $G_1, \ldots, G_a$, where each G_i consists of a vertex v_i and the set of edges starting at v_i. If v_i is correctly ranked above all of its children vertices, then graph G_i is satisfied.

Then, given a graph decomposition $A(G_i) = G_{i,1}, \ldots, G_{i,s_i}$, the loss can be found for a boosting model with features $h(x, y)$, where y is a label in the preference graph.

$$\text{Loss}(F, S, A) = \sum_{i=1}^{m} \frac{1}{s_i} \sum_{k=1}^{s_i} L(F(x_i), G_{i,k}) \tag{2}$$

where

$$L(F(x), G) = \log\left(1 + \sum_{e \in E} \exp\left(\sum_t \alpha_t(h_t(x, \text{term}(e)) - h_t(x, \text{init}(e))\right)\right) \tag{3}$$

They investigated dominated and domination loss, along with others, for the label-ranking task on the Reuters dataset, using a boosting-based approach to minimize the loss in Eq. (3). Domination loss in particular should promote useful rankings, since most applications are primarily concerned with the top results from ranking. The domination error promotes focusing on this set of top results in the training set, and so we expect it to generalize to a test set. In this thesis, I further investigate the domination loss, applying it to the instance ranking problem with a new coordinate-descent method for minimizing domination loss.

2 Basic Ranking with Domination Loss

Vectors are denoted in bold face, e.g, $\boldsymbol{x}$, and are assumed to be in column orientation. The transpose of a vector is denoted $\boldsymbol{x}^\dagger$ and is thus a row vector. The (vector) expectation of a set of vectors $\{\boldsymbol{x}_i\}$ with respect to a discrete distribution $\boldsymbol{p}$ is denoted, $\mathbb{E}_{\boldsymbol{p}}[\boldsymbol{x}] = \sum_j p_j \boldsymbol{x}_j$. We are given a set of instances (images) per query which are represented as vectors in $\mathbb{R}^n$. The i^{th} image for query q, denoted $\boldsymbol{x}_{q,i}$, is associated with a quality feedback, denoted $r_{q,i}$. Given groupings of images into queries with their feedback we wish to learn a *ranking*, and not a *classification* function, $f : \mathbb{R}^n \to \mathbb{R}$. Concretely, we would like the function f to induce an ordering such that $f(\boldsymbol{x}_{q,i}) > f(\boldsymbol{x}_{q,j}) \Rightarrow r_{q,i} > r_{q,j}$. We use the domination loss as a surrogate convex loss function to promote the ordering requirement. For an instance $\boldsymbol{x}_{q,i}$ we denote by $\mathcal{D}(q,i)$ the set of instances that should be ranked below it according to the feedback,

$$\mathcal{D}(q,i) = \{j \,|\, r_{q,i} > r_{q,j}\} \quad . \tag{4}$$

The combinatorial domination loss is one if there exists $j \in \mathcal{D}(q,i)$ such that $f(\boldsymbol{x}_{q,j}) > f(\boldsymbol{x}_{q,i})$, that is, the requirement that the i^{th} instance dominates all the instances in $\mathcal{D}(q,i)$ is violated at least once. Note this implies a large effect from false positives.

We would like to optimize this loss function via coordinate descent, so we need a convex approximation to it. The convex relaxation we use for each dominating instance $\boldsymbol{x}_{q,i}$ is,

$$\ell_{\mathcal{D}}(\boldsymbol{x}_{q,i}; f) = \log\left(1 + \sum_{j \in \mathcal{D}(q,i)} e^{f(\boldsymbol{x}_{q,j}) - f(\boldsymbol{x}_{q,i}) + \Delta(q,i,j)}\right) \quad , \tag{5}$$

where $\Delta(q,i,j) \in \mathbb{R}_+$ is a margin requirement between the scores of the i^{th} and the j^{th} instances. See Section 5 for a discussion of adding a regularization term.

For now, we focus on a special case in which $r_{q,i} \in \{-1,+1\}$, that is, each instance is either positive (good, relevant) or negative (bad, irrelevant). Later on we discuss a few generalizations. In this case, the set $\{j \text{ s.t. } \exists i : \boldsymbol{x}_{q,j} \in \mathcal{D}(q,i)\}$ simply amounts to $\{j | r_{q,j} = -1\}$, which does not depend on the index i and thus we overload notation and denote it as $\mathcal{D}(q)$. For simplicity we assume that $\Delta(q,i,j) = 0$ and discuss relaxations of this assumption below. We further assume that f is a linear function, $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$. The empirical loss with respect to $\boldsymbol{w}$ for a set of queries, instances, and their feedback is,

$$L_{\mathcal{D}}(\boldsymbol{w}) = \sum_{q,i} \log \left(1 + \sum_{j \in \mathcal{D}(q,i)} e^{\boldsymbol{w} \cdot (\boldsymbol{x}_{q,j} - \boldsymbol{x}_{q,i})} \right) \quad . \tag{6}$$

For brevity and simplicity we drop the dependency on the query index q, focus on a single domination-loss term, $\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w})$, and perform simple algebraic manipulations,

$$\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) = \log \left(\frac{\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j}}{e^{\boldsymbol{w} \cdot \boldsymbol{x}_i}} \right) = \log \left(\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j} \right) - \boldsymbol{w} \cdot \boldsymbol{x}_i \quad , \tag{7}$$

where $\bar{\mathcal{D}}(i) = \mathcal{D}(i) \cup \{i\}$.

In order to minimize this loss term, we would like to select a series of minimizing updates to the weight $\boldsymbol{w}$. To do so, we will upper bound the true loss function with a Taylor expansion at each step, and then select the update that minimizes that upper bound. Specifically, we would like to construct a quadratic upper bound that may be minimized easily. Let us first compute the gradient and Hessian of $\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w})$ with respect to $\boldsymbol{w}$ so that we may construct this upper bound. Taking the first order derivatives with respect to the components of $\boldsymbol{w}$ we get,

$$\nabla_{\boldsymbol{w}} \, \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) = \frac{\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j} \boldsymbol{x}_j}{\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j}} - \boldsymbol{x}_i = \mathbb{E}_{\boldsymbol{p}_i}[\boldsymbol{x}] - \boldsymbol{x}_i \quad ,$$

where $\boldsymbol{p}_i$ is the empirical distribution whose r^{th} component is $p_r = e^{\boldsymbol{w} \cdot \boldsymbol{x}_r}/Z$ and Z is a normalization constant $\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j}$. Thus,

$$p_r = \frac{e^{\boldsymbol{w} \cdot \boldsymbol{x}_r}}{\sum_{j \in \bar{\mathcal{D}}(i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j}} \quad .$$

Taking the second order derivatives while using the same convention above we get that the Hessian computed at $\boldsymbol{w}$ is,

$$H_i(\boldsymbol{w}) = \nabla^2_{\boldsymbol{w}} \, \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{p}_i}\left[\boldsymbol{x}\boldsymbol{x}^\dagger\right] - \mathbb{E}_{\boldsymbol{p}_i}[\boldsymbol{x}] \left(\mathbb{E}_{\boldsymbol{p}_i}[\boldsymbol{x}]\right)^\dagger \quad .$$

Using the mean value theorem, the loss for a single query at $\boldsymbol{w} + \boldsymbol{\delta}$ can now be written as,

$$\begin{aligned} \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w} + \boldsymbol{\delta}) &= \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) + \nabla_{\boldsymbol{w}} \cdot \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^\dagger \, H_i(\boldsymbol{w} + \alpha \boldsymbol{\delta}) \, \boldsymbol{\delta} \\ &= \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) + \left(\mathbb{E}_{\boldsymbol{p}_i}[\boldsymbol{x}] - \boldsymbol{x}_i\right) \cdot \boldsymbol{\delta} + \frac{1}{2} \mathbb{E}_{\tilde{\boldsymbol{p}}_i}\left[(\boldsymbol{\delta} \cdot \boldsymbol{x})^2\right] - \frac{1}{2}\left(\boldsymbol{\delta} \cdot \mathbb{E}_{\tilde{\boldsymbol{p}}_i}[\boldsymbol{x}]\right)^2 \quad , \end{aligned}$$

where $\tilde{\boldsymbol{p}}_i$ is the empirical distribution induced by $\boldsymbol{w} + \alpha\boldsymbol{\delta}$ for some $\alpha \in [0,1]$.

Now, let us assume that we would like to update a single coordinate on each round. (We can revisit this assumption later.) Denote the index of the feature whose weight is going to be updated by r. Let $\boldsymbol{e}_r$ denote the singleton vector whose components are 0 except for the r^{th} component which is 1. Then, the Taylor expansion can be written as,

$$\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w} + \delta\boldsymbol{e}_r) = \kappa(\boldsymbol{x}_i, \boldsymbol{w}) + \delta\left(\sum_{j\in\bar{\mathcal{D}}(q,i)} p_j x_{j,r} - x_{i,r}\right) + \frac{1}{2}\delta^2\left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}^2 - \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}\right)^2\right) ,$$

where $\kappa(\boldsymbol{x}_i, \boldsymbol{w})$ is a constant that does not depend on δ. The problem is that we do not know $\tilde{\boldsymbol{p}}_j$. We can use a few bounds on the norm of the instances. For brevity let us keep focusing on a single query q and let B_r be a bound on the absolute value of the r^{th} feature for all the instances retrieved for query q such that $B_r \geq x_{j,r}^2 \forall j$. Thus, we can bound the Hessian,

$$\begin{aligned} H_{rr}(\boldsymbol{w} + \alpha\boldsymbol{\delta}) &= \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}^2 - \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}\right)^2\right) \\ &\leq \sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}^2 \\ &\leq B_r \ . \end{aligned}$$

While dropping the squared term leads to a loose bound, it is necessary that we maintain a strict upper bound to the loss in order to guarantee stability of coordinate descent (and it is entirely possible that the squared term will be zero for any given feature, especially with a sparse feature set). An upper bound is necessary. Both the loss and the approximation are convex (the approximation is a quadratic, and the loss was designed to be convex). Thus, as long as the approximation is an upper bound (with the same slope and value at any current $\boldsymbol{w}$), minimizing it will yield a new $\boldsymbol{w}$ at which both the approximation function and the true loss are smaller. [2] Experimental attempts to find a tighter bound generally led to instability and did not converge. While we could find a slightly tighter theoretical bound by setting a different $B_{i,r}$ for each dominating instance x_i, in practice the maximal $x_{j,r}$ likely occurs in dominated instances, so $B_{i,r}$ would be the same as the simpler B_r in practice.

