Internet Mathematics Vol. 4, No. 2–3: 267–298

Learning to Rank in Vector Spaces and Social Networks

Soumen Chakrabarti

Abstract. We survey machine learning techniques to learn ranking functions for entities represented as feature vectors as well as nodes in a social network. In the feature-vector scenario, an entity, e.g., a document x, is mapped to a feature vector $\psi(x) \in \mathbb{R}^d$ in a d-dimensional space, and we have to search for a weight vector $\beta \in \mathbb{R}^d$. The ranking is then based on the values of $\beta \cdot \psi(x)$. This case corresponds to information retrieval in the "vector space" model. Training data consists of a partial order of preference among entities. We study probabilistic Bayesian and maximum-margin approaches to solving this problem, including recent efficient near-linear-time approximate algorithms. In the graph node-ranking scenario, we briefly review PageRank, generalize it to arbitrary Markov conductance matrices, and consider the problem of learning conductance parameters from partial orders between nodes. In another class of formulation, the graph does not establish PageRank or prestige-flow relationships between nodes, but encourages a certain smoothness between the scores (ranks) of neighboring nodes. Some of these techniques have been used by Web search companies with very large query logs. We review some of the issues that arise when applying the theory to practical systems. Finally, we review connections between the stability of a score/rank-learning algorithm and its power to generalize to unforeseen test data.

1. Motivation for Learning to Rank

In traditional information retrieval (IR), ranking techniques matured over more than a decade. During that time, the basic vector space model [Salton and McGill 83] remained a known constant and scoring functions were designed by hand. In the very early days of hyperlink-based ranking for the Web [Page

1542-7951/07 $0.50 per page

et al. 98], ranking functions continued to be designed and tuned by hand, because relatively few sources of ranking information were taken into account.

1.1. Maintaining Search Engine Ranking Functions

Search systems have become very complex compared to classic IR systems and since the early days of link-assisted ranking on the Web. Web search engines now use dozens to hundreds of features [Richardson et al. 06] from pages and sites, and possibly many other features, extracted from the query or query session, source location, browser cookies, and possibly user profile information. Pages are often scanned by entity tagging systems to identify and canonicalize entities mentioned therein. The extracted entities are then connected through a knowledge base, which is often graph-structured (e.g., location or product hierarchies). The increasing complexity of ranking logic makes manual tuning difficult; while "fixing" the ranking function to eliminate a few wrinkles, it is possible to worsen responses for millions of queries. Semi-automated, learning-based maintenance of ranking functions is now essential.

Traditional IR evolved several approaches to adapting the ranking function via *relevance feedback*—users expressing opinions about the ranking systems used to tune the ranking engine automatically [Salton and Buckley 99]. It is only recently that the relevance feedback task was performed in rigorous ways from the perspective of machine learning [Herbrich et al. 99, Joachims 02, Burges et al. 05]. Using some form of supervised rank-learning in the presence of graphical structure is even more recent [Agarwal 06, Vassilvitskii and Brill 06, Agarwal et al. 06, Agarwal and Chakrabarti 07].

1.2. Searching and Mining Social Networks

Learning ranking functions is becoming more important because more diverse forms of data are being indexed by search systems that must rank responses to queries. A graph in which nodes (and possibly edges) have associated text, and in which nodes and edges have types (e.g., nodes representing the types *person* and *organization*, and an edge of type "works-for"), form a convenient and powerful common denominator representation for the object exchange model [Papakonstantinou et al. 95], XML, and even relational data [Bhalotia et al. 02, Agarwal et al. 02, Balmin et al. 04]. We collectively call these *entity-relationship graphs* or E-R graphs. An example is shown in Figure 1.

E-R graphs are a rich representation of semistructured data. The flexibility also makes ranking difficult: Nodes and edges have diverse semantics and are not all equally important. Their importance may even vary in the course of a single query session. This is in stark contrast to traditional IR, where the

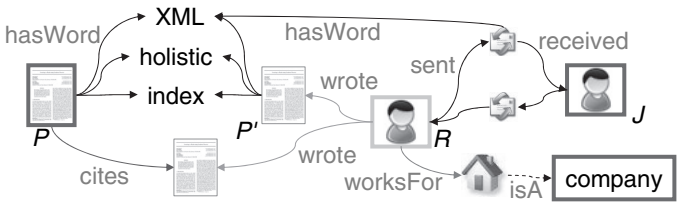


Figure 1. Entity-relationship graph. A personal search application may include the submitted paper P, the journal editor J, and the desirability of a reviewer from industry ("company"), and then be required to rank reviewers like R.

"schema," in the parlance of E-R graphs, has been fixed: The corpus is a set of documents, each a sequence of tokens. Consequently, there is no single successful ranking function for general E-R graph search applications. Recent years have seen several efforts to automate the design of ranking functions [Diligenti 05, Nie et al. 05, Chakrabarti and Agarwal 06] for E-R graphs.

Summarizing, scoring and ranking applications are becoming too complex for completely manual tuning and maintenance, and, increasingly, there is emphasis on supplementing editorial judgment with automatic learning of scoring and ranking functions.

2. Preliminaries

We set up some notation. An *instance* is denoted $x \in \mathcal{X}$ if it is interpreted as a feature vector $\psi(x) \in \mathbb{R}^d$. Sometimes we will omit $\psi(\cdot)$ and just write x as the feature vector as well. If an instance is a node in a graph, as will be the case later in this survey, it will be denoted as a *node* u, v, etc.

This section has two subsections. In Section 2.1, we describe two training and evaluation scenarios for which there are reasonably direct learning algorithms. In Section 2.2, we describe other, practically motivated training and evaluation scenarios, for which satisfactory direct learning algorithms are not known.

2.1. Simple Training and Evaluation Scenarios

We consider two common methods of training and evaluation here. In the first, there is no explicit notion of absolute quality of the items being ranked: The trainer can only compare pairs of items and assert that one is better or worse than the other. Ideally, we should not assume global consistency (acyclicity) in the resulting preference graph. In the second setup, the trainer assigns scores to items, but on a coarse-grained scale, as in movie reviews or Amazon.com product

reviews. Items in the same bucket cannot be compared, but items across any pair of buckets can be compared meaningfully.

In the remainder of this section, we review a number of different evaluation methods for ranking functions. We will see in the rest of the paper how these evaluation criteria drive the development of methods to obtain good scoring and ranking functions.

2.1.1. Pair preferences and their violations. For some applications, it is difficult or meaningless to talk about an absolute quality of training instances. For instance, a trainer may only be able to compare pairs of instances but not be able to assign each instance an absolute score. If the user prefers instance u less than instance v, we will denote that fact by "$u \prec v$." We will overload $\prec$ also to denote the set of preferences, i.e., use $\prec$ as both a relation and a set. In an application with multiple trainers, the directed graph induced by $\prec$ can even have cycles. Ideally, learning algorithms should be robust enough to tolerate inconsistent preferences.

Even though the training process presents the learning system with instance pairs, at testing time almost all algorithms use an internal scoring system to map each instance to a real *score*, and use the score to impose a total order on instances.

If the training input is denoted $\prec_{\text{train}}$, a separate $\prec_{\text{test}}$ is used to evaluate the trained system, and the test error is simply the fraction or preference pairs in $\prec_{\text{test}}$ that are violated by the model produced by the trained system [Joachims 02].

Counting the number of violated preference pairs is very similar to counting the number of misclassified instances in traditional classification. The total loss over a training set can be written as a simple sum of 0/1 indicators of violation, one for each preference pair. This makes both learning optimization and analysis relatively simple.

2.1.2. Average precision. In other applications, an ideal total order may be available, and we may wish our ranking algorithm to produce a total order to be evaluated against the ideal one.

Consider a text search system and a query for which there are R relevant documents. A search engine creates a ranking r_{engine} that lists them at ranks $p_1 < p_2 < \cdots < p_R$. The *average precision*[1] of r_{engine} wrt r_{ideal} is defined to be

$$\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}}) = \frac{1}{R} \sum_{i=1}^{R} \frac{i}{p_i}.$$

[1] http://en.wikipedia.org/wiki/Information_retrieval#Average_precision

Observe that average precision rewards the search engine if all p_i are as small as possible.

An ideal system creates a ranking r_{ideal} that lists all relevant documents before any irrelevant document but keeps the relative ordering within the relevant and irrelevant subsets the same. For example,

$$r_{\text{engine}} = d_1^+, d_2^-, d_3^+, d_4^+, d_5^-, d_6^-, d_7^+, d_8^-,$$
$$r_{\text{ideal}} = d_1^+, d_3^+, d_4^+, d_7^+; d_2^-, d_5^-, d_6^-, d_8^-.$$

Across the two strict (meaning that ties have somehow been resolved) rankings, documents d_i and d_j are *concordant* if they are in the same order in the two rankings, and *discordant* otherwise (see Section 2.2.1).

Account for the number of discordant pairs Q as follows: First consider the relevant document at position p_1 in r_{engine}. Because it has been moved from position 1 to position p_1, the number of inversions introduced is $p_1 - 1$. For the document at position p_2 in r_{engine}, the number of inversions introduced is $p_2 - 1 - 1$, the last "-1" thanks to having the first relevant document ahead of it. Summing up, we get

$$\sum_{i=1}^{R} p_i - 1 - (i-1) = Q, \quad \text{or}$$
$$\sum_{i=1}^{R} p_i \;=\; Q + \sum_{i=1}^{R} i = Q + \frac{R(R+1)}{2} \;=\; Q + \binom{R+1}{2}.$$

Intuitively, if Q is small, $\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}})$ should be large. This can be formalized [Joachims 02] by framing an optimization problem that gives a lower bound to $\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}})$ given a fixed Q (and R): $\min_{p_1,\ldots,p_R} \frac{1}{R}\sum_{i=1}^{R} \frac{i}{p_i}$ such that $p_1 + \cdots + p_R = Q + \binom{R+1}{2}$, $1 \le p_1 < p_2 < \cdots < p_R$, where $p_1, \ldots, p_R$ are positive integers.

