Semi-supervised Learning to Rank with Preference Regularization

Martin Szummer
Microsoft Research
Cambridge, UK
szummer@microsoft.com

Emine Yilmaz
Microsoft
Cambridge, UK
eminey@microsoft.com

ABSTRACT

We propose a semi-supervised learning to rank algorithm. It learns from both labeled data (pairwise preferences or absolute labels) and unlabeled data. The data can consist of multiple groups of items (such as queries), some of which may contain only unlabeled items. We introduce a preference regularizer favoring that similar items are similar in preference to each other. The regularizer captures manifold structure in the data, and we also propose a rank-sensitive version designed for top-heavy retrieval metrics including NDCG and mean average precision.

The regularizer is employed in SSLambdaRank, a semi-supervised version of LambdaRank. This algorithm directly optimizes popular retrieval metrics and improves retrieval accuracy over LambdaRank, a state-of-the-art ranker that was used as part of the winner of the Yahoo! Learning to Rank challenge 2010. The algorithm runs in linear time in the number of queries, and can work with huge datasets.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; I.2.6 [**Learning**]: Parameter learning

General Terms

Algorithms

Keywords

Learning to rank, semi-supervised, partially labeled data, regularization, LambdaRank

1. INTRODUCTION

A ranking function, or ranker, is a function that orders items in a set based on their features. For example, a search engine orders the documents in a corpus based on features of document content and the search query, such as how many query words the document contains. The ranking function strives to maximize some ranking objective, e.g. to order documents by descending relevance to the query. Modern ranking systems can combine many features to build complex ranking functions. Learning to rank algorithms aim to learn ranking functions that achieve good ranking objectives on test data. Such learning methods require labeled data for training. In the case of search engines, the training data consist of queries, documents and the labels are human relevance judgments of documents with respect to queries. To learn a complex ranking function, a significant amount of labeled data may be required.

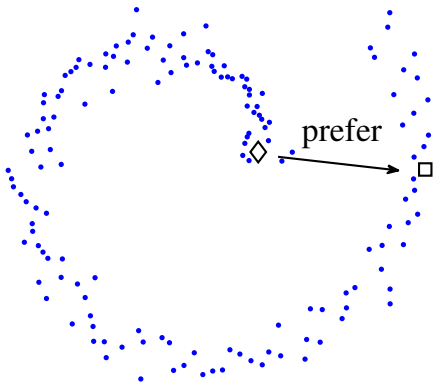

Figure 1: A semi-supervised ranking instance: labeled items, in the form of a preference pair (rhomb and square), and unlabeled items (dots).

In many cases, it is difficult to acquire labeled data, but easy to collect unlabeled training data. For example, unlabeled queries and documents can be obtained mechanically from query logs and web crawls, but the labeling requires paid human assessors. When building systems for small markets, such as for minority languages, specialized domains, or narrow query segments, it can be cost-ineffective to acquire sufficient label data. It can also be impractical to keep large label sets up to date; by the time judges have completed their assessments, some label information may no longer be fresh. As another example, when building personalized rankers for individuals or enterprises, we can only expect individuals to provide relatively few labeled examples.

A well-established retrieval scenario is *relevance feedback*, in which a user issues a query, and labels a few of the resulting documents from a (typically) hand-tailored ranker (BM25 or language modeling), but leaves most documents unlabeled.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

The ranker then incorporates the partially labeled results to produce refined results.

In this paper, we propose semi-supervised learning to rank methods. Such methods learn both from labeled as well as unlabeled data, so called partially labeled data. Commonly, there is a small amount of labeled data and a large amount of unlabeled data. By leveraging both types of data, the need for labeled examples can be reduced. In web search, the labeled data consists of labeled (judged) query-document features (or preference relationships between items), whereas the unlabeled data are simply query-document feature vectors without any labels (or preference information).

There is a wealth of work in semi-supervised classification [27, 28] and regression. These tasks involve predicting class labels or function values of individual items. The labels are assigned to individual items and are absolute. In contrast, there is comparatively little prior work in semi-supervised ranking. Ranking has the different goal of learning a function that orders multiple items correctly. In this setting, labeled data can be provided as relative preferences among items. Absolute labels are not required, but can still be used by inducing pairwise preferences.

The key to semi-supervised learning is a principle connecting the structure of the unlabeled data with the function to be learned. In classification, a commonly used principle is the cluster assumption [24], which states that data points in each high-density region (cluster) should have the same labels. Essentially, the unlabeled data define the extent of the clusters, whereas the labeled data determine the class of the clusters. In this paper, we formulate such a principle for ranking: we favor that similar documents are similar in preference to each other with respect to a query. We embody this principle in a preference regularizer that exploits unlabeled data. The regularizer tries to capture manifold structure in the data. The regularizer is general and can be applied to any paired comparison ranking model.

For many problems, we desire ranking functions that correctly order the top of the ranking in particular. For this purpose, we develop a rank-sensitive preference regularizer that emphasizes the influence of unlabeled data at the top of the ranking.

We apply this rank-sensitive regularizer to obtain a semi-supervised version of LambdaRank [7], a state-of-the-art learning to rank algorithm. LambdaRank can optimize NDCG, mean average precision, and many other popular ranking metrics. It has proven to be one of the most effective and practical algorithms and the winner of the Yahoo! Learning to Rank challenge 2010 [9] used an used an ensemble of Lambda-Gradient models.

Data and resources for this paper are available at `http://purl.org/net/semisupervised-ranking-cikm11.html`.

2. BACKGROUND

The semi-supervised ranking problem consists of a set of ranking instances. Each ranking instance is a set $X = \{\boldsymbol{x}_i \in \mathbb{R}^n\}$ of items with feature vectors, to which we would like to assign a ranking $\boldsymbol{r}$, a permutation of the numbers $1, \ldots, |X|$. Here r_i refers to the rank of item i and the top (preferred) rank is 1.

Figure 1 illustrates the semi-supervised ranking task. It depicts two labeled items, which by themselves offer very limited information for learning a ranking function. The distribution of unlabeled items suggests that there is additional structure in the data; for example, it would make sense for the ranking function to be smooth along the spiral shape.

In ranking, label information can take several forms, such as absolute labels (e.g. relevance grades), preference relations, complete orderings or partial orderings. We convert these by inducing pairwise preference relations of form "item i is preferred to j", and denoted by $i \succ j$. Then, let L be a set of pairs with given preferences, and U a set of items that will be used in an unsupervised way (and whose preferences may be unknown). We will typically include all available items in U and induce all possible pairs within a rank instance.