The generalization to multiple queries is straightforward and involves an additional summation. Let m_q denote the number of dominating elements for query q, $m_q = |\{i|r_{q,i} = +1\}|$. Then, we can bound the sum of domination losses for query q,

$$L(\boldsymbol{w} + \delta \boldsymbol{e}_r) \leq \sum_{i:r_{q,i}=+1} \kappa(\boldsymbol{x}_i, \boldsymbol{w}) + \delta \sum_{i:r_{q,i}=+1} \left(\sum_{j \in \bar{\mathcal{D}}(q,i)} p_j x_{j,r} - x_{i,r} \right) + \frac{m_q B_r}{2} \delta^2 \ .$$

The above is a quadratic equation in δ and thus the value of δ which minimizes it can be easily obtained. It therefore remains to show how to efficiently compute the multiplier of the linear element in the above equation. Note that for a two-tier feedback the sets $\mathcal{D}(q,i)$ are all the same and consist of the negatively rated elements. We thus abbreviate $\mathcal{D}(q,i)$ as $\mathcal{D}(q)$ and simplify the summation multiplying δ in the above equation as follows:

$$\begin{aligned}
\sum_{i:r_{q,i}=+1} \sum_{j \in \bar{\mathcal{D}}(q,i)} p_j x_{j,r} &= \sum_{i:r_{q,i}=+1} \left(\sum_{j \in \mathcal{D}(q)} p_j x_{j,r} + p_i x_{i,r} \right) \\
&= \sum_{i:r_{q,i}=+1} \sum_{j \in \mathcal{D}(q)} \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_j}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} x_{j,r} + \sum_{i:r_{q,i}=+1} \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} x_{i,r} \\
&= \left(\sum_{i:r_{q,i}=+1} \frac{1}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \right) \left(\sum_{j \in \mathcal{D}(q)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} x_{j,r} \right) + \sum_{i:r_{q,i}=+1} \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_i} x_{i,r}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \ ,
\end{aligned}$$

where

$$Z_q = \sum_{j \in \mathcal{D}(q)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} \ .$$

Note that the above expression can be computed in time that is proportional to the number of instances returned for query q, and *not* the number of pairs. Thus, each update can be calculated:

$$\delta = \frac{1}{m_q B_r} \left[\sum_i \frac{Z_q x_{i,r}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} - \left(\sum_i \frac{1}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \right) \left(\sum_{j \in \mathcal{D}(q)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} x_{j,r} \right) \right]$$

Let us define an auxiliary variable

$$\mu_{q,r} = \sum_{j \in \mathcal{D}(q)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} x_{j,r} \ .$$

Then,

$$\delta = \frac{1}{m_q B_r} \sum_i \frac{Z_q x_{i,r} - \mu_{q,r}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \ .$$

The algorithm for initialization of variables used in the coordinate descent is shown in Alg. 1, while the set

Algorithm 1 Initialization: Bounds need only be calculated once, before training begins. Margins for all instances are initially set to zero, since the weight vector is zero.

```
for each feature $r$ do
  $B_r = \max_j x_{jr}^2$
end for
for each instance $i$ do
  $margin_i = 0$
end for
$m_q = |\{i \in \bar{\mathcal{D}}(q)\}|$
$Z = m_q$
```

Algorithm 2 Training: The weight vector is updated, one feature at a time. Stopping criteria will be discussed in Sec. 7.1, and depends on either a maximum number of steps or a combination of regularization and the remaining possible decrease in loss. At each step, only changes to be applied to μ, Z, and the margins are calculated, which avoids the need to iterate over all features or instances on each update.

```
while not converged do
  for each feature $r$ do
    $g = 0$
    $\mu = 0$
    for each $j \in \bar{\mathcal{D}}(q)|x_{jr} \neq 0$ do
      $\mu \leftarrow \mu + e^{margin_j} x_{jr}$
    end for
    for each $i \notin D(q)|x_{ir} \neq 0$ do
      $g \leftarrow g + (Z * x_{ir} - \mu_r)/(Z + e^{margin_i})$
    end for
    $\delta \leftarrow g/(m_q B_r)$
    $w_r \leftarrow w_r + \delta$
    for each $j \in D(q)|x_{jr} \neq 0$ do
      $Z \leftarrow Z - e^{margin_j}$
      $margin_j \leftarrow margin_j + \delta * x_{ir}$
      $Z \leftarrow Z + e^{margin_j}$
    end for
  end for
end while
```

of updates is shown in Alg. 2. While each update depends on significant calculation, much of that calculation can be saved from prior update rounds. Saving so much old data presents possible problems of rounding and accumulated small errors. We investigated recalculating these stored variables from scratch after many rounds of updates in experiments, and found that the only effect of this recalculation was increased training time – any accumulated errors were insignificant. The final implementation also avoids numerical overflow problems with exponentials, as described in Section 6.

3 Generalization to Multiple Layers

Consider now the generalization to more than one layer of ranking labels. Above, we assumed that for any query there exist a set of correct matches and a set of incorect matches; labels $r_{q,i}$ were restricted to the set $\{-1,+1\}$. The derivations above can be extended to the case where there may be up to K ordered groups. Let $G(k)$ denote the kth set of instances, such that $r_i \in G(k_1) < r_j \in G(k_2)$ for all layers $k_1 < k_2$. That is, any instance in $G(k)$ dominates all instances in groups below it $G(k-1), G(k-2), \ldots, G(0)$. The instances dominated by some $i \in G(k)$ are then denoted by $\mathcal{D}(q,k) = \{j \in G(m) | m < k\}$.

The bound for $\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w})$ with multiple layers is identical for any given x_i:

$$\ell_{\mathcal{D}}(\boldsymbol{x}_{i\in G(k)}; \boldsymbol{w}) = \log \left(\frac{\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j}}{e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \right) = \log \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} \right) - \boldsymbol{w}\cdot\boldsymbol{x}_i \ ,$$

where $\bar{\mathcal{D}}(q,i) = \mathcal{D}(q,k) \cup \{i\}$. Following the same arguments as before,

$$\ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w} + \delta \boldsymbol{e}_r) \quad = \quad \kappa(\boldsymbol{x}_i, \boldsymbol{w}) + \delta \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} p_j x_{j,r} - x_{i,r} \right) + \frac{1}{2}\delta^2 \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r}^2 - \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} \tilde{p}_j x_{j,r} \right)^2 \right) \ ,$$

and the bound on the Hessian remains B_r. When summing over all dominating x_i, however, we break down the summation by group k:

$$L(\boldsymbol{w} + \delta \boldsymbol{e}_r) \leq \sum_{k=1}^{K-1} \sum_{i\in G(k)} \kappa(\boldsymbol{x}_i, \boldsymbol{w}) + \delta \sum_{k=1}^{K-1} \sum_{i\in G(k)} \left(\sum_{j\in\bar{\mathcal{D}}(q,i)} p_j x_{j,r} - x_{i,r} \right) + \frac{1}{2} \sum_{k=1}^{K-1} m_q(k) B_r \delta^2 \ ,$$

where $m_q(k)$ is the number of elements in group k. The quadratic term can still be bounded by $m_q B_r/2$, where m_q is the number of dominating elements, that is, the number of elements not in the bottom layer $k = 0$.

Again, efficient computation requires decomposing the linear multiplier. By the same argument as earlier,

$$\sum_k \sum_{i\in G(k)} \sum_{j\in\bar{\mathcal{D}}(q,i)} p_j x_{j,r} = \left(\sum_k \sum_{i\in G(k)} \left(\frac{1}{Z_{q,k} + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \sum_{j\in\mathcal{D}(q,k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} x_{j,r} \right) \right) + \sum_k \sum_{i\in G(k)} \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_i} x_{i,r}}{Z_{q,k} + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \ ,$$

where

$$Z_{q,k} = \sum_{j\in\mathcal{D}(k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} \ .$$

The auxiliary variable $\mu_{q,r}$ from Section 2 generalizes to:

$$\mu_{q,k,r} = \sum_{j \in G(k)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j} x_{j,r} \quad .$$

Thus, the multilayer gradient is:

$$\sum_{k=1}^{K-1} \sum_{i \in G(k)} \left(\sum_{j \in \bar{\mathcal{D}}(q,i)} p_j x_{j,r} - x_{i,r} \right) = \sum_{k} \sum_{i \in G(k)} \frac{\mu_{q,k,r} - Z_{q,k} x_{i,r}}{Z_{q,k} + e^{\boldsymbol{w} \cdot \boldsymbol{x}_i}} \quad .$$

Note that Z and μ can be constructed recursively in $O(n)$ time given n inputs:

$$\begin{aligned} Z_{q,k} &= \sum_{j \in G(k-1)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j} + Z_{q,k-1} \\ \mu_{q,k,r} &= \sum_{j \in G(k-1)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j} x_{j,r} + \mu_{q,k-1,r} \quad . \end{aligned}$$

To recap, each multilayer update δ is computed as follows:

$$\delta = \frac{1}{m_q B_r} \sum_{k=1}^{K-1} \sum_{i \in G(k)} \frac{Z_{q,k} x_{i,r} - \mu_{q,k,r}}{Z_{q,k} + e^{\boldsymbol{w} \cdot \boldsymbol{x}_i}} \quad .$$

For the changes to the basic algorithm shown in Alg. 1 and Alg. 2 to adapt these to the multilayer case, see Alg. 3 and Alg. 4.