Relaxing the last two constraints can only decrease the optimal objective, so we still get a lower bound. The relaxed optimization is also convex because $1/p_i$ is convex in p_i, as far as p_i is concerned the numerator i is a "constant," and the sum of convex functions is convex.

Solving the relaxed optimization using the Lagrangian method, we get

$$\mathcal{L}(p_1, \ldots, p_R; \lambda) = \frac{1}{R}\sum_{i=1}^{R} \frac{i}{p_i} + \lambda \left(\sum_{i=1}^{R} p_i - Q - \binom{R+1}{2} \right).$$

Differentiating,

$$\frac{\partial \mathcal{L}}{\partial p_i} = -\frac{i}{R p_i^2} + \lambda \stackrel{\text{set}}{=} 0 \quad \text{to get} \quad p_i^* = \sqrt{\frac{i}{R\lambda}}.$$

Substitute this in the Lagrangian, set the derivative wrt λ to zero, and again substitute in the Lagrangian to get the optimal objective (in the relaxed optimization) as

$$\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}}) \geq \frac{\left(\sum_{i=1}^{R} \sqrt{i}\right)^2}{R\left(Q + \binom{R+1}{2}\right)}.$$

Q and the lower bound on average precision are inversely related, which makes sense. The above analysis shows that reducing Q (which is easier to tackle using learning algorithms) will increase average precision.

Recently, however, Yue et al. [Yue et al. 07] have given a direct max-margin optimization of the average precision objective.

2.1.3. Ordinal regression, bipartite ranking, and AUC. In many applications, items are assigned *ratings* on a discrete k-point scale; items for sale at Amazon.com, for example, are rated on a five-point scale. This imposes a partial order on instances. We will represent ordinal regression problems as the problem of regressing an instance $x \in \mathcal{X}$ to a label $y \in \mathcal{Y}$, where $\mathcal{Y}$ typically has a small size k. A special case of ordinal regression is *bipartite ranking*, where $k = |\mathcal{Y}| = 2$, so we can write $\mathcal{Y} = \{-1, +1\}$.

At this point, ordinal regression may look like plain k-class classification, but the performance measures are different. Unlike in classification, where labels in $\mathcal{Y}$ are *incomparable*, here they have a total order imposed on them. (In standard regression, $\mathcal{Y} = \mathbb{R}$.) Second, even for two-level ordinal regression (also called bipartite ranking), the number of relevant documents is typically very small compared to the number of irrelevant documents, so it is not enough to reduce a misclassification objective as in classification. We elaborate on the latter issue below. The former is handled in Section 3.3.

In the special case of bipartite ranking, there is a close connection between pair preference violation and the area under the receiver operating characteristics (ROC) curve (area under the curve, or AUC). First we define the ROC curve.

As mentioned before, we assume the algorithm assigns to instance x a *score* $f(x)$, which we threshold suitably to assign instances to either the $+1$ or the -1 class, i.e., let the predicted class be $\text{sign}(f(x) - b)$, where b is the threshold. Ideally, f should be chosen/trained such that all the positive instances have greater $f(\cdot)$ than any negative instance.

To perform a recall-precision type evaluation of f, the usual procedure is to start from $b \to -\infty$, for which all instances are labeled $+1$, and increase b until all instances are labeled -1. The number of interesting values of b is $1 + n$, where there are n instances. Let these values of b be called $b_0, \ldots, b_n$. For each

$i = 0, \ldots, n$, we get a different "classifier" h_i. Each of these classifiers has a *false positive rate* fpr_i and a *true positive rate* tpr_i, defined as

$$fpr_i = \frac{\text{number of negative instances misclassified as positive by } h_i}{\text{number of negative instances}},$$
$$tpr_i = \frac{\text{number of positive instances classified as positive by } h_i}{\text{number of positive instances}}.$$

The ROC curve is a scatter plot of (fpr_i, tpr_i) as i varies, connecting points corresponding to consecutive is. As i increases, b_i increases, and therefore tpr_i and fpr_i decrease.

If the learning system randomly guesses the label of instances, the slope of the curve should be roughly constant. In expectation, it would be the diagonal from $(0, 0)$ to $(1, 1)$, and the area under the ROC curve would be $1/2$. In contrast, if the learning system is perfect, it places all positive examples ahead of any negative example. Therefore, there is a threshold b_i such that fpr_i is very small, but tpr_i is large (close to 1). The AUC would be closer to 1 as well. Thus, the AUC is a reasonable measure of bipartite ranking performance [Agarwal et al. 05].

Suppose the learning algorithm has trained a function $f : \mathcal{X} \to \mathbb{R}$. Suppose a sample contains n_+ positive and n_- negative instances. Then it can be shown that the AUC is also equal to

$$\hat{A}(f, S) = \frac{1}{n_- n_+} \sum_{i:y_i=+1} \sum_{j:y_j=-1} \left([\![f(x_i) > f(x_j)]\!] + \frac{1}{2} [\![f(x_i) = f(x_j)]\!] \right) \quad (2.1)$$

Here $\hat{A}$ is the *empirical* accuracy over the labeled set S using the scoring function f. Therefore, there is a direct connection between AUC and the fraction of pairwise preferences that is satisfied ($j \prec i$ is satisfied if $f(x_i) > f(x_j)$).

2.2. Other Real-Life Training and Evaluation Scenarios

In Section 2.1, we reviewed objectives that are decomposable over instances or instance pairs, and are therefore relatively easy to target by learning algorithms. There are other ranking objectives that are important in applications; we discuss some of them here.

First we set up some notation. Fix a query and suppose we somehow know the "perfect" instances that belong to the top k rank positions. Let this sequence be S^k, and suppose the "perfect" scores of these instances are known. Let us denote the perfect score of instance v as $S^k(v)$, overloading S for convenience. The learning algorithm, meanwhile, returns a sequence $\hat{S}^k$, with score $\hat{S}^k(v)$ for instance v.

2.2.1. Rank correlation. In many ranking applications, perceived ranking quality depends strongly on the instances listed at the top of the ranked list. In Web search, the density of clicks on query response lists provided by search engines decays steeply with rank [Cho and Roy 04]. Ranking mistakes made far into the response list are easily overlooked or forgiven, while mistakes made in the "top 10" are damaging.

One way to capture this using S^k and $\hat{S}^k$ is to use rank correlation. Consider all instances in $S^k \cup \hat{S}^k$. If some v in the union is not in S^k, set $S^k(v) = 0$, and similarly for $\hat{S}^k(v)$. A node pair $v, w \in S^k \cup \hat{S}^k$ is *concordant* if $(S^k(v) - S^k(w))(\hat{S}^k(v) - \hat{S}^k(w))$ is strictly positive, and *discordant* if it is strictly negative. It is an *exact tie* if $S^k(v) = S^k(w)$, and is an *approximate tie* if $\hat{S}^k(v) = \hat{S}^k(w)$. If there are c, d, e and a such pairs, respectively, and m pairs overall in $S^k \cup \hat{S}^k$, then Kendall's τ is defined as[2]

$$\tau(k) = \frac{c - d}{\sqrt{(m - e)(m - a)}} \in [-1, 1].$$

One problem with using rank correlation to evaluate search engines is that training data, in the form of the total order S^k, is nearly impossible to gather on a large scale. For a Web query, a trainer may express a reliable opinion on the absolute quality of a response in an ordinal regression setting, or compare pairs of responses reliably as well, but to claim that a given instance u should be listed at rank 4 means that the trainer is sure that out of, say, 20 billion accessible Web pages, exactly three are better than u, which is unrealistic.

2.2.2. Precision at top-k. It is more realistic to interpret S^k and $\hat{S}^k$ as sets instead of sequences, each of size k. The precision at top-k is defined to be $|S^k \cap \hat{S}^k|/k$, a score in $[0, 1]$. Clipping the evaluation at some small k (usually between 10 and 100) is reasonable, because, in applications, users are generally not adversely affected by erroneous ranking lower in the ranked list. Note that the merit of the ranking algorithm is now judged according to how many items in $\hat{S}_k$ are known to be relevant, not the specific order in which they are presented.

2.2.3. Relative aggregated goodness. Precision can be somewhat severe, especially if there are many ties in score. In the case of ties, to achieve high precision, the learning algorithm must, by some kind of magic, also learn the way the "reference" system breaks ties, which may be quite arbitrary. Listing a different set of instances with (reference) scores almost as good as the "true" top-k instances would still yield terrible precision.

[2] http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient

The relative aggregated/average goodness (RAG) measure [Fogaras et al. 05] attempts to address this unsatisfactory state of affairs:

$$\mathrm{RAG}(k,u) = \frac{\sum_{v\in\hat{S}^k} S^k(v)}{\sum_{v\in S^k} S^k(v)} \in [0,1].$$

Note that $S^k(v)$, and not $\hat{S}^k(v)$, is used in both numerator and denominator.

Whereas precision is regarded as too severe, RAG is usually regarded as a little too lenient, because it completely ignores the identity of the top-k instances.