Our task is to learn a function $f(\boldsymbol{x}; \boldsymbol{w})$ with parameters $\boldsymbol{w}$ that ranks the items of a ranking instance X. We will consider rankers that assign a score $s_i = f(\boldsymbol{x}_i; \boldsymbol{w})$ to each item, where high scores indicate preference. A ranking can be produced simply by sorting the scores[1]. We will consider probabilistic models that assign a probability of preference $P(i \succ j)$, based on score differences. Such models are referred to as "paired comparison" models.

We will mostly employ the Bradley-Terry model [22], a long-standing model that has been successfully applied in learning to rank [6]. It associates a parameter s_i with each item, which can be thought of as a score. Then it defines the probability $P(i \succ j)$ that item i is preferred to j to be the logistic function of their score difference

$$P(i \succ j) = 1/(1 + \mathrm{e}^{-(s_i - s_j)}). \tag{1}$$

For example, if $s_i > s_j$, then $P(i \succ j) > 0.5$. By construction, pairwise preferences from the model are transitive, so that $P(i \succ j) > 0.5$ and $P(j \succ k) > 0.5$ implies $P(i \succ k) > 0.5$; in fact, $P(i \succ j)$ and $P(j \succ k)$ uniquely determine $P(i \succ k)$ [6].

Let L be a set of observed preferences, $L = \{i_1 \succ j_1, j_1 \prec i_1, i_2 \succ j_2, \ldots\}$ (where pairs (i,j) are included both ways (i,j) and (j,i)). We can estimate item scores s (or parameters $\boldsymbol{w}$) by maximizing the likelihood

$$C = \sum_{(i,j) \in L} \mathbb{I}_{i \succ j} \log P(i \succ j). \tag{2}$$

Here the indicator $\mathbb{I}_{i \succ j}$ denotes the observed pairwise preferences, and $\mathbb{I}_{i \succ j} = 1$ when $i \succ j$, and 0 otherwise.

For notational convenience, we have displayed just a single ranking instance (consisting of a single set of items, and a single rank ordering); in practice, we will have training data consisting of multiple item sets (queries) with associated orderings, $\{X^{(q)}, \boldsymbol{r}^{(q)}\}$, from which we learn a single ranking function f. We note that some of the ranking instances may be completely unsupervised. For example, in information retrieval, the training set consists of multiple queries each with an associated partial ranking of documents (where the partial ranking may be empty). For a new test query, the goal is to rank all the documents in the collection.

2.1 Related Work

Related semi-supervised work falls in three broad classes: self-training, feature extraction approaches, and graph-based regularization.

The self-training approach, termed *pseudo relevance feedback*, is very popular in retrieval. It assumes that the top k retrieved results from the ranker are relevant, and uses them to retrain the ranker. A variation of this is to take the results below rank k to be negative (non-relevant); this

[1]We resolve ties randomly.

is a fairly safe assumption, as the vast majority of items are non-relevant, and including such items improves ranker training in practice.

Other self-training approaches [2, 33] and co-training [21] first train a standard supervised ranker (RankBoost and RankNet&BM25 respectively) on the labeled items. They then apply a nearest neighbor classifier to assign labels to k unlabeled items that are the most similar to the labeled ones. Finally they retrain the supervised ranker including the newly labeled points, and repeat the process. It is important to control k to reduce the propagation of errors as noisily classified points accumulate in the training set.

The feature extraction approach [14] also employs an existing supervised ranker (RankBoost), but apply it not to the given features, but instead to the output of an unsupervised feature extractor (kernel PCA) trained separately on unlabeled data. The method thus involves two stages, and the structure extracted from unlabeled data is not guided by labeled data.

Graph-based regularization is another way to combine labeled and unlabeled data, via a manifold regularizer (the second term below) [5, 34, 12]. This is usually done in the context of regression, where we have

$$\min_f \sum_{i \in L} \|f(\boldsymbol{x}_i) - y_i\|^2 + \beta \sum_{i,j \in U} W_{ij} \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|^2. \quad (3)$$

This formulation fits the function f to target labels y_i while also making sure that function values of similar items $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close, according to a similarity measure W_{ij}, commonly defined in terms of graph Laplacians.

This is typically formulated as a transductive approach, meaning that unlabeled test data can (and must) be included at learning time, something difficult to do when the goal is to learn a model for ranking novel (unseen) queries. The approach can be extended to transductive ranking by handling input given as pairwise preference relations [26], and the function f can be a Gaussian process [11]. These approaches are currently defined only for a single ranking instance, corresponding to a single graph, and have high computational cost $O(|U|^3)$.

Rank-aggregation [10, 25] involves combining multiple ranked lists to produce a better overall ranking, and has been performed in a semi-supervised way by combining large-margin preference constraints with a manifold regularizer.

A preliminary version of the present work was presented in [30]. See also [29] for a broad overview that positions semi-supervised ranking in the context of cost-sensitive machine learning for information retrieval.

2.2 Motivations for Our Work

Our work is related to the graph-based regularization approaches, but formulates the regularizer specifically for paired comparison ranking models, rather than for regression models. It is also the first algorithm to satisfy all of the following properties:

Preference learning: labeled data is given as (or interpreted as) pairwise item preferences; it neither requires [ordinal] labels, nor assigns any [ordinal] labels. The output is simply a ranked list.

End-to-end optimization of ranking metrics: it directly optimizes most established ranking metrics (e.g. mean average precision, NDCG), including their emphasis of top rank positions.

Single-stage learning: The learning is done jointly on labeled and unlabeled data in a single stage. Information in the labeled data can guide the extraction of structure extracted from unlabeled data.

Multiple and completely unlabeled rank instances: in information retrieval, the training data usually comes grouped into queries, some of which have labeled and unlabeled documents, but some of which have exclusively unlabeled ones. Our method can utilize information from queries for which no labeled items are available.

Scalability: linear time in the number of queries, and approximately linear in the number of unlabeled items (section 5.1).

Performance: it extends one of the most effective learning to rank algorithms, LambdaRank, to the semi-supervised setting.

Generality: the regularization principle can be applied to many learning to rank algorithms and settings, beyond semi-supervised learning, such as cases with fully-labeled data in order to reduce overfitting, adaptation, etc.

In the following sections, we first introduce a preference regularizer for learning to rank. This main idea is generally applicable to most paired learning to rank algorithms. As LambdaRank is one of the strongest learning to rank algorithms, we illustrate how to apply the regularizer to it.