Algorithm 3 Multilayer Initialization: Z must now be calculated for each layer.

```
for each feature $r$ do
    $B_r = \max_j x_{jr}^2$
end for
for each instance $i$ do
    $margin_i = 0$
end for
$m_q = |\{i \in G(k) \forall k \neq 0\}|$
for each layer $k$ do
    $Z_k \leftarrow Z_{k-1} + |G(k)|$
end for
```

Algorithm 4 Multilayer Training: Now, μ and Z are calculated for each layer k. Just as before, most of the calculation is saved from prior rounds.

```
while not converged do
  for each feature $r$ do
    $g = 0$
    for each layer $k$ do
      $\mu_k = 0$
      for each $j \in G(k) | x_{jr} \neq 0$ do
        $\mu_k \leftarrow \mu_k + e^{margin_j} x_{jr}$
      end for
      for each $i \notin D(q) | x_{ir} \neq 0$ do
        $g \leftarrow g + (Z_k * x_{ir} - \mu_{kr}) / (Z_k + e^{margin_i})$
      end for
    end for
    $\delta \leftarrow g / (m_q B_r)$
    $w_r \leftarrow w_r + \delta$
    for each layer $k$ do
      $Z_{k,old} = Z_k$
      $Z_k \leftarrow Z_k + Z_{k-1} - Z_{k-1,old}$
      for each $j \in G(k) | x_{jr} \neq 0$ do
        $Z_k \leftarrow Z_k - e^{margin_j}$
        $margin_j \leftarrow margin_j + \delta * x_{ir}$
        $Z_k \leftarrow Z_k + e^{margin_j}$
      end for
    end for
  end for
end while
```

4 Incorporating Margin

As a generalization of the work presented above, we would also like to consider the case where it is not sufficient for positive examples to be ranked above negative examples – we want them to be separated by some margin $\Delta(r, s)$. Thus, the loss function with margin would be:

$$L_{\mathcal{D}} = \sum_{q,i} \log \left(1 + \sum_{j \in \bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w} \cdot (\boldsymbol{x}_{q,j} - \boldsymbol{x}_{q,i}) + \Delta(i,j)} \right) \quad .$$

We will focus on the case where the margin is separable into a sum of two functions, $\Delta(i, j) = s(i) - s(j)$, since updates for a non-separable margin cannot be calculated efficiently. In the separable case, each term $e^{\boldsymbol{w} \cdot \boldsymbol{x}_j}$ can be replaced by $e^{\boldsymbol{w} \cdot \boldsymbol{x}_j \pm s(j)}$. More concretely,

$$\nabla_{\boldsymbol{w}} \ell_{\mathcal{D}}(\boldsymbol{x}_i; \boldsymbol{w}) = \frac{\sum_{j \in \bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j \pm s(j)} \boldsymbol{x}_j}{\sum_{j \in \bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j \pm s(j)}} - \boldsymbol{x}_i \;=\; \mathbb{E}_{p_i}[\boldsymbol{x}] - \boldsymbol{x}_i \quad ,$$

Algorithm 5 Initialization for Margins

```
...
for each instance $i$ do
    $margin_i = s(i)$
end for
...
```

where probability is now defined

$$p_r = \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_r \pm s(r)}}{\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j \pm s(j)}} \ .$$

Given this new definition for p_r, the rest of the derivation is identical to that for the original zero-margin case, and yields updates

$$\delta = \frac{-1}{m_q B_r}\sum_i \frac{\mu_{q,r} - Z_q x_{i,r}}{Z_q + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i + s(i)}} \ ,$$

where

$$\begin{aligned} \mu_{q,r} &= \sum_{j\in\mathcal{D}(q,k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j - s(j)} x_{jr} \\ Z_q &= \sum_{j\in\mathcal{D}(q,k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j - s(j)} \ . \end{aligned}$$

Thus, update calculation remains linear in the number of examples. The other variants may be extended to use a separable margin by initializing margins to a nonzero value (See Alg. 5).

4.1 Non-Separable

As before, we define a probability

$$p_{r,i} = \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_r + \Delta(i,r)}}{\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + \Delta(i,j)}} \ .$$

While the bound on the Hessian will not change, since it does not depend on margins, the gradient calculation does change:

$$\nabla_{\boldsymbol{w}}\, \ell_{\mathcal{D}}(\boldsymbol{x}_i;\boldsymbol{w}) = \frac{\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + \Delta(i,j)}\boldsymbol{x}_j}{\sum_{j\in\bar{\mathcal{D}}(q,i)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + \Delta(i,j)}} - \boldsymbol{x}_i \ = \ \mathbb{E}_{\boldsymbol{p}_i}[\boldsymbol{x}] - \boldsymbol{x}_i \ .$$

Note the new dependence of probability on each dominating $\boldsymbol{x}_i$. As a result, updates cannot be separated into sums of independent calculations on dominating and dominated points:

$$\delta = \frac{1}{m_q B_r}\sum_i \frac{Z_{q,i} x_{i,r} - \mu_{q,r,i}}{Z_{q,i} + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i}} \ ,$$

where

$$\begin{aligned}
\mu_{q,r,i} &= \sum_{j \in \mathcal{D}(q,k)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j + \Delta(i,j)} x_{jr} \\
Z_{q,i} &= \sum_{j \in \mathcal{D}(q,k)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j + \Delta(i,j)} \quad .
\end{aligned}$$

These updates cannot be computed in linear time. If, however, if the non-separable margin is defined between layers as described in Section 3 (see Figure 2), then some separation can be done:

$$\delta = \frac{-1}{m_q B_r} \sum_k \sum_{i \in G(k)} \frac{\mu_{q,k,r} - Z_{q,k} x_{i,r}}{Z_{q,k} + e^{\boldsymbol{w} \cdot \boldsymbol{x}_i}} \quad ,$$

where

$$\begin{aligned}
\mu_{q,k,r} &= \sum_{j \in \mathcal{D}(q,k)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j + \Delta(k,j)} x_{jr} \\
Z_{q,k} &= \sum_{j \in \mathcal{D}(q,k)} e^{\boldsymbol{w} \cdot \boldsymbol{x}_j + \Delta(k,j)} \quad .
\end{aligned}$$

Each $Z_{q,k}$ and $\mu_{q,k,r}$ must be recalculated independently, since they are no longer purely additive as before. Thus, the runtime will be $O(nk)$, where n is the number of examples and k is the number of layers.

5 Regularization

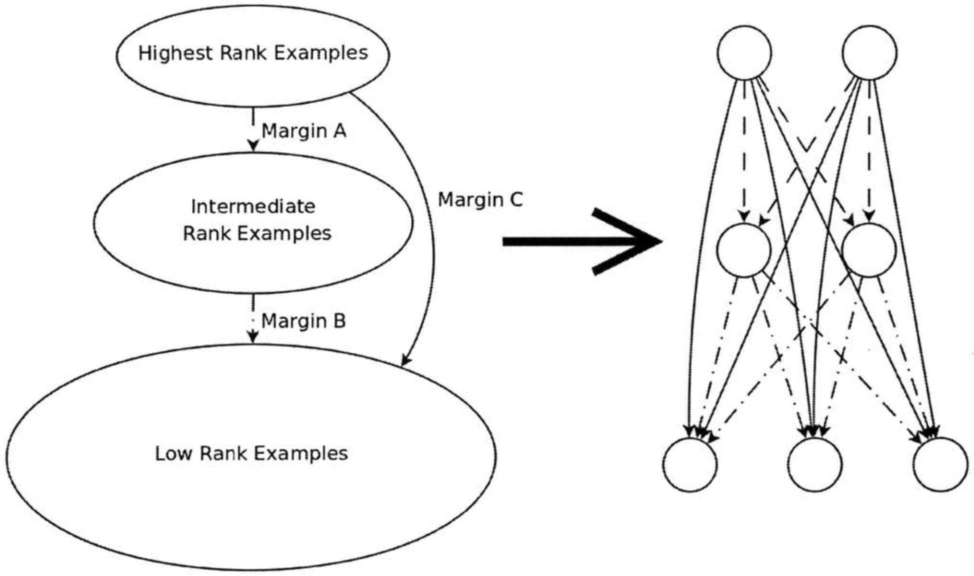
We would also like to promote sparse weight vectors by penalizing complexity. In experiments with simulated data, we found that applying a regularization penalty was essential for good performance on datasets with label noise. Adding regularization will also help determine a good stopping point for the learning algorithm. In each case, we generalize the loss from previous sections to be

$$L(\boldsymbol{w} + \delta \boldsymbol{e}_r) \leq L(\boldsymbol{w}) + g_r \delta + \frac{\beta}{2} \delta^2$$

and add the term $\lambda |\boldsymbol{w} + \delta \boldsymbol{e}_r|$.

First, we consider ℓ_1 regularization. For an ℓ_1 penalty, we want to minimize the upper bound $L(\boldsymbol{w}) + g_r \delta + \frac{\beta}{2} \delta^2 + \lambda \left\| \boldsymbol{w} + \delta \boldsymbol{e}_r \right\|_1$ with respect to δ. We define two lemmas to help with the derivation of a minimizing

Figure 2: One margin may be specified per boundary between layers. As shown above, Margin A translates to the dashed arrows, Margin B to the dotted dashed arrows, and Margin C to the solid arrows.



update $\delta^\star$, where

$$\delta^\star = \arg\min_{\delta} \; g_r\delta + \frac{\beta}{2}\delta^2 + \lambda \left\|w_r + \delta\right\|_1 \quad . \tag{8}$$

Lemma 1. *If $\beta w_r - g_r > 0$, then $w_r + \delta^\star \geq 0$. Likewise, if $\beta w_r - g_r < 0$, then $w_r + \delta^\star \leq 0$.*

Proof. Without loss of generality, consider the case where $\beta w_r - g_r > 0$. Suppose that $w_r + \delta^\star < 0$. Then, Eq. (8) reduces to:

$$0 = g_r + \beta\delta^\star - \lambda \quad .$$

If we combine this equation with our bound on $\beta w_r - g_r$, and recall that $\lambda \geq 0$ and $\beta > 0$, we find that

$$\begin{aligned} \beta(w_r + \delta^\star) &> \lambda \\ (w_r + \delta^\star) &> 0 \quad , \end{aligned}$$

which is a contradiction, so $w_r + \delta^\star \geq 0$. The symmetric case follows similarly.

□

Lemma 2. *The minimizing update $\delta^\star = -w_r$ iff $\lambda \geq |g_r - \beta w_r|$.*

Proof. Without loss of generality, consider the case where $\beta w_r - g_r > 0$. We can then use Lemma 1 to rewrite Eq. (8), substituting $w_r + \delta$ for $\|w_r + \delta\|_1$ and adding the constraint that $w_r + \delta \geq 0$.

$$\delta^\star = \arg\min_\delta \; g_r\delta + \frac{\beta}{2}\delta^2 + \lambda(w_r + \delta) \;\text{ s.t. }\; w_r + \delta \geq 0$$

If the constraint is satisfied, $\delta^\star = -(\lambda + g_r)/\beta \geq -w_r$. If, however, the minimum point of the quadratic $(\lambda + g_r)/\beta \geq w_r$, then the minimizing value must be on the constraint boundary, $\delta^\star = -w_r$. Thus, if $\lambda \geq \beta w_r - g_r \geq 0$, $\delta^\star = -w_r$, and if $\beta w_r - g_r > \lambda$, then $\delta^\star = -(\lambda + g_r)/\beta$. The symmetric case where $g_r - \beta w_r \leq 0$ follows similarly. $\square$

Thus, the ℓ_1 regularized updates can be summarized:

$$\delta = \begin{cases} -w_r & \lambda \geq |g_r - \beta w_r| \\ -\frac{g_r+\lambda}{\beta} & \lambda < -g_r + \beta w_r \\ \frac{-g_r+\lambda}{\beta} & \lambda < g_r - \beta w_r \end{cases} .$$

We would also like to consider ℓ_2^2 regularization:

$$\begin{aligned} L(\boldsymbol{w} + \delta \boldsymbol{e}_r) &\leq L(\boldsymbol{w}) + g_r\delta + \frac{\beta}{2}\delta^2 + \lambda \|w_r + \delta\|_2^2 \\ &\leq L(\boldsymbol{w}) + g_r\delta + \frac{\beta}{2}\delta^2 + \lambda \left(\sum_{j \neq r} w_j^2 + (w_r + \delta)^2 \right) \end{aligned}$$

Thus, we have an easily minimized quadratic and can find the minimizing update $\delta^\star$:

$$0 = g_r + \beta\delta^\star + 2\lambda(w_r + \delta^\star) \quad ,$$

so

$$\delta^\star = \frac{-g_r - 2\lambda w_r}{\beta + 2\lambda} \quad .$$

6 Efficient Implementation for Single-Coordinate Updates

As shown above, in order to calculate updates δ for any coordinate r with or without regularization for some number of layers, we need to compute the gradient of the loss with respect to a single coordinate, $\nabla_r L(w)$, an upper bound on the Hessian along that coordinate, β, and possibly a regularization term. I exclude here the details for non-separable margins, as they break the linear growth the algorithm will otherwise exhibit.