2.2.4. Mean reciprocal rank. For some applications, including some uses of Web search engines, it may be adequate to inspect, say, only one positive instance, because the information in positive documents is redundant. Mean reciprocal rank (MRR) [Voorhees 99] was designed to evaluate ranking algorithms for such settings. Usually, the system is evaluated over a query set Q. For a specific query $q \in Q$, suppose the rank of the first positive instance is FirstPositiveRank(q). Then MRR is defined as

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{\mathrm{FirstPositiveRank}(q)}.$$

2.2.5. Normalized discounted cumulative gain (NDCG). In yet other settings, a user would be happy with positive documents high in rank, but, unlike MRR, additional subsequent positive documents are not entirely valueless. Normalized discounted cumulative grain (NDCG) [Järvelin and Kekäläinen 00] captures this, as did average precision. NDCG is defined wrt a *rating function* that maps a rank position to a reward score. For a specific query q, let

$$N_q \sum_{i=1}^{k} \frac{2^{\mathrm{rating}(i)}-1}{\log(1+i)}.$$

Here N_q is a normalization factor so that a perfect ordering gets an NDCG score of 1 for each query, k is the number of top responses considered, and rating(i) is the evaluator rating for the item returned at position i. Observe that the numerator is rewarded if rating($\cdot$) is large, and the denominator steadily discounts matches at higher (poorer) ranks. For multiple queries, the N_q scores are averaged.

3. Ranking Feature Vectors

Here we assume that each instance x_i is represented by a feature vector in $\mathbb{R}^d$ and denote the feature vector by x_i itself. We wish to learn a model parameter

vector $\beta \in \mathbb{R}^d$ from the training instances and preference data. We will sometimes abbreviate "model parameter vector" to "model vector." The *score* of an instance x_i under model β is $\beta^\top x_i$, also written as $\beta \cdot x_i$.

In the case of ordinal regression and its special subcase, bipartite ranking, each training instance x_i will be accompanied by a label y_i, similar to classification and regression tasks. In the case of general pair preferences, we get the x_i vectors and a set of preference pairs of the form $i \prec j$, meaning that the score of x_i should be less than the score of x_j, i.e.,

$$\beta^\top x_i \leq \beta^\top x_j.$$

3.1. Max-Margin Formulation for Preference Pairs

For a given training set, there may be no feasible β satisfying all preferences $\prec$. Consider a $\prec$ put together on the basis of millions of search engine users. The directed graph induced by $\prec$ on the items may even be cyclic. To handle such cases, for the constraint $i \prec j$, introduce the slack variable $s_{ij} \geq 0$, and ask that

$$\beta^\top x_i \leq \beta^\top x_j + s_{ij}.$$

We charge a penalty for using $s_{ij} > 0$ by defining the following optimization:

$$\min_{s_{ij} \geq 0; \beta} \frac{1}{|\prec|} \sum_{i \prec j} s_{ij} \quad \text{subject to}$$
$$\beta^\top x_i \leq \beta^\top x_j + s_{ij} \quad \text{for all } i \prec j.$$

In classification, it is common to try to ensure a large separation of positive and negative instance vectors on the direction β. It can be shown that this assists generalization to unseen data [Vapnik et al. 96]. Following the same principle, we would like to strengthen the requirement $\beta^\top x_i \leq \beta^\top x_j + s_{ij}$ to the following "confident separation" requirement:

$$\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij}.$$

Again, as with support vector machines (SVMs), if we can find a β that satisfies $\beta^\top x_i + \epsilon \leq \beta^\top x_j$ for some $\epsilon > 0$, however small, we can scale up β to achieve the fixed margin of 1. To avoid this, the objective must penalize not only $s_{ij} > 0$ but also some norm of β. Many optimizers can conveniently handle the L_2 norm, so a reasonable modified optimization is

$$\min_{s_{ij} \geq 0; \beta} \frac{1}{2}\beta^\top \beta + \frac{B}{|\prec|} \sum_{i \prec j} s_{ij} \quad \text{subject to} \tag{3.1}$$
$$\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij} \quad \text{for all } i \prec j.$$

Here B is a magic parameter that balances pair preference violations against model strength.

The structure of the optimization above is very similar to that of a standard SVM, and quadratic program (QP) solvers used to run SVM optimizations can be used readily.

In the discussion above, we modeled the score of the item x as $\beta^\top x_i$. If for every pair of instances $i \prec j$ we create an instance z_{ij} in a new optimization, we are looking for a β such that $\beta^\top z_{ij} > 0$ for all z_{ij}. If a solution β exists, then there is a hyperplane separating the origin from all z_{ij}s. In applications, the separator may not always be so simple. We can first map x to a vector $\psi(x)$ using a suitable, possibly nonlinear transformation ψ, however. In this case, the score of x will be $\beta^\top \psi(x)$. The hope is that, even if no hyperplane can separate z_{ij}s from the origin in the original space, in the space where ψ maps the points x, the zs will have a linear separator from the origin. The application of transformation ψ here is completely analogous to classification problems. We refer the reader to Schölkopf and Smola [Schölkopf and Smola 02] and Shawe-Taylor and Christianini [Shawe-Taylor and Cristianini 04] for details.

The formulation in (3.1) has some performance issues:

- Common SVM implementations will take time almost quadratic in the number of training pairs. Here a training instance is actually a *pair* of data instances. Therefore rank-learning problems may have even larger training sets.

- Consider relevance-judgment input of the form used in standard IR, over a corpus of 10^6 documents. For each query, we are given, say, 10 relevant and (implicitly) $10^6 - 10$ irrelevant documents. This will lead to about 10^7 preference pairs $x_i \prec x_j$.

- Many of these preference pairs are unnecessary for learning the best β. If $\beta^\top x_0 \leq \beta^\top x_1$ and $\beta^\top x_0 \leq \beta^\top x_2$, for example, then $\beta^\top x_0 \leq \lambda\beta^\top x_1 + (1 - \lambda)\beta^\top x_2$ for $\lambda \in [0, 1]$.

- Unfortunately, we cannot, in general, say ahead of time which preferences will be redundant.

We can ease the running time a bit by replacing the max-margin optimization with a simpler, unconstrained approximation where gradient descent can be used. There is no strict mathematical correspondence between (3.1) and the approximation, however. In Section 3.2, we describe a recent breakthrough that achieves, in linear time, a approximation with a formal guarantee.

Together, $\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij}$ and $s_{ij} \geq 0$ mean $s_{ij} = \max\{0, \beta^\top x_i - \beta^\top x_j + 1\}$ ("hinge loss"). Optimization (3.1) can be rewritten without using s_{ij} as follows:

$$\min_\beta \frac{1}{2}\beta^\top\beta + \frac{B}{|\prec|}\sum_{i\prec j}\max\{0, \beta^\top x_i - \beta^\top x_j + 1\}.$$

Now, $\max\{0, t\}$ can be approximated by a number of smooth functions. A particularly convenient one is $\log(1 + e^t)$. It asymptotes to $y = 0$ as $t \to -\infty$ and to $y = t$ as $t \to \infty$. This gives us a simple unconstrained optimization, which can be solved by Newton's method:

$$\min_{\beta\in\mathbb{R}^d} \frac{1}{2}\beta^\top\beta + \frac{B}{|\prec|}\sum_{i\prec j}\log(1 + \exp(\beta^\top x_i - \beta^\top x_j + 1)).$$

Observe that if $\beta^\top x_i - \beta^\top x_j + 1 \ll 0$, i.e., if $\beta^\top x_i \ll \beta^\top x_j$, then we pay a small penalty, whereas if $\beta^\top x_i - \beta^\top x_j + 1 \gg 0$, i.e., if $\beta^\top x_i \gg \beta^\top x_j$, then we pay a large penalty.

3.2. Near-Linear Time Max-Margin Approximation

The primal optimization (3.1) can be reformulated as

$$\min_{\beta, s\geq 0} \frac{1}{2}\beta^\top\beta + B\,s, \quad \text{such that} \tag{3.2}$$
$$\forall \vec{c} \in \{0,1\}^{|\prec|} : \frac{1}{|\prec|}\beta^\top\sum_{i\prec j} c_{ij}(x_j - x_i) \geq \frac{1}{|\prec|}\sum_{i\prec j} c_{ij} - s.$$

Note that, instead of $|\prec|$ slack variables, there is now only one slack variable s, but now we have $2^{|\prec|}$ primal constraints (one constraint for each value of $\vec{c}$) and $2^{|\prec|}$ corresponding dual variables. This may appear to be a step backward, but, if most primal constraints are redundant, most dual variables will be inactive, i.e., 0. We will exploit this property.

First, we will establish that any solution to (3.2) corresponds to a solution to (3.1), and vice versa. Fix a β_0 in (3.1). For optimality, we must then pick $s^*_{ij} = \max\{0, 1 + \beta_0^\top x_i - \beta_0^\top x_j\}$. Now fix the same β_0 in (3.2). For optimality, we must pick

$$s^* = \min_{\vec{c}\in\{0,1\}^{|\prec|}} \left\{ \frac{1}{|\prec|}\sum_{i\prec j} c_{ij}\big(1 + \beta_0^\top x_i - \beta_0^\top x_j\big) \right\}.$$

Observe that $\vec{c}$ can be picked element-wise:

$$c^*_{ij} = [\![1 + \beta_0^\top x_i - \beta_0^\top x_j \leq 0]\!].$$

With this choice, we can verify that objectives of (3.1) and (3.2) will be equal using $\beta_0, \{s^*_{ij}\}, s^*, \{c^*_{ij}\}$.

