Incremental Learning to Rank with Partially-Labeled Data

Kye-Hyeon Kim
Department of Computer Science
POSTECH, Korea
fenrir@postech.ac.kr

Seungjin Choi
Department of Computer Science
POSTECH, Korea
seungjin@postech.ac.kr

ABSTRACT

In this paper we present a semi-supervised learning method for a problem of learning to rank where we exploit Markov random walks and graph regularization in order to incorporate not only "labeled" web pages but also plenty of "unlabeled" web pages (click logs of which are not given) into learning a ranking function. In order to cope with scalability which existing semi-supervised learning methods suffer from, we develop a scalable and incremental method for semi-supervised learning to rank. In the graph regularization framework, we first determine features which well reflects data manifold and then make use of them to train a linear ranking function. We introduce a matrix-fee technique where we compute the eigenvectors of a huge similarity matrix without constructing the matrix itself. Then we present an incremental algorithm to learn a linear ranking function using features determined by projecting data onto the eigenvectors of the similarity matrix, which can be applied to a task of web-scale ranking. We evaluate our method on *Live Search query log*, showing that search performance is much improved when *Live Search* yields unsatisfactory search results.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*web search*; H.4.m [**Information Systems Applications**]: Miscellaneous—*machine learning*

General Terms

Algorithms, Experimentation, Theory

Keywords

Information retrieval, Learning to rank, Web search, Clickthrough data, Semi-supervised learning, Incremental learning

1. INTRODUCTION

WSCD '09, Feb 9, 2009 Barcelona, Spain.

A user's implicit feedback such as click logs plays an important role in improving ranking performance in a search engine, since click logs are easy to collect but well reflect users' relevance judgement. Extensive research has been performed on how to utilize those click logs to improve search performance (see [18] and references therein), where the fundamental assumption is that "a user tends to click a web page if it looks interesting or relevant to the given query". In [18] for instance, each click log entry is converted into a pairwise preference "a clicked web page is preferred over a non-clicked web page" and then used as training data for *preference learning* algorithms [5, 8, 11].

While those methods have proven their effectiveness in many recent studies, we think click-through data alone is not sufficient to obtain a good ranking function. A commercial search engine may have billions of click logs, so it seems that there is enough user feedback. However, most of the click logs are focused on the top several results for each query, since most users scan only the first few pages of results. That is, among infinitely many indexed web pages, only a few of them have user feedback for a query.

This *small sample problem* can be more serious in *personalized search*. In this search service, each user has his own ranking function that is trained by his feedback. However, for most of users, there are not enough click logs to obtain correct ranking functions. The similar problem is there in recommender systems, called *cold start problem*.

1.1 Previous Work

Semi-supervised learning can solve the small sample problem by using both labeled and unlabeled data points to train a target function.[1] The key idea of semi-supervised learning is to use unlabeled samples for recognizing the underlying clusters or the latent feature space of data. To this end, one should construct the *edge-weight matrix* $\boldsymbol{K}$, whose (i, j)-th element $[\boldsymbol{K}]_{ij}$ denotes the pairwise base similarity between the i-th and the j-th items. Then, one can utilize the underlying feature information by performing *label propagation* [10, 12, 17, 21], or by computing the inverse or the principal components of the *graph Laplacian* [1, 7]. Although most research has focused on classification (see [22] and references therein), several algorithms have been proposed for learning to rank [1, 2, 7, 21], including *PageRank*-based algorithms [10, 12, 17]:

[1]Now we call a web page *labeled* if one or more related click log entries are given, and *unlabeled* otherwise. Clearly, most of the indexed web pages in a search engine is unlabeled.

- Amini et al. [2] proposed another semi-supervised *RankBoost* algorithm, assuming k nearest unlabeled neighbors of a labeled item have the same label with the labeled one, so that incorporating those "pseudo-labeled" pairs into training data. The limitation is that it can utilize only a small portion of unlabeled data (k nearest neighbors of labeled data).
- Agarwal [1] incorporated graph regularization [4, 20] into the existing ranking formulation based on support vector machines (*RankSVMs*) [11, 18]. The ranking model can be trained by optimizing the standard dual problem of *kernel RankSVMs*, where the kernel function is the (pseudo-)inverse of the graph Laplacian.
- Duh and Kirchhoff [7] proposed a novel re-ranking framework. Whenever a query is given, the initial list of results is retrieved by a standard method such as TF-IDF-based cosine similarity [15]. Then the proposed algorithm re-ranks the results by (1) projecting them onto their kernel principal directions (*Kernel PCA*), and then (2) training the boosting model (*RankBoost*) [8] using the labeled pairs among the projected items.