6.1 Efficient calculation of β

As shown above, $\beta_r = m_q B_r$. These variables can be calculated for each r in a preprocessing step (or generated on the fly and cached for future use). In either case, any β_r can be calculated in $O(n)$ time, and the whole set may be calculated in parallel.

6.2 Efficient calculation of $\nabla_r L(w)$

Calculating the gradient is substantially more involved. Recall that in the most general case,

$$\nabla_r L(w) = \sum_k \sum_{i \in G(k)} \frac{\mu_{k,r} - Z_k x_{i,r}}{Z_k + e^{\boldsymbol{w}\cdot\boldsymbol{x}_i + s(i)}} \quad ,$$

where

$$\begin{aligned}
\mu_{k,r} &= \sum_{j \in \mathcal{D}(k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} x_{j,r} = \sum_{j \in G(k-1)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} x_{j,r} + \mu_{k-1,r} \\
Z_k &= \sum_{j \in \mathcal{D}(k)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} = \sum_{j \in G(k-1)} e^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} + Z_{k-1} \quad ,
\end{aligned}$$

and

$$\begin{aligned}
Z_0 &= 0 \\
\mu_{0,r} &= 0 \ .
\end{aligned}$$

Clearly, then, all μ_k, Z_k, and $\nabla_r L(w)$ can be calculated in $O(nd)$ time, where d is the time required to compute any single $e^{\boldsymbol{w}\cdot\boldsymbol{x}}$. If we calculate each of these on the fly, $d = O(m)$ where any input has m features. Doing so would be highly inefficient, however, since only one component of x changes at each iteration:

$$e_t^{\boldsymbol{w}\cdot\boldsymbol{x}_j} = e_{t-1}^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} e^{x_{j,r}\delta} \quad ,$$

where the rth coordinate was updated on iteration $t-1$, and $e_{t=0}^{\boldsymbol{w}\cdot\boldsymbol{x}_j + s(j)} = e^{s(j)}$. Thus, if we store the margins $\boldsymbol{w}_t \cdot \boldsymbol{x}_j + s(j)$ at each iteration, we can compute a new $\boldsymbol{w}_{t+1} \cdot \boldsymbol{x}$ in constant time. More obviously, we will also want to compute and store $\mu_{k,r}$ and Z_k once within each iteration, since they are the same for

Algorithm 6 Let M_i denote the margin associated with the ith example, $M_i = \boldsymbol{w} \cdot \boldsymbol{x}_i + g(i)$. Given an index r on iteration t and update δ applied to the previous coordinate s on iteration $t-1$, the gradient g_r can be calculated as follows:

```
for all $k$ do
  $Z_k^t = Z_k^{t-1}$
end for
for all $x_{i,s}, x_{i,r} \neq 0$ do
  $M_i^t \leftarrow M_i^{t-1} + \delta x_{i,s}$
  $Z_k \leftarrow Z_k + \exp(M_i^t) - \exp(M_i^{t-1})$
  $\mu_{k,r} \leftarrow \mu_{k,r} + x_{i,r}(\exp(M_i^t) - \exp(M_i^{t-1}))$
  if $c < M_i^t$ then
    $c_{old} = c$
    $c \leftarrow M_i^t$
    $Z_k \leftarrow Z_k e^{c_{old}-c}$
    $\mu_{k,r} \leftarrow \mu_{k,r} e^{c_{old}-c}$
  end if
end for
```

all i. Some of these margins are, however, likely to be large enough to cause the exponentials to overflow. To avoid this problem, we will also calculate a $c = \max_i \boldsymbol{w} \cdot \boldsymbol{x}_i$, and for the purposes of calculating any e^z instead calculate e^{z-c}. Thus, we will have better numerical stability.

Before deriving an update algorithm, we should also consider the fact that for any update δ, if the inputs are sparse, then most of the margins will not change. Thus, we can compute the margins, μ, and Z by storing previous computations and then altering them with only the changed portions. Thus, I propose decomposing the gradient calculation as shown in Algorithm 6.

Thus, we want to calculate M, Z, and μ together and then give the results to another piece of code to calculate the actual updates.

6.3 Efficient calculation of regularization and δ

With no regularization, δ may simply be calculated as $-\nabla_r L(w)/\beta$. With regularization, this calculation may become more complex, but it only depends on the gradient, upper bound on the Hessian, and $\boldsymbol{w}$, and may be calculated in constant time. This time bound should be obvious for ℓ_1 and ℓ_2^2, which can be recalculated on each iteration with only the previous value and the last single-coordinate update.

7 Feature Induction

The algorithm described above will work for many datasets, but in some cases it maybe possible to train a good model much faster. Consider the case where a dataset consists of a large number of features, which are mostly sparse for any given instance. If we are using regularization (particularly if we are using ℓ_1 regularization with a large regularization weight), our learned weight vector will be sparse. For instance, when training on a set of images with ten thousand features, we found that a good model was learnable with only 700 nonzero features. Instead of training on all of these features, then, we could infer which ones will be useful and focus on training this subset of the features. More concretely, we can bound the change in loss for an update δ_r for feature r:

$$L_{\mathcal{D}}(\boldsymbol{w}) - L_{\mathcal{D}}(\boldsymbol{w} + \delta_r \boldsymbol{e}_r) \geq -\delta_r g_r - \frac{m_q B_r}{2}\delta_r^2 \tag{9}$$

Using this bound, we can select the top α features that have the largest minimum decrease in loss. Once we have trained these until they are close to convergence, we can recalculate the loss bound and add additional features. See Alg. 7. With sufficient regularization, a lack of good additional features provides a stopping condition for the algorithm. In experiments, values for α were selected via cross-validation.

Algorithm 7 Feature Induction: The most promising features are chosen, and then trained to convergence. Once no more features remain to train, the algorithm terminates.

```
{Chosen Features} ← $\emptyset$
while True do
  for each feature $r$ do
    Calculate gradient $g_r$ and update $\delta_r$
    minloss$_r$ ← $-\delta_r g_r - \frac{m_q B_r}{2}\delta_r^2$
  end for
  Sort features by minloss
  new_features ← Top $\alpha$ features $\notin$ {Chosen Features}
  if new_features == $\emptyset$ then
    break
  end if
  {Chosen Features} ← {Chosen Features} $\cup$ new_features
  Train {Chosen Features} to convergence.
end while
```

7.1 Stopping Criteria

For most experiments, we simply set a maximum number of iterations as a stopping criterion. For some smaller datasets, such as the LETOR dataset, the algorithm terminated when the decrease in loss at one iteration through the features was small compared to the initial change in loss.

Once feature induction was added, though, early termination was easy to achieve. Since only the most useful features are chosen, and then those are trained to convergence, the model converges much more quickly. Once there are no new features to select, since all the best features have already converged, training terminates. In the experiments described in Sec. 8, such convergence with feature induction was common.

8 Results

We have evaluated the domination rank algorithm against four datasets and found it to be competitive with existing algorithms. We investigate webpage ranking with Microsoft's LETOR dataset, which is of relatively modest size. We then look at image ranking, specifically at the Corel image datasets and an internal Google dataset. Finally, we investigate scaling to many features and many instances with Reuters (RCV1) document ranking.

For each dataset, we endeavored to find a combination of the modifications described above which yield better performance than other algorithms. In general, the algorithm performs about as well as the best published results for the various datasets. Some modifications have substantial impact – for instance, feature selection drastically decreases training time and promotes sparse weight vectors. The datasets are generally very noisy, and regularization is key to achieving good generalization.

8.1 LETOR Dataset

Microsoft's Learning To Rank (LETOR) [13] consists of nine small datasets with precomputed features. For each of these datasets, there are associated results from many other algorithms.

8.1.1 Dataset Structure

In LETOR, each dataset consists of a set of queries; for each query, there is a set of documents. These documents are divided into three layers based on how relevant they are to the query – the most relevant documents are in layer 2, the mediocre documents in layer 1, and the irrelevant documents in layer 0. The features are all calibrated against the individual query, so the objective is to find a weighting of these features that performs well for all queries, thereby ranking the good above the bad on a query by query basis (see Fig. 3). For example, Letor4 contains web pages that correspond to search results, where each page is described by 46 features that indicate properties like how frequently the search term appears in the document.

Previous work with this dataset has focused on learning a weighting of feature importances on a query-by-query basis. For instance, the margin between instances is only considered between two documents associated with the same query.

Figure 3: Each LETOR dataset consists of a set of queries, each of which is associated with some training/test instances, which are labeled based on how relevant they are to that query.

Query	Document	Document Layer	Document Features
q_0	D_0	1	$0.5, 1.4, \ldots$
	D_1	2	$\ldots$
	D_2	0	$\ldots$
q_1	D_0	0	$\ldots$
	D_1	1	$\ldots$
	D_2	1	$\ldots$
	D_3	0	$\ldots$
	D_4	2	$\ldots$

Thus, there are few enough examples for each query to make the all-pairs formulation feasible.

The dataset defines a paritioning into training, validation, and test examples, so that our choice of which examples to use for each are consistent with those used by the competing algorithms.