We set up the approximate optimization as follows:

- Instead of all $\{0,1\}^{|\asymp|}$, start with $\mathcal{W} \subset \{0,1\}^{|\asymp|}$, typically $\mathcal{W} = \varnothing$;
- Solve (3.2) with $\mathcal{W}$ instead of $\{0,1\}^{|\asymp|}$ to get the current β_0, s^*;
- Look for a *violator* c^* such that

$$\frac{1}{|\prec|}\beta_0^\top \sum_{i \prec j} c^*_{ij}(x_j - x_i) \quad < \quad \frac{1}{|\prec|}\sum_{i \prec j} c^*_{ij} \quad - \quad s^* \underbrace{-\epsilon}_{\text{tolerance}} ;$$

- If no such c^* can be found, exit with an objective that is at most the optimal objective plus ϵ;
- Otherwise, add c^* to $\mathcal{W}$ and repeat.

The capability of inducting nonzero dual variables "on demand" and thereby getting a faster algorithm comes from a general scheme for extending SVMs to very large $\mathcal{Y}$ spaces, by Tsochantaridis et al. [Tsochantaridis et al. 05]. Specifically, we can make the following claims:

- For fixed (constant) ϵ, B, and $\max_i \|x_i\|_2$, the number of inclusions into $\mathcal{W}$ before no further c^* is found is *constant*.

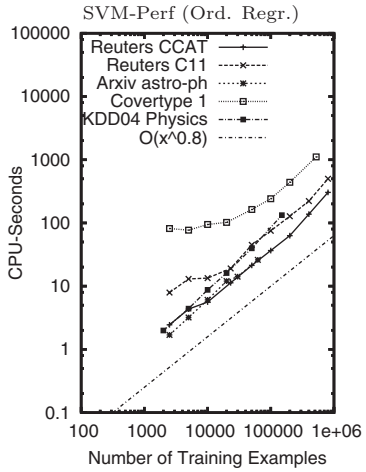


Figure 2. Scaling behavior of approximate linear-time RankSVM invented by Joachims [Joachims 06].

- Each loop above can be implemented in $O(n \log n)$ vector operations in $\mathbb{R}^d$, where all $x_i \in \mathbb{R}^d$. In text applications the vectors are generally very sparse, typically with a small number of nonzero elements that is almost independent of corpus size or d. In such cases, we can assume even that each vector operation takes constant time.

Figure 2 shows that the approximate algorithm indeed scales almost linearly in practice as well, which is a dramatic improvement over known implementations of (3.1), which scale roughly as $n^{3.4}$ (not shown). As of now, extensions to nonlinear kernels are not known.

3.3. Max-Margin Formulation for Ordinal Regression

In the case of ordinal regression, we generally do not wish to reduce the problem to preference pair learning, because there can be too many pairs. Suppose we are scoring instances on a r-point scale. Apart from β, we will optimize over $r-1$ thresholds:

$$-\infty = b_0 \leq b_1 \leq b_2 \leq \cdots \leq b_{r-2} \leq b_{r-1} \leq b_r = +\infty.$$

Let $j \in \{1, \ldots, r\}$ index score levels, and the ith instance in the jth level be denoted x_i^j. We wish to pick β such that, for any x_i^j,

$$b_{j-1} \ < \ \beta^\top x_i^j \ < \ b_j.$$

Extending the max-margin principle to these two inequalities, we will insist that

$$b_{j-1} + 1 \ < \ \beta^\top x_i^j \ < \ b_j - 1,$$

and introduce lower slacks $\underline{s}_i^j \geq 0$ and upper slacks $\overline{s}_i^j \geq 0$, and we relax the above inequalities to

$$b_{j-1} + 1 - \underline{s}_i^j \ \leq \ \beta^\top x_i^j \ \leq \ b_j - 1 + \overline{s}_i^j.$$

For an illustration of the boundaries and slacks, see Figure 3.

The objective to minimize is modified to

$$\min_{\beta, b, \underline{s} \geq \vec{0}, \overline{s} \geq \vec{0}} \tfrac{1}{2}\beta^\top \beta + B \sum_{j,i} (\underline{s}_i^j + \overline{s}_i^j),$$

where B is, as before, a parameter to balance slacks against model complexity $\|\beta\|_2$. The resulting quadratic program with linear inequality constraints can be solved [Chu and Keerthi 05] by some of the same optimization packages as are used to optimize standard SVMs.

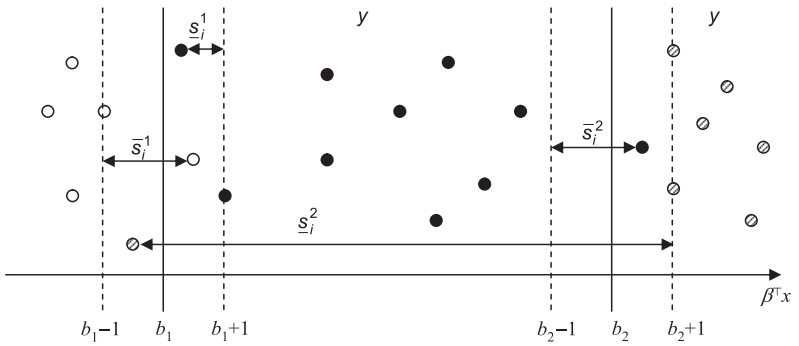


Figure 3. Illustration of class boundaries and slacks in ordinal regression [Chu and Keerthi 05].

3.4. Ranking Loss and Gradient-Based Approaches

In some applications, the preference $i \prec j$ may be accompanied by a confidence score. For example, in multi-user Web searches, there may be no consensus on the relative merits of instances i and j, and a compromise may be desired. We will model this by adding a *target probability* $\bar{p}_{ij}$ with which the trained system should rank i worse than j.

To appreciate this training model better, consider a standard logistic regression in machine learning, where $x \in \mathbb{R}^d$ is regressed to two classes $\mathcal{Y} = \{-1, +1\}$. The assumption is that the regressor first computes a probability

$$\Pr(Y = +1|x) = 1 - \Pr(Y = -1|x) = \frac{\exp(\beta^\top x)}{1 + \exp(\beta^\top x)},$$

and then tosses a coin with these probabilities to decide a class label. Training data $\{(x_i, y_i)\}$ usually has binary class labels, and the training optimization seeks to find

$$\max_\beta \sum_i \log \Pr(Y = y_i|x_i).$$

There is no particular reason, however, why training data has to be certain about the y_i values. In particular, the trainer may well provide empirical class probabilities in the form $(x_i, \bar{p}_i)$ where $\bar{p}_i$ is some empirical confidence of the trainer that $Y_i = 1$. We can now seek to minimize some form of disagreement between $p_i = \Pr(Y = +1|x_i)$ and $\bar{p}_i$. In the ranking problem, $\bar{p}_{ij}$ represents the trainer's confidence that $i \prec j$.

Returning to the ranking problem, suppose the system implements ranking by first evaluating a scoring function f on instances. The score of x_i is $f(x_i) \in \mathbb{R}$.

The *modeled* posterior probability p_{ij} of ranking i worse than j is assumed to have a familiar log-linear form as in logistic regression:

$$p_{ij} = \frac{\exp(f(x_j) - f(x_i))}{1 + \exp(f(x_j) - f(x_i))}.$$

If $f(x_j) \gg f(x_i)$, $p_{ij} \to 1$; if $f(x_j) \ll f(x_i)$, $p_{ij} \to 0$.

Because preference pairs can share instances in arbitrary ways, the trainer cannot assign $\bar{p}_{ij}$ arbitrarily; there are some *consistency requirements* on $\bar{p}_{ij}$s. Let us assume that $\bar{p}_{ij}$ must be *consistent* with some ideal node-scoring function $\bar{f}$, such that

$$\bar{p}_{ij} = \frac{\exp(\bar{f}(x_j) - \bar{f}(x_i))}{1 + \exp(\bar{f}(x_j) - \bar{f}(x_i))}.$$

Using the above, Burges et al. [Burges et al. 05] showed that

$$\bar{p}_{ik} = \frac{\bar{p}_{ij}\bar{p}_{jk}}{1 + 2\bar{p}_{ij}\bar{p}_{jk} - \bar{p}_{ij} - \bar{p}_{jk}}.$$

The above expression has some nice properties. Consider $\bar{p}_{ik}$ when $\bar{p}_{ij} = \bar{p}_{kj}$. In particular, when $\bar{p}_{ij} = \bar{p}_{jk} = 0$, $\bar{p}_{ik} = 0$ as well. That is, if we know with certainty that $i \succ j$ and $j \succ k$, we know with certainty that $i \succ k$. Likewise, $\bar{p}_{ij} = \bar{p}_{jk} = 1$, which is equivalent to certain knowledge that $i \prec j$ and $j \prec k$, implies that $\bar{p}_{ik} = 1$, or $i \prec k$. More interestingly, when $\bar{p}_{ij} = \bar{p}_{kj} = 1/2$, $\bar{p}_{ik} = 1/2$. Thus, perfect uncertainty and perfect certainty propagate.