3. PREFERENCE REGULARIZATION

The key to semi-supervised learning is a principle connecting the structure of the unlabeled data with the function to be learned. In classification, a commonly used principle is the cluster assumption [24], which states that data points in each high-density region (cluster) should have the same labels. In regression, a principle is to assume that the function value changes slowly in high-density regions (as formalized by Eq. 3). In retrieval, it has been observed that closely related documents tend to be relevant to the same query [16, 32], which is termed the "cluster hypothesis".

We adapt the cluster hypothesis to a ranking context, and require that similar documents be similar in preference to each other, with respect to a query. Specifically, neither document should be much preferred to the other; ideally, they should tie in preference — an indifference of preference.

This preference similarity assumption is weaker than the classification one, as it only constrains relative item order, and not absolute score or class. For a pair of similar documents, it does not assume which document is preferred.

The two spaces we need to connect are: i) the feature space of the unlabeled data, and ii) the preference space of the ranking function. In the feature space, the type of structure we hope to exploit is manifold structure of the unlabeled data. To do this, we quantify the similarity between documents i and j as the probability of transitioning between them under a noise model. We base the transition probability on the exponentiated distance $d_{ij}^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / \sigma_i^2$ between

the (suitably normalized) document feature vectors, where σ_i is a length scale. However, we only allow transitions to K-nearest neighbors $N_K(i)$, as we want to follow the local manifold relying only on local distances. The probability of an i to j transition is

$$\hat{q}_{j|i} = \begin{cases} \mathrm{e}^{-d_{ij}^2/\sigma_i^2} / \sum_{k \in N_K(i)} \mathrm{e}^{-d_{ik}^2/\sigma_i^2} & \text{if } j \in N_K(i), \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

The probability of going there and back is then $\hat{q}_{ij} = \hat{q}_{i|j}\hat{q}_{j|i}/Z$, where Z normalizes $\hat{q}_{ij}$ to sum to one over all pairs.

In the preference space, we hope to constrain the preference structure of the ranking function. To do so, we quantify the probability that neither document is preferred to the other. This is given by $P(i \not\succ j)P(j \not\succ i)$, where $P(i \not\succ j) = 1 - P(i \succ j) = P(j \succ i)$. The lowest possible preference is a tie, when $P(i \succ j) = P(j \succ i) = 0.5$. These probabilities can be produced by the Bradley-Terry model, or any paired comparison ranking model. Via this model, the probabilities will depend on the ranking function with parameters $\boldsymbol{w}$.

To form the regularizer, we must link the feature space similarity to the preference similarity. Recall the manifold regularizer for regression (Eq. 3) that penalized the output difference between a pair of items (i, j) according to their input similarity. In the ranking case, for every pair (i, j) we have a probabilistic preference similarity, and a probabilistic input similarity. We shall penalize the difference between these two distributions according to their KL divergence. This penalty is the same as the stochastic neighbor embedding (SNE) objective [31], which ensures that probabilistic neighborhood relations in a high-dimensional space are preserved when the points are embedded in a low-dimensional space. One can think of this regularizer as penalizing bad embeddings from the input space to the low-dimensional preference space.

Definition. The *preference regularizer* is

$$\sum_{(i,j) \in U} \hat{q}_{ij} \log P(i \not\succ j)P(j \not\succ i). \quad (5)$$

It strongly penalizes models for which similar documents (high $\hat{q}_{ij}$) have dissimilar preference (probabilities close to 0 and 1, yielding a low product $P(i \not\succ j)P(j \not\succ i)$). However, it only weakly penalizes the converse: dissimilar documents can be similar in preference. This behavior springs from the asymmetry of the KL-divergence. It is desirable, as we can assume more about preferences between similar documents than about dissimilar ones.

4. SEMI-SUPERVISED BRADLEY-TERRY MODEL

To obtain a semi-supervised ranking model, we add the preference regularizer to a paired comparison likelihood, obtaining

$$C = \sum_{(i,j) \in L} \mathbb{I}_{i \succ j} \log P(i \succ j) + \beta \sum_{i,j \in U} \hat{q}_{ij} \log P(i \not\succ j)P(j \not\succ i). \quad (6)$$

The parameter β weights the preference regularizer against the likelihood. The combination of these two terms mirrors manifold regularization (Eq. 3).

The above is general and applies to any probabilistic ranking model that can be expressed in terms of paired comparison probabilities. For example, we can apply it to the Bradley-Terry model by substituting Eq. 1 to get

$$C = \sum_{(i,j) \in L} \mathbb{I}_{i \succ j} \log \frac{1}{1 + \mathrm{e}^{-(s_i - s_j)}} \quad (7)$$

$$+ \beta \sum_{i,j \in U} \hat{q}_{ij} \log \frac{0.5}{1 + \cosh(s_i - s_j)}. \quad (8)$$

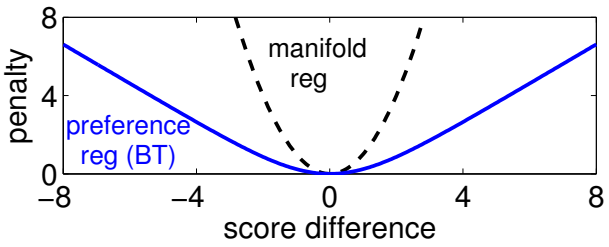

Figure 2: Comparison between the preference regularizer applied to the Bradley-Terry model and the manifold regularizer. We have subtracted a constant from the preference regularizer to align it with the manifold regularizer at the origin (constant offsets have no effect on training).

In the context of the Bradley Terry model, we see that the regularizer operates on score differences $(s_i - s_j)$, which it tries to reduce proportionally to $\hat{q}_{ij}$. We can directly compare it with the manifold regularizer, since that also operates on score differences. The manifold regularizer imposes a quadratic penalty, whereas the preference regularizer grows linearly for large score differences (Figure 2). The latter is milder. Importantly, our regularizer is matched to the ranking task and the chosen Bradley-Terry model, so it should nicely balance the likelihood; a score-difference for labeled data will be treated consistently with a score-difference in unlabeled data, which would not be the case if one added a manifold regularizer to a ranking likelihood.

Learning to Rank Algorithm. At this point, we can already design a simple semi-supervised learning to rank algorithm. Given scores, the Bradley-Terry model defines pairwise preference probabilities, but our task is to go from item features to a ranking. So, we could for example choose a neural network to assign the scores s_i for the Bradley-Terry model. To obtain a semi-supervised ranker, we then simply include the preference regularizer during training. We learn it by maximizing regularized likelihood (Eq. 7 and 8) via gradient descent with respect to $\boldsymbol{w}$.