8.1.2 Application of Domination Rank to LETOR

In order to apply domination ranking, we group the instances by layer, neglecting which query they came from. In so doing, we define the problem more loosely than did the other algorithms. For each dataset, we tried several variations on the basic algorithm. In each case, the model was trained to convergence and relevant parameters were selected via cross validation with a validation set. These variations consisted of:

- The basic algorithm (with regularization selected via cross-validation) performed similarly to existing algorithms, but was slightly more effective at properly ordering the top-ranked instances than at ordering the full ranking. Here, we used the same three layers as elsewhere in LETOR.
- The algorithm with both semi-relevant and very-relevant instances treated as one binary positive layer of instances. Unsurprisingly, this method often performed much worse than the multilayer method, which has more useful labels. This method demonstrates the usefulness of using multiple layers.
- The algorithm with an additional separable margin. In the basic algorithm, any positive instance ranked above a negative was considered correct for training purposes. Here, a positive must be ranked above a negative by at least some extra margin to be considered correctly ranked. The effect of this modification is for training to focus more on the instances which are being correctly learned. There is no clear trend as to whether this method does better or worse than the basic algorithm, most likely

Figure 4: Performance on the Microsoft LETOR Datasets. Domination Rank is shown against published results from other algorithms. Layered denotes the domination rank algorithm with cross-validated regularization and multiple layers. Flat denotes a bilayer formulation, and Margin a separable margin term, again selected via cross-validation. Results are measured by normalized discounted cumulative gain (NDCG) at the top five and top ten instances per query, and by precision at the top ten.

NDCG5

Algorithm	Domination Rank			RankSVM	ListNet	AdaRank		RankBoost
Dataset	Layered	Flat	Margin	Struct		NDCG	MAP	
Letor4-MQ2007	0.4065	0.415	0.4149	0.41426	0.417	0.4102	0.407	**0.4183**
Ohsumed	**0.4759**	0.4726	0.4701	N/A	0.4432	0.4673	0.4613	0.4494
2003_np	0.7685	0.7667	0.7682	N/A	**0.7843**	0.7447	0.7482	0.7818
2004_np	0.7558	0.6716	0.7424	N/A	**0.7965**	0.7122	0.731	0.6512
2003_hp	0.7988	0.8032	0.7941	N/A	**0.8298**	0.8006	0.8252	0.8034
2004_hp	0.7318	0.7344	0.7517	N/A	0.7694	0.792	**0.8277**	0.7211
2003_td	**0.340**	0.3356	0.3348	N/A	0.3393	0.2939	0.3029	0.3149
2004_td	0.3366	0.328	0.3194	N/A	0.3325	0.3514	0.3602	**0.3878**

NDCG10

Algorithm	Domination Rank			RankSVM	ListNet	AdaRank		RankBoost
Dataset	Layered	Flat	Margin	Struct		NDCG	MAP	
Letor4-MQ2007	0.44	0.4405	0.4405	0.44386	0.444	0.4369	0.4335	**0.4464**
Ohsumed	0.4453	0.4538	**0.4542**	N/A	0.441	0.4496	0.4429	0.4302
2003_np	0.8025	0.7949	0.797	N/A	0.8018	0.7672	0.7641	**0.8068**
2004_np	0.7918	0.7056	0.7622	N/A	**0.8128**	0.7384	0.7497	0.6914
2003_hp	0.7847	0.8117	0.8084	N/A	0.8372	0.806	**0.8384**	0.8171
2004_hp	0.7529	0.7607	0.7799	N/A	0.7845	0.8057	**0.8328**	0.7428
2003_td	**0.353**	0.3474	0.3457	N/A	0.3484	0.3036	0.3069	0.3122
2004_td	0.2909	0.2985	0.302	N/A	0.3175	0.3163	0.3285	**0.3504**

Precision10

Algorithm	Domination Rank			RankSVM	ListNet	AdaRank		RankBoost
Dataset	Layered	Flat	Margin	Struct		NDCG	MAP	
Letor4-MQ2007	0.3767	0.3776	0.3776	0.38332	0.3798	0.3756	0.3738	**0.3862**
Ohsumed	0.504	**0.5088**	0.5079	N/A	0.4975	0.5087	0.4976	0.4966
2003_np	0.0933	0.0927	0.0933	N/A	0.0927	0.0893	0.0867	**0.094**
2004_np	**0.0947**	0.0867	0.092	N/A	**0.0947**	0.0907	0.088	0.088
2003_hp	0.0993	0.1033	0.1033	N/A	**0.1067**	0.1	0.106	0.1053
2004_hp	0.0973	0.0973	**0.0987**	N/A	**0.0987**	0.096	0.0947	0.092
2003_td	0.19	0.192	0.19	N/A	**0.2**	0.164	0.158	0.17
2004_td	0.236	0.2333	0.2413	N/A	0.256	0.248	0.2493	**0.2747**

because the increased sensitivity to the top training instances increases the model's reliance on these being highly representative instances. Note that it does much better than the others for precision at top 10 and NDCG at the top 10, but not for NDCG at the top 5.

See Fig. 4 for detailed results. Domination Rank performs similarly to existing algorithms. As shown in Fig. 5, which shows the average ranking of each algorithm, Domination Rank generally exceeds the

Figure 5: Averaged performance on the Microsoft LETOR Datasets. Domination Rank is shown against published results from other algorithms. Layered denotes the domination rank algorithm with cross-validated regularization and multiple layers. Flat denotes a bilayer formulation, and Margin a separable margin term, again selected via cross-validation. Results are measured by normalized discounted cumulative gain (NDCG) at the top five and top ten instances per query, and by precision at the top ten. For each of these measures, we ran the algorithms against eight datasets and show here their averaged ranks.

Algorithm	Domination Rank			ListNet	AdaRank		RankBoost
Measure	Layered	Flat	Margin		NDCG	MAP	
NDCG 5	3.75	4.25	4.50	2.75	4.75	4.00	4.00
NDCG 10	4.25	4.25	3.62	2.75	4.88	4.12	4.00
Precision 10	3.88	3.88	3.00	2.25	4.88	5.25	3.75

performance of all but ListNet, which was designed for this type of dataset structure. We suspect that Domination Rank might do better if we retained all of the query relations.

8.2 Corel Datasets

The Corel Dataset contains labeled images and gives rise to the task of ranking images according to their correspondence to a label/query. It was used to demonstrate the performance of the PAMIR ranking algorithm. [7]

8.2.1 Structure of Corel Datasets

Within Corel, there are two datasets. The "small" dataset consists of 5,000 images, of which 4,000 are used for training and the remainder are split evenly between validation and testing. The "large" dataset consists of 35,379 images, of which 14,861 are used for training and the remainder are split evenly between validation and testing.

8.2.2 Domination Rank Experiments with the Corel Datasets

We used the same set of features and the same partitioning of datasets used in Bengio and Grangier. [7] Whereas Bengio and Grangier investigated performance against word models, we instead look at query models. While word models offer greater coverage of queries, training word models effectively requires training all the models together, which makes parallelization nontrivial, and thus is not feasible on large problem sizes for either PAMIR or Domination Rank. In each case, then, we learned a model for each query/label, where that query was present in the training, test, and validation sets. Regularization type and weight were chosen via cross-validation for each query independently. Given the resulting models, we

Figure 6: Average Precision on the Corel datasets. Both "Feature Induction" and "Sampled" use feature induction; "Sampled" uses only 10% of the negative instances. "Sequential" is the basic algorithm. All use regularization. We also show the difference in PAMIR's performance made by running for vastly more iterations. The slower, sequential algorithm achieves the best performance of the Domination Rank variants, but is nearly matched by the much faster variants. The gap between PAMIR and Domination Rank performance is greatest on the dataset (Corel-Large-All) where running for many more training iterations is most useful for PAMIR, which suggests that practical training time is largely responsible for the performance gap.

Algorithm	PAMIR		Domination Rank		
Dataset	40K Iterations	1M Iterations	Feature Induction	Sampled	Sequential
Corel-Small-Easy	0.412	0.4522	0.438	0.411	0.444
Corel-Small-Hard	0.247	0.2425	0.230	0.230	0.234
Corel-Large-All	0.03361	0.0764	0.0577	0.0570	0.0578

recorded each model's performance against its associated query and averaged the results. See Fig. 6.

As in Bengio and Grangier [7], we also investigate the performance on the small dataset when it is paritioned into easy and hard datasets, where easy queries have three or more images in the test set (and hard queries do not). Note that our results for PAMIR differ from those published in Bengio and Grangier, since we use a smaller subset of queries and train for queries instead of words. Thus, we obtain higher precision, but on an easier problem.

Our reported numbers are those on models selected via cross validation for each query. For both algorithms, we attempt to train to convergence. In practice, both approach an asymptote for average precision, but very slowly. Since PAMIR trains much faster, we are able to train it closer to convergence. Thus, PAMIR does a bit better than domination rank, but trains a much denser model. See Figs. 8 and 7. Note that for domination rank we compare training with the standard sequential algorithm as well as with only 10% of the negatives (sampled) and with feature selection. All were trained for the same number of iterations, except for instances of feature selection which finished training early. While sequential training does a bit better when measured by average precision, it does not train as quickly as the other variants. Ultimately, either Domination Rank or PAMIR appears capable of achieving superior performance, given sufficient time to train. In practice, PAMIR trains so much faster that Domination Rank is competitive only in its feature induction variant.

8.3 Google Image Datasets

In order to demonstrate the applicability of domination ranking to large problem sizes, we investigated the algorithm's performance on a much larger image dataset available internally at Google.

Figure 7: Details for Corel-Small-Easy (Averaged across test set). Both "Feature Induction" and "Sampled" use feature induction; "Sampled" uses only 10% of the negative instances. "Sequential" is the basic algorithm. All use regularization. As expected, PAMIR achieves lower all-pairs error, and the basic Domination Rank algorithm performs better than the Feature Induction version (at the cost of training time). Interestingly, PAMIR also achieves superior domination error – most likely as a result of having run for more iterations. Nonetheless, the most striking difference in results is the sparse weight vector achieved for Sampled Domination Rank for relatively little performance cost.

Algorithm	PAMIR	Domination Rank		
Measure	1M Iterations	Feature Induction	Sampled	Sequential
Domination Error	0.7574	0.7658	0.7954	0.7650
All Pairs Error	0.0935	0.1114	0.1176	0.1059
Percent Nonsparse	0.6282	0.4332	0.2303	0.7468

Figure 8: Details for Corel-Small-Easy (Averaged across validation and test sets). Note that errors here are smaller, indicating that the parameters selected in cross-validation were most likely better-targeted to the validation set than to the test set.

Algorithm	PAMIR	Domination Rank		
Measure	1M Iterations	Feature Induction	Sampled	Sequential
Domination Error	0.7580	0.7352	0.7405	0.7332
All Pairs Error	0.0995	0.1096	0.1122	0.1020
Percent Nonsparse	0.6282	0.4332	0.2300	0.7472

8.3.1 Structure of Google Image Datasets

This dataset consists of roughly 2.3 million training images and 400 thousand test images. We selected a random subset of 1,000 of the 41,000 available queries and learned a model for each of these queries. The dataset is similar to Corel, except that it is much larger.