As before, we can model $f(x_i) = \beta^\top x_i$. Finding f amounts to fitting β. We want to fit β so as to minimize disagreement between trainer-specified $\bar{p}_{ij}$ and modeled p_{ij}. Burges et al. propose to minimize the KL divergence:

$$\begin{aligned}\mathrm{KL}(\bar{p}_{ij} \| p_{ij}) &= \bar{p}_{ij} \log \frac{\bar{p}_{ij}}{p_{ij}} + (1 - \bar{p}_{ij}) \log \frac{1 - \bar{p}_{ij}}{1 - p_{ij}} \\ &= -\bar{p}_{ij} \log p_{ij} - (1 - \bar{p}_{ij}) \log(1 - p_{ij}) + \text{constant independent of } p_{ij}.\end{aligned}$$

In the above,

$$p_{ij} = \frac{\exp(\beta^\top x_i - \beta^\top x_j)}{1 + \exp(\beta^\top x_i - \beta^\top x_j)}.$$

We thus get a final optimization problem over β. Burges et al. used a neural network to optimize β.

4. Ranking Nodes in Graphs

Thus far, training and test instances were assumed to be feature vectors in $\mathbb{R}^d$, drawn from some unknown but fixed distribution. In this part of the survey, we consider situations where instances are not (only) feature vectors but nodes in a graph $G = (V, E)$. As discussed in Section 1, many forms of social network data are being represented as graphs, with strongly motivated ranking applications that use such graph representations.

What kind of signal might be available from G? If the edges of G represent similarity, we may want neighboring nodes to have similar scores, and therefore, ranks. The BLAST data set[3] lists dissimilarity scores between pairs of proteins. In a corpus of documents, similarity between a pair of documents, represented as a feature vector of word counts, may dictate that they be ranked similarly wrt a query. If each node u is assigned a score $f(u)$, the undirected edge $(u, v) \in E$ expresses the prior belief that scores of its endpoints should not vary much, i.e., f should be *smooth* wrt G. We explore undirected graph Laplacians in Section 4.1, which let us enforce such smoothness constraints easily.

The second interpretation, leading to by far the most popular algorithms for ranking nodes in social networks [Page et al. 98, Kleinberg 99], is that each directed edge $(u, v) \in E$ represents an *endorsement* of v by u. Social network analysis has led to "random surfer models" where the steady state of a Markovian walk on a directed graph leads to scores for the nodes. These random walks are related to a directed version of the graph Laplacian. We explore these in Sections 4.2.1, 4.2.2, and 4.2.3.

4.1. Undirected Graph Laplacian Smoother

Let us represent the undirected graph G by a symmetric weighted node adjacency matrix: $A(u, v) = A(v, u)$ is the weight of edge (u, v), which is 0 if the edge does not exist. Suppose f is the vector of scores assigned to nodes, with node u having score $f(u)$. An intuitive way to encourage smoothness is to minimize

$$\sum_{(u,v)\in E} A(u, v)\big(f(u) - f(v)\big)^2.$$

This objective can be expressed in a simple matrix notation. Consider the simple (having no self-loops or parallel edges) undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$. The node-edge incidence matrix $N \in \{-1, 0, 1\}^{n\times m}$ is

[3] http://www.kyb.tuebingen.mpg.de/bs/people/weston/rankprot/supplement.html

defined as

$$N(v,e) = \begin{cases} -\sqrt{A(e)} & \text{if } e = (v,\cdot), \\ \sqrt{A(e)} & \text{if } e = (\cdot,v), \\ 0 & \text{if } v \text{ is not either endpoint of } e. \end{cases}$$

Here the endpoints of an undirected edge are ordered arbitrarily. Let D be a diagonal matrix with $D(u,u)$ equal to the total weight of edges incident with u:

$$D(u,u) = \sum_v A(u,v).$$

The Laplacian matrix for G is defined as

$$L_G(u,v) = \begin{cases} \sum_w A(u,w), & u = v \\ -A(u,v), & u \neq v, (u,v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Then it is well-known [Dhillon 01, Agarwal 06] that

$$L_G = NN^\top = D - A,$$

and, therefore, $D - A$ is a symmetric positive semidefinite matrix. Furthermore, for any vector $f \in \mathbb{R}^n$,

$$f^\top Lf = \sum_{(u,v)\in E} A(u,v)\big(f(u) - f(v)\big)^2.$$

Therefore, subjecting the score vector f to the operation $f^\top Lf$ penalizes node scores that are very different across "heavy" edges.

Returning to learning from preference pairs, if $u \prec v$ in the training data, we want $f(u)+1 \leq f(v)$ (including the margin and excluding the slack). Define the *ranking loss* of score vector f wrt the preference $u \prec v$ as $\max\{0, 1+f(u)-f(v)\}$. Observe that the sum of ranking losses over all preference pairs, $\sum_{u\prec v} \max\{0, 1+f(u)-f(v)\}$, is an upper bound on the number of violated training preferences.

By combining smoothness with ranking loss, we can now write down the complete optimization problem as

$$\min_{f\in\mathbb{R}^n} f^\top Lf + B\sum_{u\prec v} \max\{0, 1 + f(u) - f(v)\},$$

or, if we incorporate slack variables,

$$\min_{f\in\mathbb{R}^n, \{s_{uv}\geq 0 : \forall u\prec v\}} f^\top Lf + B\sum_{u\prec v} s_{uv} \tag{4.1}$$

subject to $s_{uv} \geq 1 + f(u) - f(v)$, i.e., $f(v) - f(u) \geq 1 - s_{uv} \quad \forall u \prec v$. B balances between roughness and data fit as before. Because L is positive semidefinite, this is a convex quadratic program with linear inequality constraints.

4.2. From Score Smoothness to Endorsement

For some applications, smoothness of scores across undirected edges is not reasonable to expect. Millions of obscure Web pages link to http://kernel.org, but one would not expect their scores to be anywhere near that of http://kernel.org, say, for the query "linux kernel." Obscure pages confer endorsement to popular pages, but are not necessarily as popular.

We will discuss two interrelated approaches to learning in this setting, both based on random walks on directed graphs. One approach is to extend the graph Laplacian to directed graphs; this is discussed in Section 4.2.1. Another approach is to model the random walk as a network flow. The asymmetric endorsement scenario above is naturally modeled as many small flows converging on to a popular node with very large flow-through. This is discussed in Section 4.2.2. A third, slightly different, learning problem involving a limited number of edge types (see Figure 1) is explored in Section 4.2.3.

4.2.1. Directed Laplacian. The Laplacian of a directed graph, proposed by Chung [Chung 05], is defined as follows. Assume each row of A has at least one nonzero element, i.e., there is no dead-end node. Define a diagonal matrix D as before, with $D(u,u)$ being the sum of the uth row of A. Define Markovian transition probability matrix $Q \in [0,1]^{n\times n}$ with $Q(u,v) = \Pr(u \to v) = \Pr(v|u) = A(u,v)/D(u,u)$. Assume that the Markov random walk defined by Q is *irreducible* and *aperiodic*. Let $\pi \in \mathbb{R}^n$ be the steady-state probability vector for the random walk represented by Q, and let $\Pi = \mathrm{diag}(\pi)$. The directed graph Laplacian is defined as

$$L = \mathbb{I} - \frac{\Pi^{1/2} Q \Pi^{-1/2} + \Pi^{-1/2} Q \Pi^{1/2}}{2}.$$

Zhou et al. [Zhou et al. 05] explain why this specific definition is useful for encouraging a smooth score vector. Once the directed Laplacian is defined, it can be used in optimization (4.1) in place of the undirected graph Laplacian. For the directed Laplacian, it can be shown that

$$f^\top L f = \sum_{(u,v)\in E} \pi(u) Q(u,v) \left(\frac{f(u)}{\sqrt{\pi(u)}} - \frac{f(v)}{\sqrt{\pi(v)}} \right)^2.$$

Therefore, if there are no training preferences, $f^\top L f$ will be minimized for $x(u) \propto \sqrt{\pi(u)}$. In other words, in the absence of training preference pairs, the algorithm will choose a rank order on the nodes that is identical to the rank order obtained using the Markov walk defined by Q. This shows that Laplacian smoothing has a reasonable default or "parsimonious" belief. Agarwal [Agarwal 06] gives more

compelling reasons for using a Laplacian smoother: Reducing $f^\top Lf$ is equivalent to regularizing f in a reproducing kernel Hilbert space (RKHS).

Despite these elegant foundations, the Laplacian smoothing perspective has two limitations. First, the optimization (4.1) involves diagonalizing and computing the pseudoinverse L^+ of a large $n \times n$ matrix L, which is time-consuming and perhaps even impractical for Web-scale work. Second, the generalization power (see Section 5) of the learning algorithm is expressed in terms of the inaccessible and somewhat inscrutable quantity $\max_u L^+(u,u)$.

4.2.2. Constrained network flows. In this section, we propose an alternate view of the random walk as a network flow, and use this view to design a score vector for nodes in V. We first study basic properties satisfied by many Markovian walks (such as the one yielding PageRank) and then look for other solutions that satisfy these properties while also satisfying preference pairs between nodes.

Earlier we assumed that the Markov random walk defined by Q is *irreducible* and *aperiodic*. In PageRank, this is achieved using the device of the *teleport*:

- When at any node u, the random surfer tosses a coin with head probability α.
- If the coin comes up heads, the surfer uses the Markov transition matrix to walk to an out-neighbor v of u.
- With the remaining probability $1-\alpha$, the surfer *teleports* to a random node w, chosen using a multinomial distribution $r = (r_1, \ldots, r_w, \ldots, r_n)$. In this paper, we usually consider the uniform teleport $r = \vec{1}/|V|$.