Experiment on Spiral Data. Here we train a ranker on the spiral dataset from Figure 1. We choose the ranking function to be a 1-hidden layer neural network, with x and y coordinates as inputs, and 8 hidden units. First, we train the ranker using only labeled data. As shown in the contour plot in Figure 3 (left), we obtain a ranking function $f(\cdot)$ that decreases uniformly from left to right. This function correctly ranks the given labeled preference, but it ignores the structure in the unlabeled points.

Next, we add the preference regularizer, employing $K = 6$ nearest neighbors, and with length scale $\sigma_i = 0.2$ for all i (the input data ranges from -1 to 1 along both axes). Semi-supervised training then yields a ranking function that de-

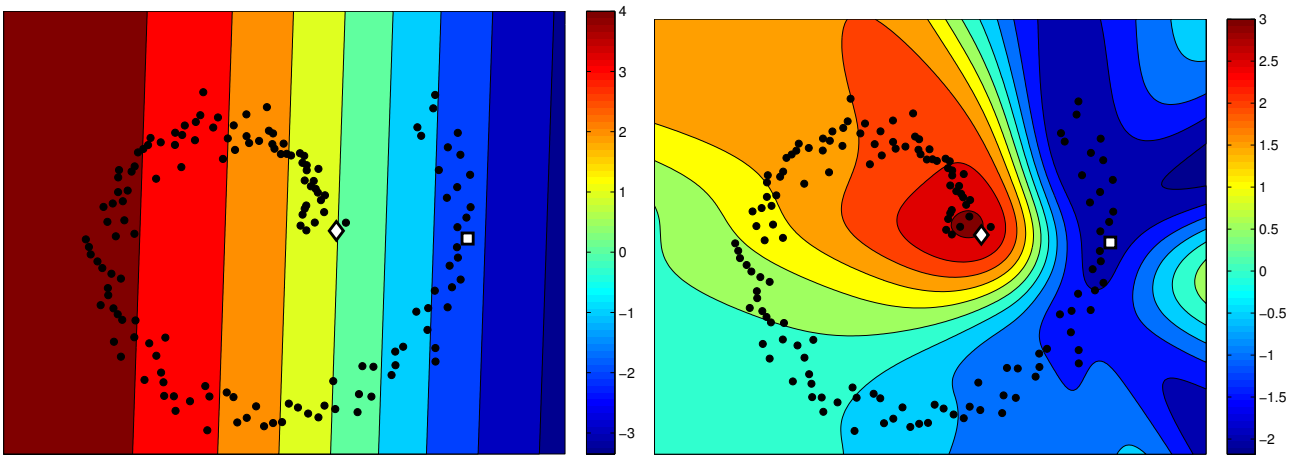

Figure 3: Left: Ranking function trained in a supervised way on a single preference pair (the rhomb is preferred to the square), ignoring unlabeled data. Right: ranking function trained in a semi-supervised way on a single preference pair plus unlabeled data.

creases smoothly along the spiral shape, while still satisfying the labeled preference (Figure 3, right).

This approach is adequate for generic ranking scenarios. However, we now move on to a more sophisticated approach, suitable to retrieval problems.

5. SEMI-SUPERVISED RANK-SENSITIVE BRADLEY-TERRY MODEL

In this section we extend the semi-supervised ranking framework to fit rank-sensitive objectives used in information retrieval, such as normalized discounted cumulative gain (NDCG) [17]. These objectives are rank-sensitive in that they take rank position of the documents into account. We focus on NDCG here, but note that it is straightforward to derive semi-supervised objectives for other popular objectives, such as mean average precision (MAP) and reciprocal rank (RR) metrics, using results from [13]. Ranking objectives give superior performance compared to classification or regression objectives in retrieval applications [23].

NDCG. The NDCG ranking metric captures two common requirements. Firstly, it focuses on the top of the ranking by weighting rank positions according to a decreasing discount function $R(r_i)$. Secondly, it caters for multi-grade relevance information given in terms of ordinal labels l_i for items i, by assigning a utility value $L(l_i)$ for each label. The relevance label l_i of a document can be 0, 1, 2, etc. depending on whether the document is nonrelevant, relevant, highly relevant, etc. for a query, respectively. Then, the NDCG objective is defined as

$$\text{NDCG} = \frac{1}{\text{DCG}_{\max}} \sum_i L(l_i) R(r_i), \tag{9}$$

where $\text{DCG}_{\max}$ is the maximum value of the sum achieved by an ideal ranking. $L(l_i)$ is usually referred to as a gain function which measures the utility ("gain") a user obtains by observing a document that has label (relevance) l_i. We use $L(l_i) = 2^{l_i} - 1$ as it is the most commonly used gain function in the literature [7].

A useful quantity is the change in the metric value if items i and j were to be swapped in the ranking. We denote this *swap cost* by $|\Delta_{ij}|$, and it is given by

$$|\Delta_{ij}| = |L(l_i)R(r_i) + L(l_j)R(r_j) - L(l_i)R(r_j) - L(l_j)R(r_i)|. \tag{10}$$

For notational convenience we will also absorb the $\text{DCG}_{\max}$ normalization into $|\Delta_{ij}|$. For unlabeled data, we introduce an unlabeled swap cost that depends only on the discount difference between items in the ranking (as there are no labels):

$$|\Delta^U_{ij}| = |R(r_i) - R(r_j)|. \tag{11}$$

Rank-sensitive Bradley-Terry. We incorporate the rank-sensitivity of retrieval metrics into the Bradley-Terry model by weighting item pairs in the likelihood by swap cost $|\Delta_{ij}|$, to produce the objective

$$C = \sum_{(i,j)\in L} |\Delta_{ij}|\, \mathbb{I}_{i\succ j} \log P(i \succ j). \tag{12}$$

The gradients of this objective coincide with the gradients that were manually crafted for the LambdaRank algorithm [7]. We note that LambdaRank successfully reaches optima of NDCG and MAP [13] using those gradients. This fact supports the design of the proposed rank-sensitive Bradley-Terry objective.