The major **problem** with semi-supervised learning is its poor scalability, due to the following reasons:

1. Most of the proposed algorithms require the $N \times N$ edge-weight matrix $\boldsymbol{K}$, where N is the number of whole (i.e. both labeled and unlabeled) data points. Since N is very large in most real-world applications, $\boldsymbol{K}$ should be a sparse matrix such that the number of non-zero elements doesn't exceed $\mathcal{O}(N)$. Commonly, the sparsity can be fulfilled by k-nearest neighbors such that "$[\boldsymbol{K}]_{ij} > 0$ if the i-th item is one of the k-nearest neighbors of the j-th item or vice versa". It guarantees that the number of non-zero elements in $\boldsymbol{K}$ scales linearly with N, but takes $\mathcal{O}(N^2)$ time to construct $\boldsymbol{K}$.
2. If an algorithm requires the (pseudo-)inverse of the graph Laplacian matrix [1], it scales as $\mathcal{O}(N^3)$ in time and $\mathcal{O}(N^2)$ in space, so it cannot be applied to web-scale data.
3. When an algorithm uses label propagation [21], each iteration "$\boldsymbol{f} \leftarrow \alpha \boldsymbol{K}\boldsymbol{f} + (1-\alpha)\boldsymbol{y}$" scales linearly with N, so the scalability is much better. However, it is not an *incremental* algorithm. More specifically, whenever the label vector $\boldsymbol{y}$ is modified by new click logs (e.g. a zero element $[\boldsymbol{y}]_i$ is set to 1 since a user clicks the i-th item), the preference vector $\boldsymbol{f}$ should be recomputed to reflect them.

To avoid those problems, several algorithms [7, 21] try to "learn in runtime". In [7], the algorithm runs with the retrieved results for the "current query", rather than with the whole dataset, so that makes the number of data points N to be small. In [21], the algorithm constructs a label vector for the current query, and then performs label propagation, so that avoids to compute the inverse of the graph Laplacian. However, both algorithms have very poor scalability in testing time, because the whole learning process should be performed whenever a query is given by a user.

To our knowledge, the current best alternative for semi-supervised web search is *PageRank* [10, 12, 17], since they can avoid the above problems as follows:

1. They use hyperlinks of web pages as the edges $[\boldsymbol{K}]_{ij}$. It is sparse enough, and well-known to be an appropriate base similarity measure. We can obtain $\boldsymbol{K}$ by simply parsing hyperlinks in each web page without any other computation, so we can avoid the first problem.
2. They use label propagation, so the second problem doesn't matter.
3. They compute multiple base preference vectors $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_H$ where H is the number of representative web pages (also called *hub pages*). Those vectors are then combined with the preference vector $\boldsymbol{f} = \sum_i w_i \boldsymbol{f}_i$, and user's click logs are now used to optimize the weight parameters $w_1, \ldots, w_H$ rather than $\boldsymbol{f}$ itself. It is not hard to derive an incremental algorithm for learning weights, so the third problem can be solved.

In this way, PageRank algorithms can successfully utilize unlabeled data for web search. However, they have limitations in that (1) one cannot apply any state-of-the-art similarity measure [13] other than the link structure, and (2) usr's click logs are represented as the weights of the base preference vectors, meaning that a variety of user interest is restricted to given hub pages.