We can express teleport without dense all-to-all transitions by defining transitions between each node and a new dummy node d, in both directions. The augmented graph is $\hat{G} = (\hat{V}, \hat{E})$. Markov transition probabilities in the augmented graph are expressed by the $(n+1) \times (n+1)$ matrix $\hat{Q}$ given by

$$\hat{Q} = \begin{bmatrix} \alpha Q & (1-\alpha)\vec{1}_{|V|\times 1} \\ \vec{1}/|V| & 0 \end{bmatrix}.$$

The earlier steady-state equation $\pi = Q^\top \pi$ is replaced with $\hat{\pi} = \hat{Q}^\top \hat{\pi}$, but π and $\hat{\pi}$ are closely related [Langville and Meyer 04] and one can be obtained from the other. Henceforth, for simplicity of notation, we will continue using Q and π in place of $\hat{Q}$ and $\hat{\pi}$ even in the augmented graph.

Q and π can be used to define a *reference circulation* q_{uv} along each edge in $\hat{E}$, given by $q_{uv} = \pi(u)Q(u,v)$. Note that, by design, $\sum_{u,v} q_{uv} = 1$, so $\{q_{uv}\}$ is a multinomial distribution.

To rank nodes in the presence of pair preferences, we estimate another circulation in $\hat{G}$, given by p_{uv} along edge (u, v), with

$$\forall u, v \in \hat{V}: \quad p_{uv} \geq 0, \tag{4.2}$$

$$\sum_{u,v \in \hat{V}} p_{uv} = 1. \tag{4.3}$$

The solution p will naturally induce a score ϕ on the nodes:

$$\phi(v) = \sum_{(u,v) \in \hat{E}} p_{uv},$$

and ϕ will be used to rank the nodes.

Apart from being a multinomial distribution, to be a legitimate circulation, p must satisfy the following property:

$$\forall v \in \hat{V}: \quad \sum_{(u,v) \in \hat{E}} p_{uv} = \sum_{(v,w) \in \hat{E}} p_{vw}. \tag{4.4}$$

We also need to make p_{uv} a legitimate probability $\Pr(u) \Pr(u \to v)$, while taking into account the teleport that happens with probability $1 - \alpha$. This is easily done using more equality constraints:

$$\forall v \in V: \quad \frac{p_{vd}}{1 - \alpha} = \frac{\sum_{w \in V} p_{vw}}{\alpha}. \tag{4.5}$$

If we asserted only these constraints on p, there could be an infinite number of solutions; q itself is a solution. Tomlin [Tomlin 03] proposed finding a p to maximize its entropy $H(p)$:

$$\max_{\{p_{uv}: u,v \in \hat{V}\}} -p_{uv} \log p_{uv}$$

subject to (4.2), (4.3), (4.4), and (4.5).

Offhand, it is not clear why $H(p)$ is the right objective. Maximizing $H(p)$ is equivalent to minimizing the KL divergence [Cover and Thomas 91] from p to the uniform distribution over edges $(1/|\hat{E}|, \ldots, 1/|\hat{E}|)$, and, in general, the uniform flow distribution over edges does not even satisfy (4.4) and (4.5).

By replacing $H(p)$ with the KL divergence [Cover and Thomas 91] from p to q, we achieve three goals:

- Establish a correspondence with the directed Laplacian regularization;
- Give stability and generalization bounds similar to the Laplacian approach (Section 5);

- Solve a numerically more benign optimization that outputs valid PageRank scores.

Changing our objective as motivated above, and taking into account preference pairs, we write the final optimization as follows:

$$\begin{aligned} &\min_{\substack{\{0\le p_{uv}\}\\ \{0\le s_{uv}\}}} \sum_{(u,v)\in\hat{E}} p_{uv}\log\frac{p_{uv}}{q_{uv}} + B\sum_{u\prec v} s_{uv} \\ &\text{subject to} \quad (4.2),\ (4.3),\ (4.4),\ (4.5), \\ \text{and} \quad \forall u \prec v: \quad &\sum_{(w,u)\in\hat{E}} p_{wu} - \sum_{(w,v)\in\hat{E}} p_{wv} - s_{uv} \le 0. \end{aligned} \tag{4.6}$$

Note that, while asserting preference constraints, we do not use a margin; we will revisit this issue soon. Clearly, if $\prec = \varnothing$, p will solve to q, and therefore the "parsimonious belief" underlying optimization (4.6) is the same as (4.1).

By design, even when $\prec \neq \varnothing$, optimizations (4.1) and (4.6) have a broad correspondence. Let p and q be valid flow distributions on $\hat{G}$. Define a score f_p for each node using p as follows:

$$f_p(v) = \sqrt{\sum_{\{u:(u,v)\in\hat{E}\}} p_{uv}}.$$

Then we can show [Agarwal and Chakrabarti 07] that

$$\mathrm{KL}(p\|q) \le \epsilon \quad \Rightarrow \quad f_p^\top L f_p \le 4\epsilon_1^2,$$

i.e., encouraging p to stay close to q also ensures that f_p as defined above is smooth wrt the graph Laplacian.

This implication is rather useful, because the optimization (4.6) is fairly benign and efficient. No diagonalization of L or a quadratic program (QP) optimizer is needed; a good Newton method such as L-BFGS [Liu and Nocedal 89] or BLMVM [Benson and Moré 01] suffices [Agarwal et al. 06].

Introducing a margin is not trivial. We cannot just modify the preference inequalities to

$$\forall u \prec v: \quad \underbrace{1}_{\text{margin}} + \sum_{(w,u)\in\hat{E}} p_{wu} \le s_{uv} + \sum_{(w,v)\in\hat{E}} p_{wv} \tag{4.7}$$

because, unlike ranking feature vectors according to $\beta^\top x_i$, where scaling up β also increased the magnitude of $\beta^\top x_j - \beta^\top x_i$, here the total inflow into any node is at most 1, and, in typical social networks, very small. This also precludes using any arbitrary lesser constant in place of 1.

The solution is to relax the equality constraint (4.3) and let p be some positive multiple of a flow. This does not hurt the KL objective, because, if q is a probability distribution and p an unnormalized distribution, so that $\sum_x p(x) = F$, then

1. $\mathrm{KL}(p\|q) \geq 0$ if $F \geq 1$.
2. For a fixed $F \geq 1$, $\arg\min_p \mathrm{KL}(p\|q) = F\,q$.

Given this fact, we can change objective (4.6) to

$$\min_{\substack{\{p_{uv}\},\{s_{uv}\} \\ F \geq 1}} \sum_{(u,v)\in E'} p_{uv} \log \frac{p_{uv}}{q_{uv}} + C \sum_{u \prec v} s_{uv} + C_1\, F^2,$$

and change (4.3) to $\sum_{(u,v)\in E'} p_{uv} = F$, and then enforce (4.7).

A "clean-room" method to evaluate the algorithm is as follows:

1. Obtain a real-life graph G, or use a realistic synthetic graph generator like RMAT [Chakrabarti et al. 04].
2. Compute the baseline random walk Q and flow q, with uniform teleport.
3. Select a small subgraph (perhaps a single node) that will be kept a secret from the learning algorithm.
4. Increase the total teleport into the secret subgraph to `HiddenTeleport` $\in (0, 1)$. Other nodes divide $1 -$ `HiddenTeleport` equally among themselves.
5. Compute PageRank flows p with this hidden teleport. These perturbed PageRank scores will be concealed from the learning algorithm as well.
6. The perturbed PageRank vector is compared with the standard PageRank vector, and concordant and discordant pairs sampled and presented to the algorithm as training data.
7. Another disjoint sample of pairs is then used to evaluate the algorithm, in terms of `TestError`, the fraction of violated test pairs.

Figure 4 shows the error rates of (4.1), (4.6), and (4.6) with (4.7). The x-axis is the amount of teleport probability routed into the secret community.

Before leaving the topic we should emphasize that the algorithms represented by (4.1) or (4.6) are *local* learning algorithms, in two related senses. When the learning algorithm adjusts the flow p_{uv} away from q_{uv}, the effect is limited to a neighborhood around the edge. Nodes that are "far" from u and v are not affected. This means that if all nodes involved in test pairs are very far

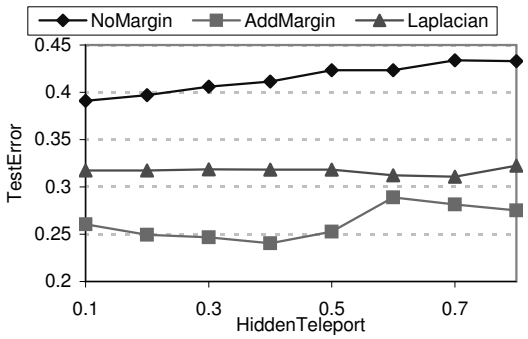


Figure 4. Comparison of accuracies of algorithms as teleport into the hidden favored community is changed [Agarwal and Chakrabarti 07]. (4.6) performs worse than (4.1), but (4.6) with (4.7) performs better than the others.

from any node involved in training pairs, the algorithm will generalize poorly. Generalization will be visible only if nodes involved in test cases are sufficiently close to nodes involved in training. Another way to look at the situation is that the model has high complexity: The number of parameters estimated by the learning algorithm scales with the size of the graph, making it easy to fit training data without guaranteeing good generalization in test data.