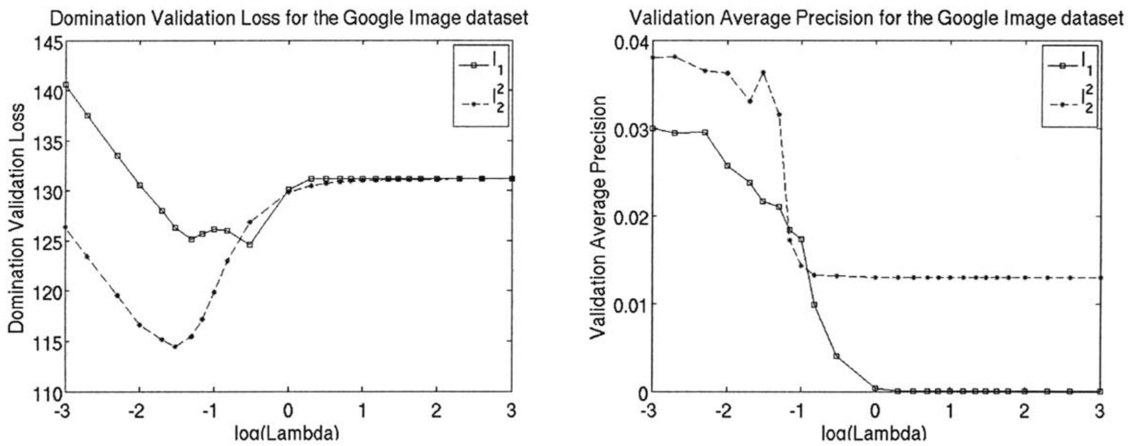
8.3.2 Domination Rank Experiments with Google Image Datasets

Results are presented in Fig. 10. All datasets used sampled negative examples to reduce training time.

Due to the size of the dataset, we did not run full cross-validation across all 1,000 queries. Instead, we selected a subset of ten queries, and divided the training set evenly into training and cross validation. We then tested many values for the regularization weight λ for both ℓ_1 and ℓ_2 regularization. The performance vs lambda is shown in Fig. 9. Although a higher average precision was reached with a lower value for λ, we decided to follow the minimum loss, since the loss results were better-behaved and appear less influenced by noise. Thus, we used $\lambda = 0.03$ and ℓ_2^2 regularization for the full 1,000 query experiment. With this regularization, all of the cross validation models converged and stopped training within 400,000 iterations.

We applied the same procedure to find the optimal regularization for PAMIR, and arrived at $\lambda = 0.03$ for PAMIR as well.

Figure 9: Cross Validation Results for the Google image dataset. Both types of regularization behave as expected for their effect on the domination loss. ℓ_2^2 regularization is more effective, probably since the data is already so sparse that promoting further sparsity through ℓ_1 regularization leads to undertraining. The plot demonstrates overtraining with little regularization and undertraining with too much. Unfortunately, average precision is not well-behaved as a function of regularization, so we used the domination loss results for cross validation of regularization parameters.



In addition to testing the basic model, we also investigated training with layers and with a separable margin. Both margin and layers are generated based on the positive feedback for each image – the more positive feedback, the higher the layer or the larger the margin. Parameters for controlling the effect of these additions were selected via cross validation. For multiple layers, we tried various divisions of feedback into additional layers. These partitions had little effect, but a three-layer paritioning gave a very minor improvement, so we applied it to the full dataset. In the case of margin, there were more parameters to select. We investigated separable margins, as described in Section 4. If r is our feedback, there are a few ways to construct the margin. In all cases, we set the margin $s(j) = 0$ for negative examples j. For positive examples, we investigated the following functions:

- $s(i) = k_1 \log(r_i) + k_2$
- $s(i) = k_1 r_i + k_2$

Cross validation examples performed much better than those of the no-margin variant for $s(i) = \log(r_i) + 2.7$ and $s(i) = r_i * 200$. Of these, the latter showed greater gains (25% higher average precision than the margin-less variant), so it was used for the full test, with results displayed in Fig. 10.

Figure 10: Averaged Results for 1,000 queries on the Google image dataset. Each Domination Rank query was trained for up to four million iterations, but in practice nearly all stopped before 200,000 iterations. Again, all variations except "Sequential" use feature selection. "Layers" attempts to separate positive images into multiple layers, while "Margin" adds a separable margin based on the positive feedback for each image. Note how close Domination Rank comes to PAMIR's performance with less than 10% as many features.

Algorithm	PAMIR		Domination Rank			
Measure	100M Iterations	1B Iterations	Feature Selection	Layers	Margin	Sequential
Average Precision	0.0552	0.0506	0.0495	0.0495	0.0499	0.0500
Precision at Top 10	0.0600	0.0798	0.0731	0.0731	0.0731	0.0731
Domination Error		0.9643	0.9625	0.9640	0.9635	0.9643
All Pairs Error		0.2344	0.2367	0.2353	0.2367	0.2132
Weight Density		0.9663	0.0568	0.0657	0.0676	0.9856

Despite the improvment on the validation set, this margin only had a small effect on the large test. The margin's effect is fairly sensitive to a small number of images – those with high training relevance – so a validation set with above-average high-relevance images should see much more improvement than the dataset as a whole.

It should be noted that the dataset is extremely noisy – many relevant images are mislabeled as negatives and vice versa. Despite the low test precision, many of the incorrectly high-ranked images are actually highly relevant to the query.

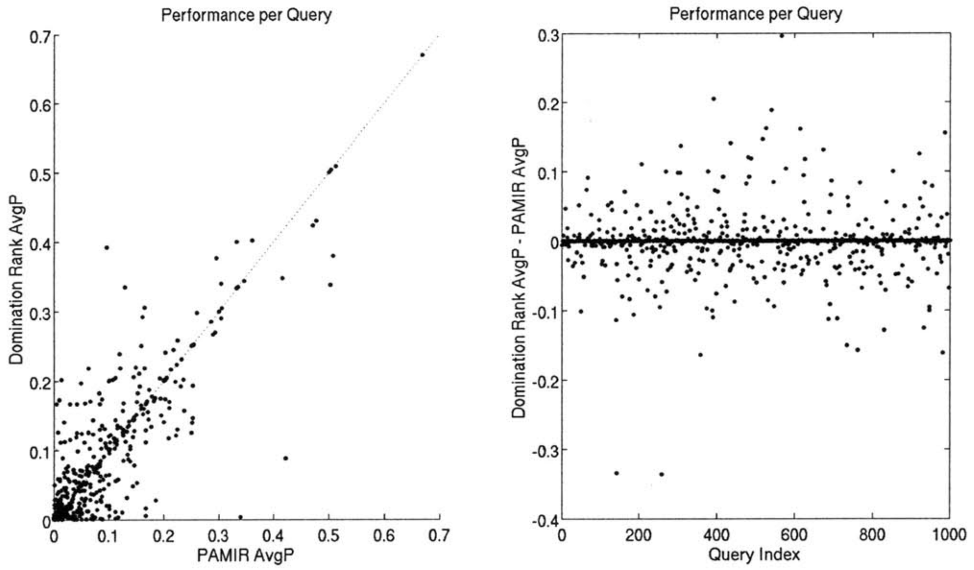
See Figs. 11 and 12 for a query-by-query plot of Domination Rank and PAMIR performance. Note that the algorithms perform roughly the same for many of the queries – suggesting those have been learned as well as is possible given the noise in the dataset – but that for other queries, either algorithm may significantly outperform the other.

8.3.3 Detailed training analysis

In order to gain insight into the effects of training parameters on the resulting model, we investigate the performance of a single representative query in the dataset.

First, we investigate performance vs the regularization weight λ. As one would expect, increasing λ increases performance on the test set (particularly loss) until it grows too large and causes undertraining. Note Fig. 13.

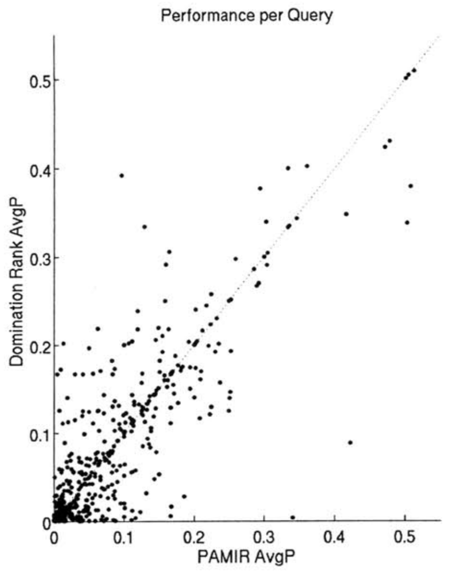
Figure 11: Performance on individual queries in the Google image dataset. Each point represents one query, which may be compared against a line that shows where Domination Rank and PAMIR performance are equivalent. These plots show that there is no clear trend – while the two come out roughly equivalent on average, and many queries are distributed around this equivalence line, a single query may perform much better on one than on another. This distribution may be due to PAMIR's training scheme which favors more queries with more training images – it spends more time on them – and it suggests that there is still room for improvement in both algorithms.



In Fig. 14, we demonstrate that for good values of regularization, training produces a compact model which terminates early. Note the (noisy) hump in iterations vs lambda. Small regularization weights λ do not lead to quick convergence, but a limit on the number of iterations and feature selection still generate sparse models. Fairly large values of regularization yield a small number of features which are quickly optimized; as lambda grows for ℓ_2^2, the number of features to optimize increases, as each feature is allowed less effect. Eventually, a large enough λ yields extreme undertraining for both types of regularization.

Finally, in Fig 15, we looked at the performance over time of a single value of regularization, in order to gain insight into the feature selection process and any overtraining. Here, we note a tendency to overtrain until the rounds of feature induction near 700,000 iterations, most likely because too few features have been selected up to that point. Note that the initial loss for $\boldsymbol{w} = 0$ is not plotted; training makes great progress in the first $100,000$ iterations that are not shown.

Figure 12: Performance on individual queries in the Google image dataset. The same plot from Fig. 11 is shown, but outliers are removed to show the majority of the points in greater detail.