1.2 Proposed Method

In this paper, we present an efficient semi-supervised rank learning algorithm that solves the above three problems by achieving the following **goals**:

1. The algorithm can utilize unlabeled data *without constructing* the edge-weight matrix $\boldsymbol{K}$, while we can still obtain the same result as if we explicitly use $\boldsymbol{K}$.
2. Instead of the inverse of the graph Laplacian, our method uses the *rank-R approximation* (i.e. dimensionality reduction from $\mathbb{R}^N$ to $\mathbb{R}^R$, where $N >> R$) so that it takes much less space, $\mathcal{O}(RN)$. Also, we propose an efficient algorithm to compute the rank-R approximation, which scales linearly in time with the number of given queries m, where $m << N$.
3. The algorithm can efficiently update a trained ranking model whenever new click log entries are given by users. To this end, we propose to derive an *incremental learning* algorithm, which applies each training sample iteratively rather than all samples simultaneously.

The proposed method consists of the following two steps:

1. The **projection algorithm** projects data points $\boldsymbol{x} \in \mathbb{R}^D$ onto the *underlying low-dimensional space* $\mathcal{M} \in \mathbb{R}^R$ ($R \leq D$), such that the similarity between data points $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is preserved as the inner product of the projected points $\boldsymbol{z}_i, \boldsymbol{z}_j \in \mathcal{M}$, i.e., $k(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \boldsymbol{z}_i^\top \boldsymbol{z}_j$. The similarity measure k is derived from the theory of Markov random walks which initiates *manifold learning* [3] and *graph-based semi-supervised learning* [4, 20, 21].
2. The **training algorithm** then trains the ranking function $f : \mathcal{M} \to \mathbb{R}$, of the form $f(\boldsymbol{x}_i) = \boldsymbol{u}^\top \boldsymbol{z}_i$, which is linear and defined on the latent space $\mathcal{M}$. We formulate the *graph regularization framework* [20] for *preference learning* [5, 8, 11], and derive an efficient training

algorithm. Even after the ranking function is trained with a given set of labeled data, users still continuously give new clicks (i.e. labeled data) *in runtime*, thus the training algorithm should be *incremental* so that the ranking function can be updated whenever new click log entries are given.

Our method is similar to *RankSVMs with graph regularization* [1] in that graph regularization is incorporated into the rank learning formulation, but different in that our method is based on the (regularized) linear ranking model with efficient training and incremental updating algorithms. Our method is also similar to the *re-ranking framework* [7] in that the ranking function is defined on the low-dimensional latent space rather than the original space, but different in that our method doesn't need to recompute the whole learning process for each search task, so that advantages over the re-ranking framework in query response time.

2. MARKOV RANDOM WALKS AND GRAPH REGULARIZATION

In this section, we briefly review the theory of *Markov random walks* and *graph regularization*. Both are closely related, and have initiated many research topics such as manifold learning [3] and graph-based semi-supervised learning ([22] and references therein), including our proposed method.

2.1 Markov Random Walks

Markov random walks consider a random walker on a graph, where the random walker moves from the current node to other nodes according to edges and the corresponding transition probabilities. The theory derives the *similarity* between two nodes on the graph, in terms of the probability distribution of the current position of the random walker. With the neighborhood graph of a given data set, the similarity measure well reflects the overall distribution of the data.

A graph is constructed from a given dataset $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, where each node represents each data point. The nodes are connected by *weighted edges* if they are sufficiently close each other in the Euclidean space (higher weights for closer pairs), so that the graph represents a skeleton of data manifolds. More specifically, an $N \times N$ *edge-weight matrix* $\boldsymbol{K}$ is constructed such that $[\boldsymbol{K}]_{ij} > 0$ if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close enough, and $[\boldsymbol{K}]_{ij} = 0$ otherwise. Commonly, $\boldsymbol{K}$ is considered as symmetric (i.e. $[\boldsymbol{K}]_{ij} = [\boldsymbol{K}]_{ji}$ for all $i, j = 1, \ldots, N$), and normalized as

$$\boldsymbol{K} = \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{W} \boldsymbol{D}^{-\frac{1}{2}}, \tag{1}$$

where $\boldsymbol{W}$ is the unnormalized edge-weight matrix, and $\boldsymbol{D}$ is diagonal such that $[\boldsymbol{D}]_{ii} = \sum_{ij} [\boldsymbol{W}]_{ij}$.