4.2.3. Typed edge conductance. In this section, we discuss another learning model parameterized by an unknown edge conductance matrix C that must be estimated. While earlier we were directly estimating $p_{uv} = p(u)C(u \to v)$, now we characterize elements of C using *edge types*. Specifically, each directed edge (u, v) has a *type* $t(u, v) \in \{1, \ldots, T\}$, where T is some small number unrelated to graph size. Types of all edges are fixed and known beforehand. The model parameters are $\beta = (\beta(1), \ldots, \beta(T))$. The *weight* of edge (u, v) is $\beta(t(u, v))$. Again, we assume there are no dead-end nodes, and we write the *conductance* of (u, v) as

$$C(i \to j) = C(j, i) = \begin{cases} \alpha \frac{\beta(t(i,j))}{\sum_{(i,k)\in E} \beta(t(i,k))}, & i \neq d, j \neq d, \\ 1 - \alpha, & i \neq d, j = d, \\ r_j, & i = d, j \neq d, \\ 0, & i = j = d. \end{cases}$$

Here, r is a fixed teleport vector and d the dummy node as before. Note that we have written C in a transposed form to ease notation. C is a function of β, and the PageRank vector satisfies $p = C(\beta)p$. The true ranking loss incurred by p is $\sum_{u \prec v} [\![p(u) > p(v)]\!]$, which we will approximate with some smooth version $\sum_{u \prec v} \text{loss}(p(u) - p(v))$.

Standard PageRank emerges as the solution if $\prec = \varnothing$ and all $\beta(t)$ are equal. This leads us to design a natural model regularization term:

$$\text{ModelCost}(\beta) = \sum_{t,t'} \big(\beta(t) - \beta(t')\big)^2.$$

Also note that $C(\beta)$ is unchanged upon scaling β, so we can insist that all $\beta(t) \geq 1$. Putting the above pieces together, the optimization problem becomes

$$\min_{\beta \geq \vec{1}} \text{ModelCost}(\beta) + B \sum_{u \prec v} \text{loss}(p(u) - p(v)),$$

where $p = C(\beta)p$. If we made both C and p variables in the optimization, we would get a quadratically constrained quadratic program (QCQP) [Boyd and Vandenberghe 04], which is more complicated than the simple QPs with linear constraints or the gradient-based methods we have been using. To avoid QCQPs, we approximate $p \approx C^H p^0$, where p^0 is an initial vector (typically $\vec{1}/|V|$) and H is a *horizon* suitably large to ensure convergence of p.

It turns out that our earlier approximation $\text{loss}(y) \stackrel{\text{def}}{=} \max\{0, y\} \approx \log(1 + e^y)$ does not work well here [Chakrabarti and Agarwal 06], and a loss function is needed that is exactly 0 for all nonpositive y. *Huber loss* works well:

$$\text{huber}(y) = \begin{cases} 0, & y \leq 0, \\ y^2/(2W), & y \in (0, W], \\ y - W/2, & W < y, \end{cases}$$

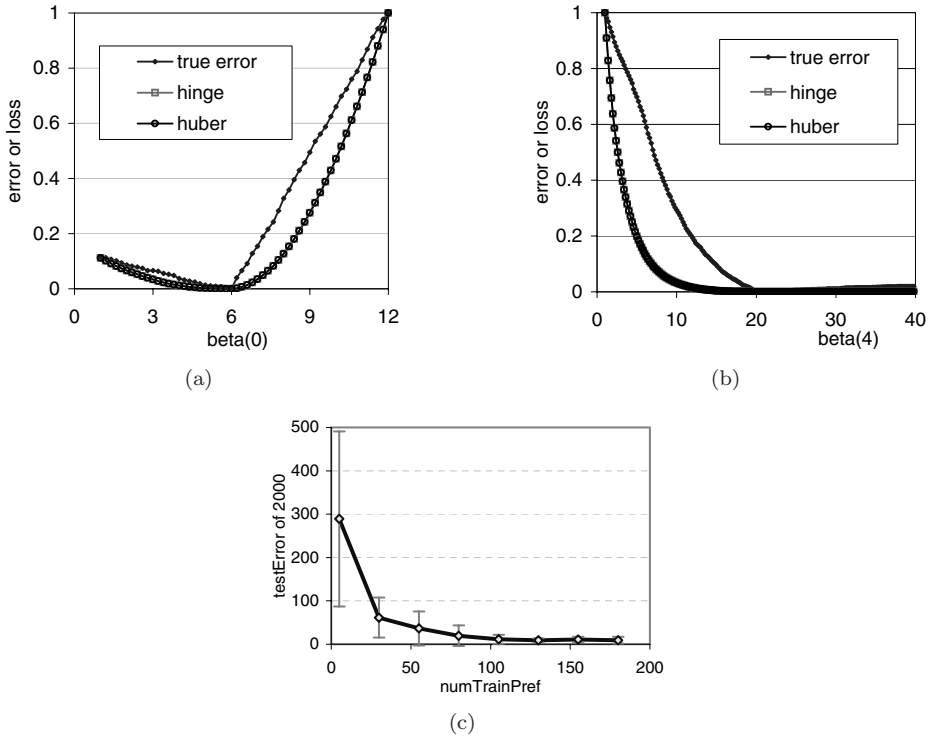
for a suitably small window W. Summarizing, our approximate optimization is

$$\min_{\beta \geq \vec{1}} \sum_{t,t'} \big(\beta(t) - \beta(t')\big)^2 + B \sum_{u \prec v} \text{huber}\left((C^H p^0)_u - (C^H p^0)_v\right).$$

Unfortunately, this is not a convex optimization. A mixture of grid search and gradient descent is reliable in practice, however.

Figure 5 shows the true (0/1) and approximate loss surfaces against two sampled $\beta(t)$s (a, b), and the decrease in test error with increasing training set size (c). The number of model parameters estimated is essentially constant compared to growing with the graph size as in (4.1) or (4.6), and numerical optimizers seem to behave much better despite having to deal with a nonconvex optimization.

Typed E-R graphs are sufficiently expressive to integrate node features into link-based ranking [Balmin et al. 04]. Consider textual annotations associated with nodes, which are very common in E-R graph applications. We first tokenize node text into words and collect the vocabulary of words over the whole graph.



(a) (b)

(c)

Figure 5. (a, b) The approximate objective tracks the true loss reasonably, reaching similar minima. The x-axis is the value of $\beta(t)$; the y-axis shows various losses. (The curves for "hinge" and "huber" overlap almost everywhere.) (c) Learning rate is high; few preference pairs suffice to learn β well. The x-axis is the number of training pairs; the y-axis is test error. The mean and one standard deviation bars are shown.

We then augment the E-R graph with a new set of word nodes, one for each word in the vocabulary. We connect each word node w to all entity nodes v where the word w occurs, using a new "word $\rightarrow$ entity" edge type. A query is interpreted as a set of keywords, and the teleport vector r is designed so that there is positive teleport only to the query words.

5. Stability and Generalization

Traditional learning theory is concerned with studying the power of a learning algorithm to generalize from finite training data to an expected accuracy over

unknown test data. Training and test data are assumed to be sampled from a population of instances using a common and fixed, but usually unknown, distribution $\mathcal{D}$.

In the case of learning to rank, because a unit of training or testing is a *pair* of instances, there are multiple ways to set the stage for generalization results in terms of how $\prec$ is sampled. In bipartite ranking, it is common [Agarwal et al. 05, Agarwal and Niyogi 05] to assume two sampling distributions $\mathcal{D}_+$ and $\mathcal{D}_-$ from which n_+ positive and n_- negative instances are sampled. This then leads to a preference pair set $\prec$ with $|\prec| = mn$. In a transductive [Zhou et al. 05] setting, there is a fixed set of instances X, and $\prec$ is sampled iid using a distribution $\mathcal{D}$ over $X \times X$. Yet other formulations are possible. The form of the bounds naturally depends on the assumptions on how $\prec$ is sampled.

Equation (2.1) gives the empirical accuracy of f on a finite labeled set S using scoring function $f : \mathcal{X} \to \mathbb{R}$. The "true" *ranking accuracy* of f is

$$A(f) = \mathbb{E}_{X \sim \mathcal{D}_{+1}, X' \sim \mathcal{D}_{-1}} \left([\![f(X) > f(X')]\!] + \frac{1}{2} [\![f(X) = f(X')]\!] \right).$$

In bipartite ranking the training set $S = \{(x_i, y_i \in \{-1, 1\})\}$. Its projections on $\mathcal{X}$ and $\mathcal{Y}$ will be called $S_\mathcal{X}$ and $S_\mathcal{Y}$. We are interested in upper-bounding

$$\Pr_{\mathcal{D}_+, \mathcal{D}_-} (|\hat{A}(f, S) - A(f)| > \epsilon).$$

It is slightly easier [Agarwal et al. 05] to first fix a specific label sequence $\underline{y}$, with n_+ +1s and n_- −1s. Then we can show that

$$\Pr_{S_\mathcal{X} | S_\mathcal{Y} = \underline{y}} \left(\hat{A}(f, T) - A(f) \geq \epsilon \right) \leq 2 \exp \left(- \frac{2\, n_+\, n_-\, \epsilon^2}{n_+ + n_-} \right).$$

Apart from the traditional PAC-style bounds above [Valiant 84], there are recently proved stability-based bounds with respect to specific loss functions [Bosquet and Elisseeff 02]. Roughly speaking, Bousquet et al.'s results allow us to infer that a learning algorithm generalizes if its regularized loss function is stable to leave-one-out perturbations. Given the data graph, the objective in (4.6) can be rewritten in the regularized risk form:

$$R_{\text{reg}}(p) = \frac{1}{m} \sum_{j=1}^{m} \underbrace{\max \left\{ 0, \textstyle\sum_{(w,u) \in \hat{E}} p_{wu} - \sum_{(w,v) \in \hat{E}} p_{wv} \right\}}_{\text{ranking loss}} + \lambda\, \text{KL}(p \| q).$$

Recall that here $\prec$ is sampled randomly from $V \times V$ according to some unknown fixed distribution. It can be shown [Agarwal and Chakrabarti 07] that, over

random draws of $\prec$ with $|\prec| = m$, with probability at least $1 - \delta$,

$$R \leq R_{\text{emp}} + \frac{4 \ln 2}{\lambda m} + \left(\frac{8 \ln 2}{\lambda} + 1 \right) \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

Here R is the true ranking loss and R_{emp} is the empirical ranking loss over training data. Note that this is a bound between R and R_{emp} for the given choice of ranking loss, which is not an upper bound on 0/1 loss. To achieve the latter, an additive margin must be added to the ranking loss [Agarwal and Chakrabarti 07]; we omit the details.