Rank-sensitive Preference Regularization. We would like the preference regularizer to be rank-sensitive in a similar way to the ranking objective above. Hence, we analogously weight item pairs in the preference regularizer by their unlabeled swap cost:

Definition. The *rank-sensitive preference regularizer* is given by

$$\sum_{i,j\in U} |\Delta^U_{ij}|\, \hat{q}_{ij} \log P(i \not\succ j) P(j \not\succ i). \tag{13}$$

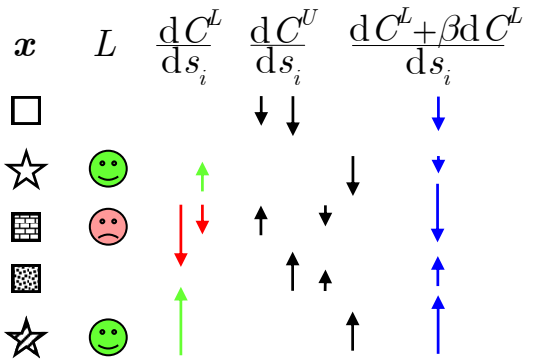

Figure 4: Example gradients during a run of SSLambdaRank. $\mathrm{d}C^L/\mathrm{d}s_i$ and $\mathrm{d}C^U/\mathrm{d}s_i$ refer to the gradients computed using labels and document similarities (unlabeled), respectively.

Semi-supervised Rank-sensitive Bradley-Terry Model. To obtain a semi-supervised ranking model, we add the rank-sensitive Bradley-Terry objective to the rank-sensitive regularizer:

$$C = \sum_{(i,j)\in L} |\Delta_{ij}|\, \mathbb{I}_{i\succ j} \log P(i \succ j) \quad (14)$$

$$+ \beta \sum_{i,j\in U} |\Delta^U_{ij}|\, \hat{q}_{ij} \log P(i \not\succ j) P(j \not\succ i), \quad (15)$$

with gradients

$$\mathrm{d}C/\mathrm{d}s_i = \sum_{\{j|(i,j)\in L\}} |\Delta_{ij}|\, (\mathbb{I}_{i\succ j} - P(i \succ j)) \quad (16)$$

$$+ \beta \sum_{j\in U} |\Delta^U_{ij}|\, \hat{q}_{ij}\, (0.5 - P(i \succ j)), \quad \forall i. \quad (17)$$

Comparing the labeled and unlabeled terms (Eq. 14 and 15), one sees that the document similarity probability $\hat{q}_{ij}$ serves a role similar to preferences $\mathbb{I}_{i\succ j}$ in the labeled data. One difference between the labeled and unlabeled gradients, is that the unlabeled gradient (Eq. 17) always tries to reduce the difference in ranks between i and j; the gradient sign is controlled by $0.5 - P(i \succ j)$ which is positive if $i \prec j$, or 0 if equal, or negative otherwise. In contrast, the labeled gradients (Eq. 16) can either increase or decrease the rank difference depending on the labels, as $\mathbb{I}_{i\succ j} - P(i \succ j)$ is positive when $i \succ j$ or negative when $j \succ i$ (or 0).

The gradients can be intuitively visualized as "forces" acting on each pair of documents (Figure 4), pushing the more relevant document up in the ranking, and the other one down. The resultant force on document i is a sum over forces from other documents j (as shown in Eq. 16 and 17). The forces depend on the current ranks and scores, as well as the labels of the documents. In the figure, the column marked $\boldsymbol{x}$ shows items in the training data, where items with similar shapes are assumed similar to each other (squares and stars). The L column shows the labels of the items in the training data (positive and negative), and $\mathrm{d}C^L/\mathrm{d}s_i$ shows the magnitude and direction of the labeled gradients. For each pair of items, positive items are pushed up and negative items are pulled down in the ranking. The force magnitude is dictated by the difference in rank between the two items. Similarly, $\mathrm{d}C^U/\mathrm{d}s_i$ shows the unlabeled gradients for the data. Similar items are pulled towards each other. The last column in the figure shows the total semi-supervised gradient.

Learning to Rank Algorithm: SSLambdaRank.

We now proceed to implement the semi-supervised rank-sensitive model above in a learning algorithm, which we will call SSLambdaRank. We choose the ranking function $s_i = f(\boldsymbol{x}_i; \boldsymbol{w})$ to be a 1-hidden layer neural network, although any differentiable function will do. We aim to calculate $\mathrm{d}C/\mathrm{d}\boldsymbol{w} = \sum_i (\mathrm{d}C/\mathrm{d}s_i)(\mathrm{d}s_i/\mathrm{d}\boldsymbol{w})$ for gradient descent training.

The challenge in learning the ranking function $f(\boldsymbol{x}; \boldsymbol{w})$ is that the ranks are discontinuous in the parameters $\boldsymbol{w}$; in particular, the ranking is determined by sorting the scores $\{s_i\}$. The swap costs $|\Delta_{ij}|$ and $|\Delta^U_{ij}|$ both depend on ranks, which implies that the gradients are discontinuous in the parameters $\boldsymbol{w}$.

LambdaRank [7] is an algorithm that successfully applies stochastic gradient descent in this discontinuous setting. Given a parameter state $\boldsymbol{w}$, it calculates item scores, sorts the scores to determine item ranks, calculates swap costs $|\Delta_{ij}|$, computes stochastic gradients, and takes a gradient step to update the parameters. The updated parameters may yield a discontinuous change in $|\Delta_{ij}|$, but the algorithm works well empirically. In fact, LambdaRank is state of the art in ranking: the winning entry in the Yahoo! Learning to Rank Challenge [9] used an ensemble of Lambda-Gradient models, including LambdaRank. SSLambdaRank is applied in the same way, but to the semi-supervised gradients.

5.1 A (Near) Linear Time Algorithm

In semi-supervised learning, we often want to use very large unlabeled datasets, thus computational efficiency is required. The proposed learning algorithms scale linearly in the number of queries. In our settings, the scalability is dominated by the number of queries, as the number of items per query is bounded by 1000 (the max number of documents the search engine returns per query), so overall we observe linear scaling behavior.

However, in other settings, scalability may be dominated by the number of items per query (both labeled and unlabeled). A naive implementation that considered all item pairs would scale quadratically in the number of items. Fortunately, our formulation regularizes only with respect to the K nearest neighbors of each item in the feature space, which reduces to linear scaling behavior. This also has the benefit of trusting the feature space dissimilarity d_X only in local neighborhoods, giving it a local manifold behavior.

We also require a preprocessing step to find the K nearest neighbors of every item. Here we can apply approximation techniques such as locality sensitive hashing, which can bucket items in near linear time, or use fast exact K-nearest neighbor techniques [4].

Finally, the learning algorithm requires sorting item scores to form the ranked list for each query. This step can be done in linear time via radix sort, given that the scores have finite precision.

6. EXPERIMENTAL RESULTS

6.1 Datasets

In order to demonstrate the quality of the SSLambdaRank algorithm, we use two different datasets: TREC 6, 7, 8 adhoc

	Features
1.	BM25
2.	LogBM25
3.	LM_ABS
4.	LM_DIR
5.	LM_JM
6.	LogNormalizedTF
7.	SumLogTF
8.	TF
9.	TF_IDF
10.	LogTF_IDF_V2
11.	NormalizedTF

Table 1: Feature Set

tracks, and the semi-supervised Million Query 2008 (MQ 2008) dataset [1] from LETOR 4.0 [23].