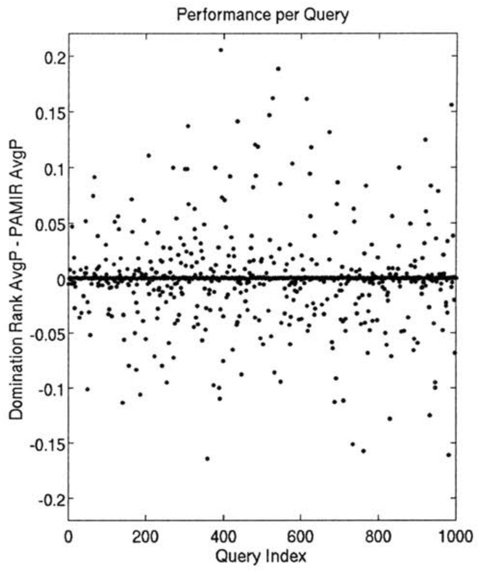
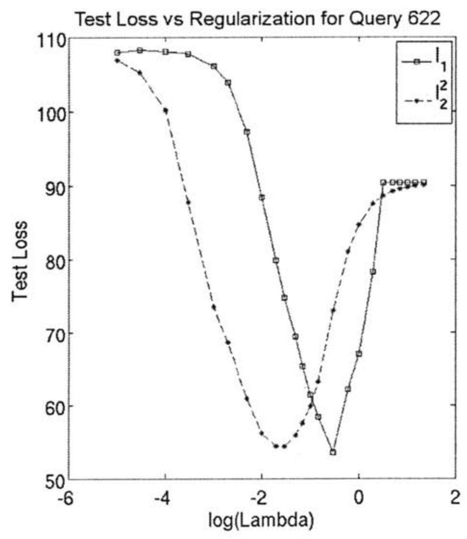
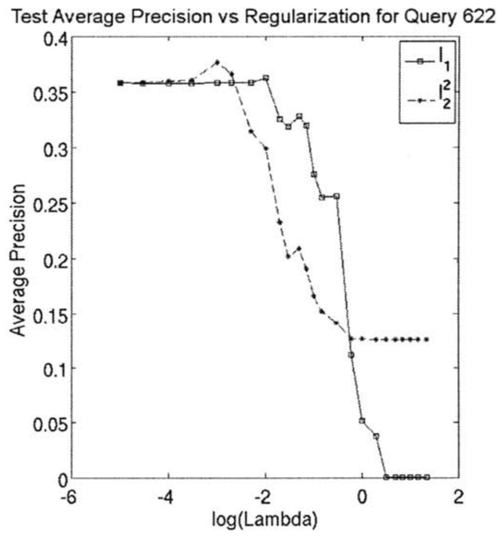


Figure 13: Detailed Performance for a single query (622) in the Google images dataset. We see the familiar performance vs λ.





8.4 Reuters Document Ranking

Finally, we addressed the problem of ranking documents based on topic in the Reuters RCV1 dataset. [1]

Figure 14: Detailed Performance for a single query (622) in the Google images dataset. We plot weight density and training iterations needed for convergence against λ. With low values of regularization and feature selection, we achieve sparse models. As regularization increases, the ℓ_1 regularized model selects even fewer features; the ℓ_2^2 model starts to train every feature, but only slightly. In both cases, higher regularization yields earlier stopping points (until ℓ_2^2 starts training every point and slows back down) – there is a sweet spot where we converge quickly before reaching the point of undertraining.

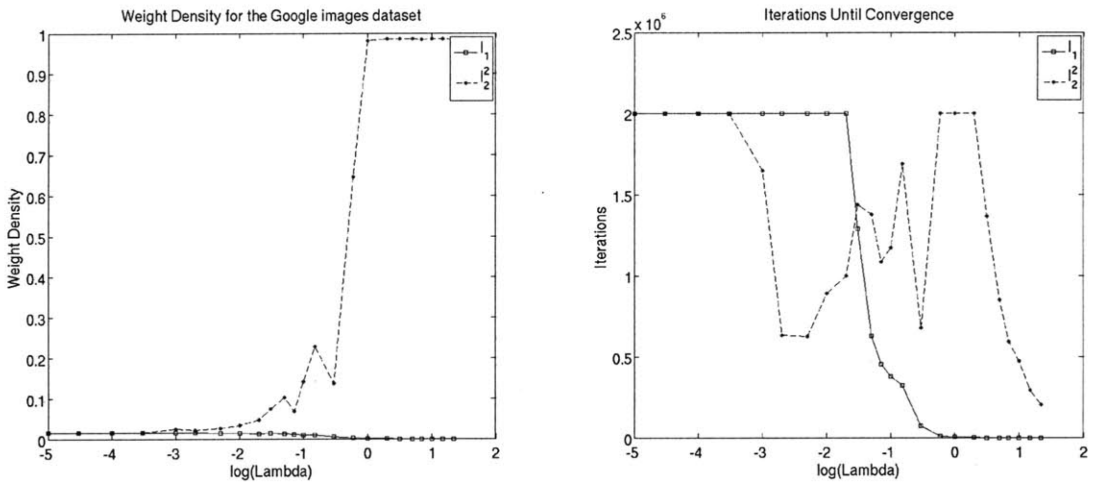
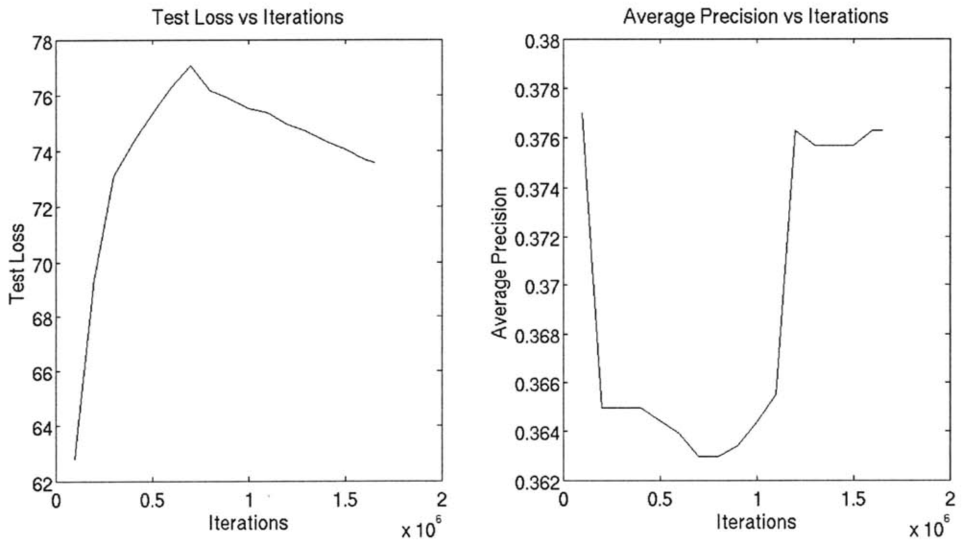


Figure 15: Detailed Performance for a single query (622) in the Google images dataset. Here, we plot the performance of the model over the course of training with the value of λ which yielded optimal average precision.



8.4.1 Structure of the Reuters Dataset

The Reuters (RCV1) dataset consists of roughly 800,000 news articles from a one-year period, each classified into one or more of 103 topics. Most are described by at least two topics; many by three or more. These topics are hierarchical – for example, there is a general economics category (ECAT) and many subtopics under it. Associated with each article is the raw text of the article and a set of metadata that includes title, date published, etc. Articles are fairly evenly split between the four top categories. Thus, the task of assigning articles in this dataset to topics presents a wide range of ranking tasks – some have much more data and are much simpler than others. In all cases, though, the articles are hand-labeled, and thus there is much less label noise than in the prior image experiments.

8.4.2 Application of Domination Rank to the Reuters Dataset

We investigated a couple of different feature sets given this dataset: first, a set of bigrams and associated appearance counts, and secondly a set of unigrams with features generated as per Crammer and Singer 2003 [5] and Singhal et al 1996 [12]. In each case, we used only the raw text of each article as input and attempted to find the most relevant articles for each topic.

In order to generate the Singhal features, we applied the following process:

1. Convert upper case characters to lower case

2. Replace non-alphanumeric characters to whitespace

3. Generate a dictionary of all words appearing twice or more. This yielded 225,329 words.

4. Discard the GMIL section, which has too few documents

5. Generate numeric feature values from the words. Each feature x_ij for word or feature j and document i is computed using the term frequency tf_j^i and the inverse document frequency of a term, idf_j,

   $$x_ij = idf_j * tf_j^i$$

   $idf_j = \log(m/r_j)$, where m is the total number of documents and r_j is the number of documents containing word j.

   $$tf_j^i = \frac{(1 + \log(d_i^j)/(1 + \log \hat{d}_i)}{1.0 - slope + slope \times (m_i/\hat{\mu})} \quad ,$$

   where m_i is the number of unique words in document i, d_i^j is the number of times word j appears in document i, $\hat{d}_i$ is the average frequency of terms in document i, and $\hat{\mu}$ is the average number of unique terms in each document in the corpus. As in [5] and [12], we set $slope = 0.3$.

Figure 16: Averaged Results for 101 topic queries in Reuters-2000. Performance was calculated for each topic model and these results were averaged. Note that the better features yield much better performance (and of course are also easier to train, as there are fewer of them). Note that Domination Rank and PAMIR perform similarly – though PAMIR is slightly better when measured by Average Precision. Domination Rank, however, does so with extremely sparse models.

4 million bigram features

Algorithm Dataset	PAMIR 1M Iterations	Domination Rank Feature Selection
Average Precision	0.4013	0.4832
Precision at Top 10	0.3505	0.7735

225,000 Singhal features

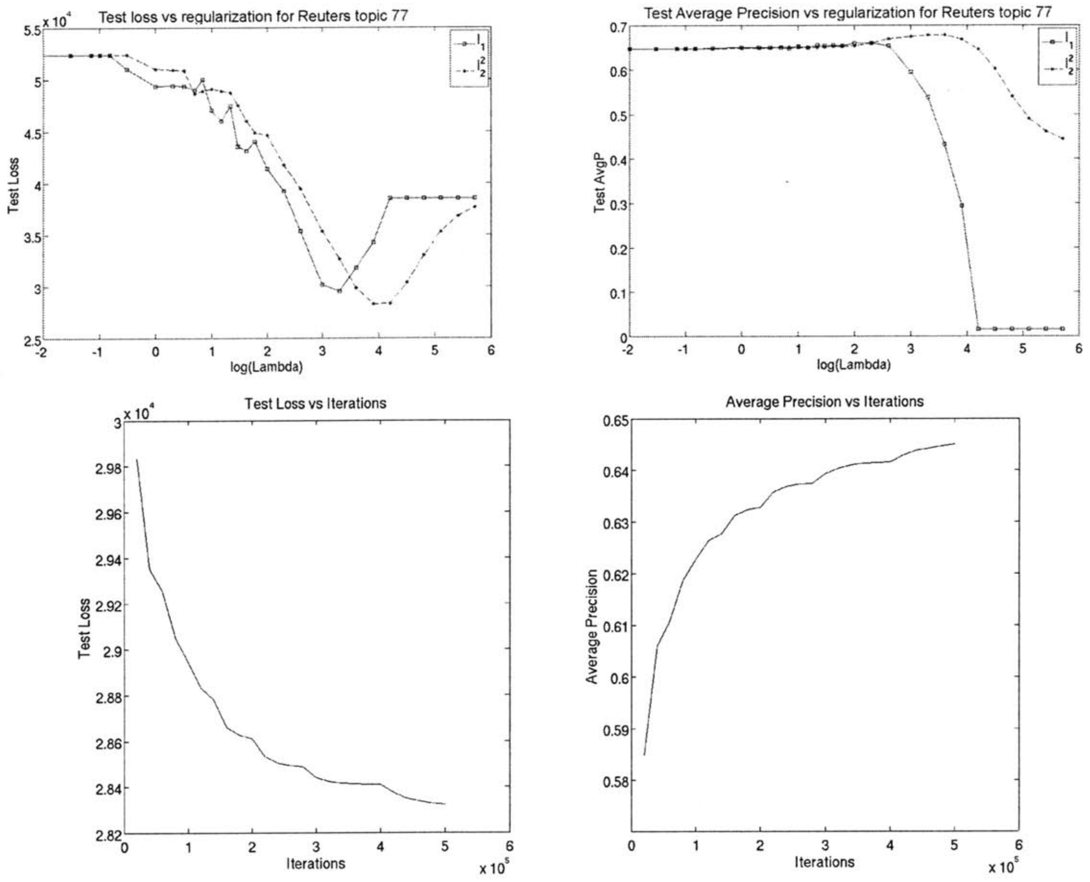
Algorithm	PAMIR		Domination Rank (200k Iterations)
Dataset	10M Iterations	100M Iterations	Feature Selection
Average Precision	0.625	0.705	0.670
Precision at Top 10	0.881	0.915	0.918
Domination Error		0.9738	0.9757
All Pairs Error		0.0137	0.0243
Weight Density		0.2189	0.0018