6. Concluding Remarks

In this survey on machine learning methods for ranking, we first discussed ways in which such systems are trained, tested, and evaluated. Then we studied rank learning for instances represented by feature vectors, followed by rank learning for instances represented by nodes in a graph. We finally summarized some results demonstrating the generalizing power of rank-learning algorithms. Some of the formulations, while working well in practice, have no benign (say, convex) optimization. It would be desirable to explore alternate formulations and algorithms that do have guarantees. Learning to rank while being especially sensitive to the quality of responses in the top-ranked positions remains underexplored. Finally, not all formulations discussed here can work with million-node graphs yet, let alone Web-scale graphs.

References

[Agarwal and Chakrabarti 07] A. Agarwal and S. Chakrabarti. "Learning Random Walks to Rank Nodes in Graphs." In *Proceedings of the 24th International Conference on Machine Learning*, pp. 9–16. New York: ACM Press, 2007.

[Agarwal et al. 06] A. Agarwal, S. Chakrabarti, and S. Aggarwal. "Learning to Rank Networked Entities." In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 14–23. New York: ACM Press, 2006.

[Agarwal 06] S. Agarwal. "Ranking on Graph Data." In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 25–32. New York: ACM Press, 2006.

[Agarwal et al. 05] S. Agarwal, T. Graepel, R. Herbrich, S. Har-Peled, and D. Roth. "Generalization Bounds for the Area Under the ROC Curve." *Journal of Machine Learning Research* 6 (2005), 393–425.

[Agarwal and Niyogi 05] S. Agarwal and P. Niyogi. "Stability and Generalization of Bipartite Ranking Algorithms." In *Learning Theory: Proceedings of the 18th Annual Conference on Learning Theory*, pp. 32–47, Lecture Notes in Artificial Intelligence 3559. Berlin-Heidelberg: Springer-Verlag, 2005.

[Agarwal et al. 02] S. Agrawal, S. Chaudhuri, and G. Das. "DBXplorer: A System for Keyword-Based Search Over Relational Databases." In *Proceedings of the 18th International Conference on Data Engineering*, p. 5. Los Alamitos, CA: IEEE Press, 2002.

[Balmin et al. 04] A. Balmin, V. Hristidis, and Y. Papakonstantinou. "Authority-Based Keyword Queries in Databases using ObjectRank." In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pp. 564–575. VLDB Endowment, 2004.

[Benson and Moré 01] S. J. Benson and J. J. Moré. "A Limited Memory Variable Metric Method for Bound Constraint Minimization." Technical Report ANL/MCS-P909-0901, Argonne National Laboratory, 2001.

[Bhalotia et al. 02] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. "Keyword Searching and Browsing in Databases using BANKS." In *Proceedings of the 18th International Conference on Data Engineering*. p. 431. Los Alamitos, CA: IEEE Press, 2002.

[Bosquet and Elisseeff 02] O. Bousquet and A. Elisseeff. "Stability and Generalization." *Journal of Machine Learning Research*, 2 (2002), 499–526.

[Boyd and Vandenberghe 04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[Burges et al. 05] 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*, pp. 89–96. New York: ACM Press, 2005.

[Chakrabarti et al. 04] D. Chakrabarti, Y. Zhan, and C. Faloutsos. "R-MAT: A Recursive Model for Graph Mining." In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pp. 442–445. Philadelphia, PA: SIAM, 2004.

[Chakrabarti and Agarwal 06] S. Chakrabarti and A. Agarwal. "Learning Parameters in Entity Relationship Graphs from Ranking Preferences." In *Knowledge Discovery in Databases: PKDD 2006*, pp. 91–102, Lecture Notes in Computer Science 4213. Berlin-Heidelberg: Springer-Verlag, 2006.

[Cho and Roy 04] J. Cho and S. Roy. "Impact of Search Engines on Page Popularity." In *Proceedings of the 13th International Conference on World Wide Web*, pp. 20–29. New York: ACM Press, 2004.

[Chu and Keerthi 05] W. Chu and S. Keerthi. "New Approaches to Support Vector Ordinal Regression." In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 145–152. New York: ACM Press, 2005.

[Chung 05] F. Chung. "Laplacians and the Cheeger Inequality for Directed Graphs." *Annals of Combinatorics* 9 (2005), 1–19.

[Cover and Thomas 91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. New York: John Wiley and Sons, 1991.

[Dhillon 01] I. Dhillon. "Co-clustering Documents and Words using Bipartite Spectral Graph Partitioning." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 269–274. New York: ACM Press, 2001.

[Diligenti 05] M. Diligenti, M. Gori, and M. Maggini. "Learning Web Page Scores by Error Back-Propagation." In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 684–689. Denver, CO: Professional Book Center, 2005.

[Fogaras et al. 05] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. "Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments." *Internet Mathematics* 2:3 (2005), 333–358.

[Herbrich et al. 99] R. Herbrich, T. Graepel, and K. Obermayer. "Support Vector Learning for Ordinal Regression." In *International Conference on Artificial Neural Networks*, pp. 97–102. New York: Springer, 1999.

[Järvelin and Kekäläinen 00] K. Järvelin and J. Kekäläinen. "IR Evaluation Methods for Retrieving Highly Relevant Documents." In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 41–48. New York: ACM Press, 2000.

[Joachims 02] T. Joachims. "Optimizing Search Engines using Clickthrough Data." In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142. New York: ACM Press, 2002.

[Joachims 06] T. Joachims. "Training Linear SVMs in Linear Time." In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 217–226. New York: ACM Press, 2006.

[Kleinberg 99] J. M. Kleinberg. "Authoritative Sources in a Hyperlinked Environment." *JACM* 46:5 (1999), 604–632.

[Langville and Meyer 04] A. N. Langville and C. D. Meyer. "Deeper Inside PageRank." *Internet Mathematics* 1:3 (2004), 335–380.

[Liu and Nocedal 89] D. C. Liu and J. Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization." *Math. Programming* 45:3 (Ser. B) (1989), 503–528, 1989.

[Nie et al. 05] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. "Object-Level Ranking: Bringing Order to Web Objects." In *Proceedings of the 14th International Conference on World Wide Web*, pp. 567–574. New York: ACM Press, 2005.

[Page et al. 98] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Manuscript, Stanford University, 1998.

[Papakonstantinou et al. 95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. "Object Exchange across Heterogeneous Information Sources." In *Proceedings of the 11th International Conference on Data Engineering*. pp. 251–260. Los Alamitos, CA: IEEE Press, 1995.

[Richardson et al. 06] M. Richardson, A. Prakash, and E. Brill. "Beyond PageRank: Machine Learning for Static Ranking." In *Proceedings of the 15th International Conference on World Wide Web*, pp. 707–715. New York: ACM Press, 2006.

[Salton and Buckley 99] G. Salton and C. Buckley. "Improving Retrieval Performance by Relevance Feedback." *Journal of the American Society for Information Science* 41:4 (1999), 288–297.

[Salton and McGill 83] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

[Schölkopf and Smola 02] B. Schölkopf and A. Smola. *Learning with Kernels*. Cambridge, MA: The MIT Press, 2002.

[Shawe-Taylor and Cristianini 04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge, UK: Cambridge University Press, 2004.

[Tomlin 03] J. A. Tomlin. "A New Paradigm for Ranking Pages on the World Wide Web." In *Proceedings of the 12th International Conference on World Wide Web*, pp. 350–355. New York: ACM Press, 2003.

[Tsochantaridis et al. 05] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. "Large Margin Methods for Structured and Interdependent Output Variables." *JMLR*, 6 (2005), 1453–1484.

[Valiant 84] L. G. Valiant. "A Theory of the Learnable." *Communications of the ACM* 27:11 (1984), 1134–1142.

[Vapnik et al. 96] V. Vapnik, S. Golowich, and A. J. Smola. "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing." In *Advances in Neural Information Processing Systems*, pp. 281–287. Cambridge, MA: The MIT Press, 1996.

[Vassilvitskii and Brill 06] S. Vassilvitskii and E. Brill. "Using Web-Graph Distance for Relevance Feedback in Web Search." In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 147–153. New York: ACM Press, 2006.

[Voorhees 99] E. M. Voorhees. "The TREC-8 Question Answering Track Report." In *Proceedings of the Eighth Text Retrieval Conference (TREC 8)*, pp. 77–82. Gaithersburg, MD: NIST, 1999.

[Yue et al. 07] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. "A Support Vector Method for Optimizing Average Precision." In *roceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 271–278. New York: ACM Press, 2007.

[Zhou et al. 05] D. Zhou, J. Huang, and B. Schölkopf. "Learning from Labeled and Unlabeled Data on a Directed Graph." In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 1041–1048. New York: ACM Press, 2005.

Soumen Chakrabarti, Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Powai, Mumbai 400 076, India (soumen@cse.iitb.ac.in)

Received May 10, 2007; accepted November 19, 2007.