6.1.1 The TREC Ad-hoc Dataset

The first dataset we use consists of queries and documents from the TREC 6, 7 and 8 adhoc tracks, as set up by Aslam et al. [3]. The dataset has depth-100 pools from the TREC tracks, along with the associated relevance judgments. In total, there are 150 queries in the dataset, and approximately 1000 labeled documents per query.

The features used are those of LETOR 3.0, excluding web features that require link information, which is not available for this corpus. The features (Table 1) are computed over the document text with and without the title, resulting in 22 features in total. Detailed feature descriptions can be found in [23].

In order to avoid judging the entire document collection, the most commonly used technique is depth-m pooling, where in the case of depth-100 pooling, for example, only the top 100 documents retrieved by the systems are judged and the rest of the documents are assumed nonrelevant. We will simulate the effect of varying the number of labeled documents per query, in a similar spirit to depth pooling. For this, we include judgments for m of the labeled documents with the highest BM25 [20] score for each query, varying $m \in \{2, 3, 5, 10, 15\}$ and treating the rest of the documents as unlabeled. In this way, we include labels for likely relevant documents that appear towards the top of the ranking, as commonly done in the literature [23].

The distance between documents, $d_X()$ was computed using two different approaches: one content-based, the other feature-based. In the content-based approach, we first identified the top 50 words in the labeled documents with highest tf-idf score, and represented each document only using these terms, to compute Euclidean distance between such tf-idf document vectors. In the feature-based approach, we used the Euclidean distance between the ranking features (BM25, LogBM25, LM_ABS, etc.) for the documents. The goal of this was to examine whether semi-supervised learning could help even when exactly the same features were used for regularization as for ranking.

6.1.2 The Million Query 2008 Dataset

The second dataset we use is the semi-supervised Million Query 2008 (MQ 2008) query set [1] from LETOR 4.0 [23].

The MQ 2008 dataset uses the Gov2 web page collection (containing approximately 25M pages). It consists of 800 queries, with approximately 15000 labeled documents and 88000 unlabeled documents in total. The LETOR 4.0 data also provides cosine-similarities between tf-idf document vectors, which we directly used as our distances $d_X()$.

6.2 Experimental Setup

Both datasets were split in five parts: three for training, one for validation, and one for testing. The splits were done five ways for 5-fold cross validation. The documents in the training and validation sets are samples of the complete collection, as described above. The test set consists of the complete set of documents. We report averages across the 5 folds.

The neural network architecture was tuned for the supervised case where we only use the labeled documents (L *only*) by varying the number of hidden units; 3 hidden units performed best on the validation set. Even though the optimal number of hidden units for our semi-supervised algorithm may be larger, we used the same model size as the supervised LambdaRank algorithm to ensure the same learning capacity in both cases.

We also validated other important model parameters: β, the weight of labeled and unlabeled data, was varied between 0.1 and 5 in 0.1 increments, and the best value on the validation set was chosen. Likewise, the epoch for early stopping was validated. Other parameters were fixed to reasonable defaults: the learning parameters K=5, σ_i=∞ for all i, yielding $\hat{q}_{ij}$=0.2 for K neighbors or 0 otherwise. The effect of number of number of neighbors, K, will be analyzed in section 6.3.3.

RankBoost [15] and Ranking SVM [19] are two commonly used learning to rank algorithms. Semi-supervised RankBoost [2] and Transductive SVM (TSVM) [18] are semi-supervised variations of these algorithms. TSVM is really a classification algorithm, but is applicable since the collections have absolute judgments. We use these algorithms as baselines. For semi-supervised RankBoost, the weight of unlabeled documents needs to be specified (similar to our β parameter). To set this value for RankBoost, we used the same validation approach as we used for specifying the β parameter for our algorithm.

6.3 Results

We use two different retrieval tasks to compare the performance of different learning to rank algorithms: ranking novel queries, and a relevance feedback task. When ranking novel queries, training and test sets are separate. In the relevance feedback task, a user issues a query and labels a few of the resulting documents from a traditional ranker (e.g. BM25). Then the system trains a query-specific ranker incorporating the user feedback, and re-ranks the collection.

6.3.1 Ranking Novel Queries Task

The left plot in Figure 5 shows the quality of our algorithm on the TREC test set (averaged across all five folds). All LambdaRank variants are trained to optimize NDCG and the accuracy is measured using the NDCG metric at rank cutoff 10. Our semi-supervised algorithm is trained on varying numbers of labeled preference pairs from 90 training queries, and all documents associated with those queries (nearly 90,000) are used as unlabeled data. All LambdaRank methods work on preference pairs; a very large number of pairs are induced from the labels: about 3500 labeled pairs and over 40 million unlabeled pairs. This experiment thereby demonstrates the scalability of our algorithms.