We randomly parititoned the dataset, with half of the images in a training set and the remainder evenly split between validation and test. Of the 103 topics in the dataset, two had too few topics to provide meaningful results, so we excluded these. Again, we sampled the negative examples, using only 10% of them. For each of the other 101 topics, we learned a model and scored it against the test set; the results averaged across topics are available in Fig. 16. Given the size of this dataset, Domination Rank with Feature Induction was the only reasonable variant to apply (and it had been shown to perform very well as measured by performance vs training time in prior experiments).

As we did with the Google image dataset, we selected a subset of seven queries on which to perform cross validation. Here, however, the minimum loss occurred well after a stable decreasing trend in average precision, so we instead selected λ to maximize average precision. For both domination rank and PAMIR, that selected value was $\lambda = 200$. We show in detail the performance of the algorithm across regularization and over the course of training for a single query in Fig. 17.

Note the large difference in performance between the two types of features. The four million bigram features offered an experiment in ranking with far more features than any of the other experiments, but as seen from the results, a smaller number of more processed features were much more effective at yielding a model with good precision on the test set. In these experiments, the performance difference between the two versions of domination rank is more relevant, since PAMIR was not trained as close to convergence on the

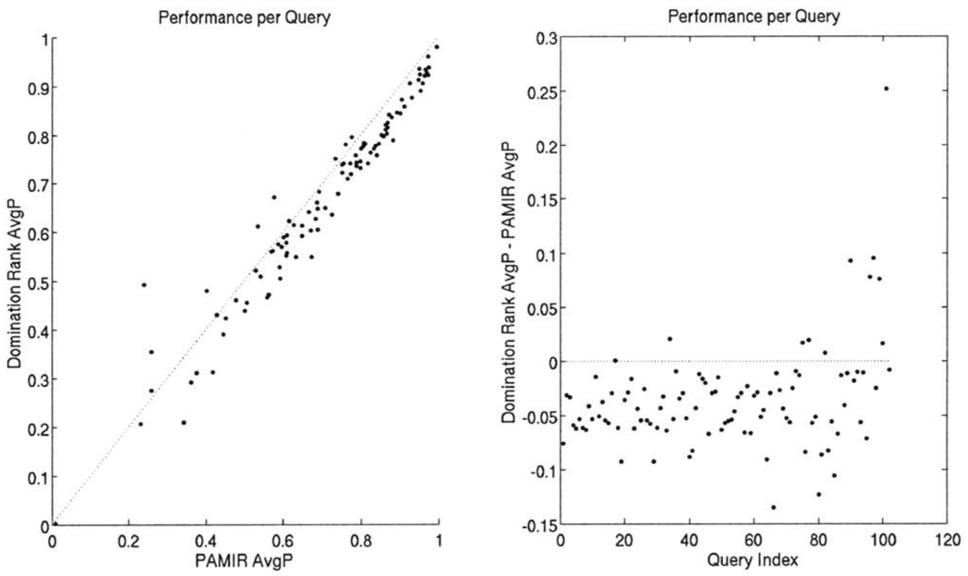
Figure 17: Detailed Performance for a single query (77) in Reuters-2000 with Singhal features. The first two plots depict performace vs regularization; the second two illustrate in detail the course of training for the best regularization. The successive rounds of feature selection are visible in these graphs – good performance is reached very quickly, even with far fewer iterations than there are features. Each additional set of selected features has less effect over a longer set of iterations.



bigram dataset. Indeed, after brief experiments demonstrating how much more effective the Singhal features are, we abandoned the bigram features. Clearly, these algorithms are capable of handling enormous numbers of rough features, but much more easily learn models with a smaller number of more normalized features – perhaps in part because of the greater ease in converging with fewer features to learn.

Once again, we were able to train PAMIR a bit closer to convergence, since it trains faster, so it gets slightly better performance. We were arbitrarily able to shift the balance one way or the other by increasing training time, but eventually cut short the training time due to diminishing returns and long time for experiments (especially with Domination Rank). In Fig. 18, we show performance on each topic. Unlike with the Google Image datset, there is a clear trend here – PAMIR does slightly better with most of the

Figure 18: Performance on individual queries in Reuters-2000. PAMIR does better on almost all queries as a result of faster (and thereby more thorough) training. Note a trend in the plot on the right – topic frequency decreases as topic index increases. Domination Rank generally does better on these less-frequent topics. Since our Domination Rank implementation trains all topics uniformly, while PAMIR focuses on the more common ones, the topic models to the right have been trained relatively more thoroughly by Domination Rank.



queries. The trend towards better relative Domination Rank performance on higher topic indices (those with fewer examples) suggests that training time is again responsible for the difference, since the models for these topics are trained relatively more thoroughly in Domination Rank. Interestingly, the performance gap narrows a bit on the highest-performing topics. This trend is reasonable, since easier topics should not need as much training time and are therefore less affected by PAMIR's faster training. Not surprisingly, PAMIR achieves lower all-pairs error, and both have very high domination error – completely satisfying domination groups with so many instances is extremely difficult. Both algorithms do very well with precision at top 10, but PAMIR does a bit better when measured by average precision. In practice, though, we expect the top results to be the more important part of the ranking output for applications.

Again, we see a large difference in the number of features needed for Domination Rank models as compared to PAMIR. In the Domination Rank model, over 99.8% of features are zero. In the PAMIR model, only 80% of the features are zero. Given trends from other experiments, it is likely that the percentage of nonzero features in the PAMIR model would decrease as the number of iterations (and the associated number of features examined) increased. Yet, Domination Rank does almost as well with only 1% as many features –

it is extremely effective at identifying the most relevant features. While it might be possible to formulate a version of PAMIR that also achieves such sparsity (the formulation shown here uses ℓ_2^2 regularization which does not promote sparsity), Domination Rank has an intrinsic advantage at learning sparse models, since it considers the effect of all instances of a given feature before deciding whether to use that feature, and skips the less useful ones. A pair-by-pair examination of features like what PAMIR does could easily exclude an extremely useful feature apparent in all pairs, but only marginally useful in any single pair. Thus, Domination Rank's consideration of every appearance of a feature, while responsible for much of the longer training time, is extremely useful whenever a sparse model is desired. For examples of the progress of training with feature selection, see Fig. 17. Note that the top features have a very quick, very substantial impact on training.

9 Conclusions

We derive a coordinate descent algorithm for finding a ranking based on the Domination Error which Dekel et al. [10] showed to be a promising candidate for a ranking algorithm. We find a convex relaxation of that error and associated updates which scale linearly with the number of training instances, even for many layers of training data labels.

Given this basic algorithm, we derive and try experimentally a number of variations. Multilayer training allows the algorithm to handle datasets where we have more detailed feedback as to which instances are most relevant. A separable margin gives us a similar ability to increase the influence of certain (high margin) instances. ℓ_1 and ℓ_2^2 regularization, one of which is used in every experiment, vastly increases the algorithm's robustness against noise. Feature induction speeds up training by reducing time spent on mediocre features and yields very sparse models with performance almost as good as that of dense models. We show that most of each update may be stored from previous rounds, again increasing the speed of training, and demonstrate the stability of the coordinate descent updates.

We apply the algorithm to a number of datasets. On the LETOR dataset, we find that Domination Rank performs similarly to existing algorithms at ranking a small dataset of web documents with few features. Using multiple layer labels, when such labels are available, slightly increases performance. With the Corel and Google Images datasets, Domination Rank performs similarly to the highly effective PAMIR algorithm, but with longer training times and ultimately more compact models. Feature induction and sampling negative instances greatly decrease training time, while greatly increasing model sparsity and slightly decreasing performance. We apply these lessons to experiments with the Reuters RCV1 dataset, which provides a trial on a large dataset with far more features than prior datasets. PAMIR and Domination Rank perform well, especially with Singhal's features, and demonstrate effective and extremely sparse models learned by Domination Rank.

References

[1] Reuters corpus vol 1. 2000. Available at http://about.reuters.com/researchandstandards/corpus/.

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

[3] Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Association for Computational Linguistics (ACL)*, pages 489–496, 2002.

[4] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. In *International Conference on Machine Learning (ICML)*, 2000.

[5] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. *J. Mach. Learn. Res.*, 3:1025–1058, 2003.

[6] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. *NIPS*, 14, 2001.

[7] David Grangier and Samy Bengio. A discriminative kernel-based model to rank images from text queries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1371–1384, August 2008.

[8] H. Li J. Xu. Adarank: A boosting algorithm for information retrieval. *SIGIR*, 2007.

[9] G. Lebanon and J. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[10] C. Manning O. Dekel and Y. Singer. Log-linear models for label ranking. In *Neural Information Processing Systems (NIPS)*, 2003.

[11] Robert Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. 1999.

[12] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–29, New York, NY, USA, 1996. ACM.

[13] T. Qin W. Xiong T. Liu, J. Xu and H. Li. Letor: Benchmark dataset for research on learning to rank for information retreival. *SIGIR*, 2007.

[14] R. Schapire Y. Singer Y. Freund, R. Iyer. An efficient boosting algorithm for combining preferences. 2003.

[15] et al Z. Cao. Learning to rank: From pairwise approach to listwise approach. *ICML*, 227.