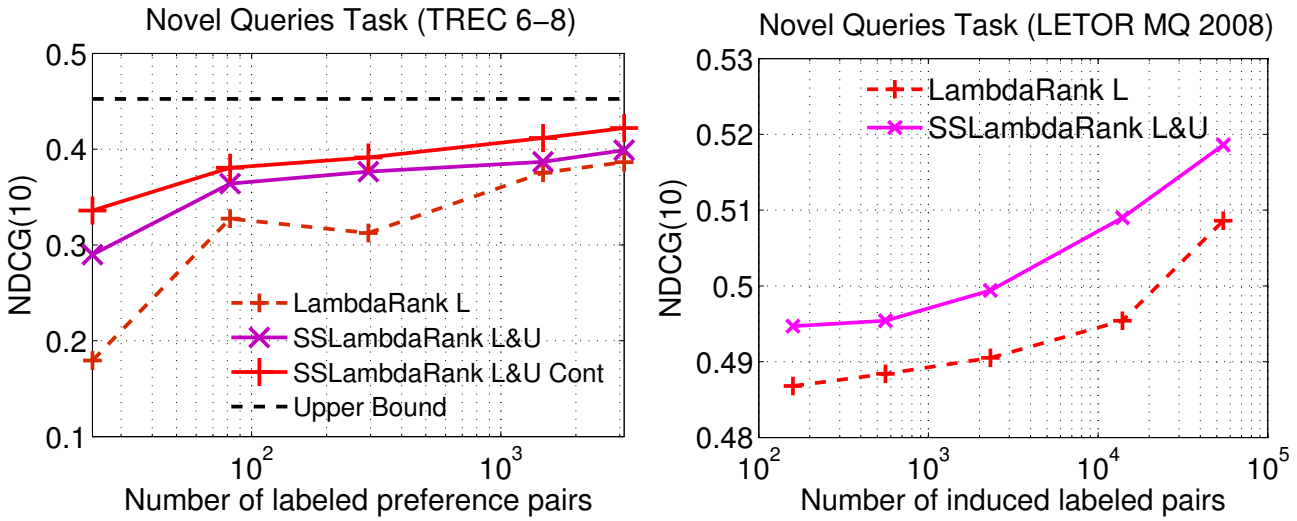

Figure 5: NDCG accuracy at rank cutoff 10 for the test set averaged over 5 folds for the (left) TREC and (right) Million Query 2008 datasets. We show supervised methods (dashed lines, suffix L) and semi-supervised methods (solid lines, suffix L & U). Feature-based SSLambdaRank has suffix (L & U), content-based SSLambdaRank has suffix (L & U Cont).

The *Upper bound*, is attained by taking all ∼150,000 docs and using them in supervised LambdaRank providing labels, which is "cheating" compared to semi-supervised methods which treat most of these documents as unlabeled. This corresponds to an ideal case in which the true labels of the unlabeled data became available.

The figures show that SSLambdaRank using either distance metric outperforms LambdaRank (*LambdaRank L*). Since the LambdaRank algorithms use preference pairs in training, the x axis in this plot displays the number of preference pairs (on a log scale) as opposed to the number of labeled documents per query. Except for the last data point (with the most labeled pairs), the differences between SSLambdaRank and LambdaRank are statistically significant in all cases according to the Wilcoxon sign-rank test at $p = 0.05$ significance level. We note that the semi-supervised ranker still benefits from unlabeled data despite the large number of labeled preferences.

The right plot of Figure 5 shows the quality of our algorithm on the LETOR MQ2008 dataset. In this experiment, we only use the document similarity values that were provided by the LETOR collection. To analyze the effect of varying the number of labeled examples, we form different training sets by sampling l% of the documents from each query ($l \in 5, 10, 20, 50, 100$), assuming the sampled documents are labeled and the rest of the documents are unlabeled. The x axis in the plot shows the number of induced pairs (on a log scale) when l% of documents are used from the complete set. SSLambdaRank beats LambdaRank consistently. All the improvements are statistically significant according to the Wilcoxon sign-rank test at $p = 0.05$ significance level. Since the LETOR MQ2008 dataset does not contain the labels for all documents (the collection itself is partially judged), no *Upper bound* information is available for this dataset.

The two semi-supervised baselines, semi-supervised RankBoost and Transductive SVM could not be used in this setting. The semi-supervised RankBoost code [2] does not apply to the problem of ranking unseen queries as it only works for a single rank instance (i.e., train on a single query, test on the same query). For both datasets, Transductive SVM was trained for more than two days without terminating. Our proposed algorithm can be trained in an hour even for a large dataset as the MQ 2008 collection.

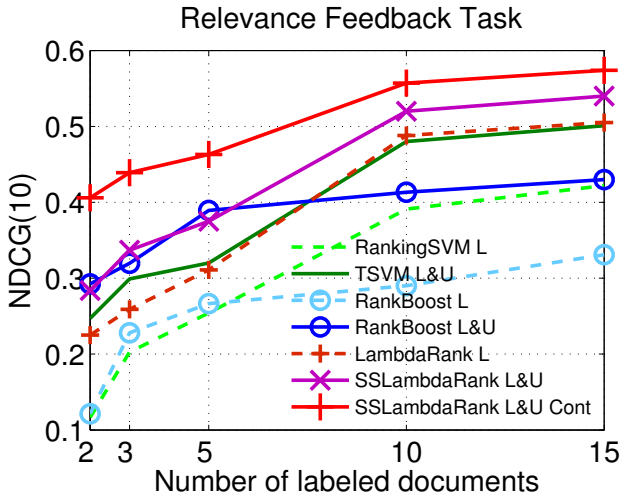

Figure 6: NDCG accuracy at rank 10 for the relevance feedback task. We show supervised methods (dashed lines, suffix L) and semi-supervised methods (solid lines, suffix L & U).

6.3.2 *Relevance Feedback Task*

Next, we focus on the relevance feedback task as this task is approachable by the other semi-supervised algorithms. Here, we can compare the quality of our semi-supervised algorithm with that of our two baselines (semi-supervised RankBoost

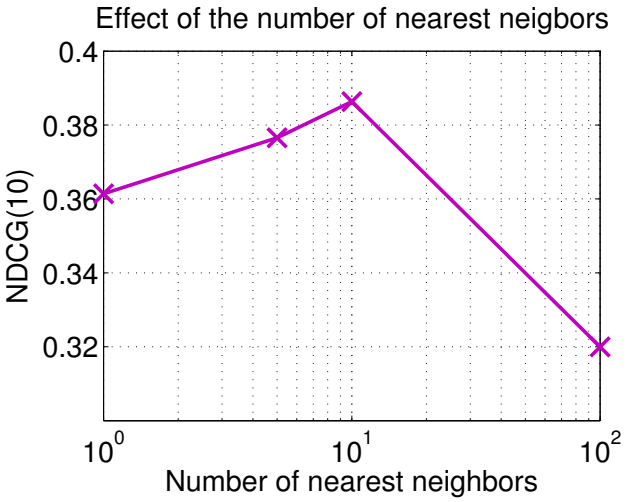

Figure 7: Effect of K, the number of nearest neighbors, on SSLambdaRank (L & U cont), for the novel queries task.

and Transductive SVM). In the relevance feedback task, a user issues a query and labels a few of the resulting documents from a traditional ranker (BM25). The system then trains a query-specific ranker incorporating the user feedback, and re-ranks the collection (test on the query the algorithm is trained on, similar to the assumption the semi-supervised RankBoost algorithm is based on). This is a transductive setting, as both labeled and unlabeled documents for the test query are available at training time. Since the training is done per query, the training data is much smaller. Hence, Transductive SVM can be run in a reasonable amount of time. For this task, we only report results from the TREC dataset.

Figure 6 shows retrieval accuracy for varying numbers of labeled training documents for the relevance feedback task. Since the relevance feedback experiment involves training on a single query and testing on the same query (excluding the training documents), we plot the mean NDCG across all queries. The figure shows that the SSLambdaRank using either distance metric outperforms LambdaRank. The differences between SSLambdaRank and LambdaRank are statistically significant in all cases according to the Wilcoxon sign-rank test at $p = 0.05$ significance level. Furthermore, our proposed algorithm consistently and significantly ($p = 0.05$) outperforms the other two semi-supervised algorithms.

6.3.3 *Effect of the Number of Nearest Neighbors (K)*

One important parameter in SSLambdaRank algorithm is the number of nearest neighbors K. Figure 7 explores the effect of K for the novel queries task using 300 labeled preference pairs per query for the TREC collection. $K = 10$ nearest neighbors results in the best performance.

Note that in all experiments we used $K = 5$ number of nearest neighbors. The performance of SSLambdaRank could be further improved had we used $K = 10$ neighbors.

7. DISCUSSION AND CONCLUSIONS

We proposed a preference regularizer for semi-supervised ranking; and showed how it could be used to extend LambdaRank to a semi-supervised setup. Experiments show that the proposed algorithm outperforms both the supervised version of itself as well as other available semi-supervised baselines.

Even though our semi-supervised algorithm can work with absolute labels, our guiding principle in this paper has been to design an algorithm for relative as well as absolute labels. We learn from neighborhood relations in the feature space, and preference relations in the preference space. Preferences are the essence of ranking, distinguishing it from classification and ordinal regression. Importantly, studies show that humans produce more consistent rankings and do so quicker when giving preferences rather than absolute labels [8]. Preference relations arise naturally as user choices from lists (e.g. clicks on search results). Thus, there are strong justifications for algorithms in this category.

We could have been more purist by optimizing a preference metric (e.g. ppref) rather than NDCG that employs absolute labels. The algorithm is designed for this setting, but preference metrics are not well-established yet.

The regularizer exploits structure in the unlabeled data. We have chosen to focus on manifold structure. However, the question remains what part of the manifold should be included; equivalently, what pairs of data points should be used in the regularizer? We included unlabeled data retrieved at the top of the ranking; since common ranking objectives emphasize top ranks, unlabeled data in that region regularizes the ranking function where it matters the most. Random items would generally be non-relevant and spread out thinly in the space, having little in common with each other. With the rank-sensitive regularizer, one could include all unlabeled data, as data at the bottom would be discounted anyway.

Future work could explore the effectiveness of different label information: *depth* (many preferences for few rank instances (queries)) vs. *breadth* (few preferences for many rank instances (queries)), as well as preferences within top vs. across top-bottom for a ranking instance. It may also be interesting to apply the regularizer with differentiable hypothesis classes other than neural networks, in particular random forests.

Acknowledgements

We thank Massih Reza Amini for making the semi-supervised RankBoost [2] implementation available, and Tom Minka and Vishwa Vinay for helpful discussions.

8. REFERENCES

[1] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. In *The Sixteenth Text REtrieval Conference Proceedings (TREC 2008)*. National Institute of Standards and Technology, 2009.

[2] M. R. Amini, T. V. Truong, and C. Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *SIGIR Conf. Research and Development in Information Retrieval*, 2008.

[3] J. A. Aslam, E. Kanoulas, V. Pavlu, and E. Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *SIGIR Conf. Research and Development in Information Retrieval*, 2009.

[4] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proc. Intl. conf. World Wide Web (WWW)*, pages 131–140, 2007.

[5] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 2006.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Intl. Conf. on Machine Learning (ICML)*, 2005.

[7] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[8] B. Carterette, P. Bennett, D. Chickering, and S. Dumais. Here or there: Preference judgments for relevance. In *European Conf. Information Retrieval (ECIR)*, 2008.

[9] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Proc. Yahoo! Learning to Rank Challenge 2010*, 2011.

[10] S. Chen, F. Wang, Y. Song, and C. Zhang. Semi-supervised ranking aggregation. *Information Processing & Management*, 47:415–425, May 2011.

[11] W. Chu and Z. Ghahramani. Extensions of gaussian processes for ranking: Semi-supervised and active learning. In *NIPS workshop on Learning to Rank*, 2005.

[12] F. Diaz. Regularizing query-based retrieval scores. *Information Retrieval*, 10(6):531–562, 2007.

[13] P. Donmez, K. M. Svore, and C. J. Burges. On the local optimality of Lambdarank. In *SIGIR Conf. Research and Development in Information Retrieval*, 2009.

[14] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR Conf. Research and Development in Information Retrieval*, 2008.

[15] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[16] N. Jardine and C. J. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7:217–240, 1971.

[17] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR Conf. Research and Development in Information Retrieval*, 2000.

[18] T. Joachims. Transductive inference for text classification using support vector machines. In *Intl. Conf. on Machine Learning (ICML)*, pages 200–209, 1999.

[19] T. Joachims. Optimizing search engines using clickthrough data. In *Intl. conf. Knowledge discovery and data mining (KDD)*, pages 133–142, 2002.

[20] K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. In *Information Processing & Management*, pages 779–840, 2000.

[21] M. Li, H. Li, and Z.-H. Zhou. Semi-supervised document retrieval. *Information Processing & Management*, 45(3):341–355, 2009.

[22] J. Marden. *Analyzing and Modeling Rank Data*. Chapman & Hall, 1995.

[23] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval Journal*, 2010.

[24] M. Seeger. Learning with labeled and unlabeled data. Technical Report, Dec. 2002.

[25] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. LambdaMerge: Merging the results of query reformulations. In *Conf. Web search and data mining (WSDM)*, pages 795–804, 2011.

[26] A. Shivani. Ranking on graph data. In *Intl. Conf. on Machine Learning (ICML)*, 2006.

[27] M. Szummer and T. Jaakkola. Kernel expansions with unlabeled examples. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 626–632. MIT Press, 2001.

[28] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pages 945–952. MIT Press, 2002.

[29] M. Szummer and F. Radlinski. Cost-sensitive machine learning for information retrieval. In Krishnapuram, Yu, and Rao, editors, *Cost-sensitive Machine Learning*. Chapman and Hall/CRC, 2011.

[30] M. Szummer and E. Yilmaz. Semi-supervised ranking via ranking regularization. In *Advances in Ranking, workshop at NIPS*, 2009.

[31] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[32] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd ed edition, 1979.

[33] J.-N. Vittaut and P. Gallinari. Supervised and semi-supervised machine learning ranking. In *Comparative Evaluation of XML Information Retrieval Systems*, volume 4518 of *Lecture Notes in Computer Science*, pages 213–222. Springer, 2007.

[34] C. Wang, E. Yilmaz, and M. Szummer. Relevance feedback exploiting query-specific document manifolds. In *Conf. Information and Knowledge Management (CIKM)*, 2011.