The similarity between two nodes is defined by "how easy to visit" one node from another on the graph. That is, a path consisting of highly weighted edges increases the similarity, and more paths between a pair generally yields greater similarity of the pair. Both cases are only possble in a "dense region" (i.e. cluster), meaning that this similarity measure well reflects the manifold structure of data.

More specifically, (1) the weight of a path is defined by the multiplication of all weights in the path, so that $[\boldsymbol{K}^t]_{ij}$ is the sum of weights of all paths between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ whose length is t. For example, when $t = 3$, the paths of the form $(\boldsymbol{x}_i, \boldsymbol{x}_k, \boldsymbol{x}_\ell, \boldsymbol{x}_j)$ for all $k, \ell = 1, \ldots, N$ are only counted in the sum such that $\sum_{k,\ell=1}^{N} [\boldsymbol{K}]_{ik} [\boldsymbol{K}]_{k\ell} [\boldsymbol{K}]_{\ell j}$, which is equal to $[\boldsymbol{K}^3]_{ij}$; (2) Similarly, weights of all possible paths between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ can be counted according to their path lengths, yielding $[\boldsymbol{I}]_{ij}$ (identity matrix for 0-length paths), $[\boldsymbol{K}]_{ij}$, $[\boldsymbol{K}^2]_{ij}$, and so on.

Consequently, the similarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, denoted by $[\widetilde{\boldsymbol{K}}]_{ij}$, is the weighted sum of all those weights:

$$\widetilde{\boldsymbol{K}} = \boldsymbol{I} + \alpha \boldsymbol{K} + \alpha^2 \boldsymbol{K}^2 + \alpha^3 \boldsymbol{K}^3 + \cdots . \tag{2}$$

A decaying factor α, within $0 < \alpha < 1$, makes α^t to be smaller for larger t so that contributions of longer paths decrease. The similarity matrix $\widetilde{\boldsymbol{K}}$ is also called the *von Neumann diffusion kernel* [19], and can be rewritten as the following closed form:

$$\widetilde{\boldsymbol{K}} = (\boldsymbol{I} - \alpha \boldsymbol{K})^{-1}, \tag{3}$$

or $\widetilde{\boldsymbol{K}} = (1 - \alpha)(\boldsymbol{I} - \alpha \boldsymbol{K})^{-1}$ with normalization.

The *PageRank* algorithm [16] is a well-known application of the diffusion kernel. For N web pages $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, assume that the next page is visited by clicking an arbitrary link in the current page. That is, $\boldsymbol{K}$ is defined such that $[\boldsymbol{K}]_{ij} = \frac{1}{o_j}$ if $\boldsymbol{x}_j$ has an outgoing link to $\boldsymbol{x}_i$ (o_j is the number of outgoing links in $\boldsymbol{x}_j$), and $[\boldsymbol{K}]_{ij} = 0$ otherwise. Then, $[\boldsymbol{K}^t]_{ij}$ is the probability of visiting $\boldsymbol{x}_i$ from $\boldsymbol{x}_j$ by t random clicks, and the weighted average $[\widetilde{\boldsymbol{K}}]_{ij}$ represents how similar $\boldsymbol{x}_i$ is to $\boldsymbol{x}_j$. With the normalizing factor $1 - \alpha$, it can also be considered as the conditional probability $\mathcal{P}(\boldsymbol{x}_i | \boldsymbol{x}_j)$, since $\sum_{i=1}^{N} [\widetilde{\boldsymbol{K}}]_{ij} = 1$ for all $j = 1, \ldots, N$.

2.2 Graph Regularization

Graph regularization framework [1, 4, 20] is to incorporate a regularization term into the empirical loss function, where the regularizer restricts the candidate model to be "smooth", in terms of the *smoothness functional* [4]. On a graph, the smoothness functional of a function f can be discretized as $\mathcal{S}(f) = \sum_{i,j=1}^{N} [\boldsymbol{I} - \boldsymbol{K}]_{ij} f(\boldsymbol{x}_i) f(\boldsymbol{x}_j)$, where $\boldsymbol{K}$ is normalized as in Eq. (1). For binary classification (i.e. $y_i \in \{+1, 0, -1\}$, where $y_i = 0$ means unlabeled) with the squared loss function, the framework formulates the following optimization problem

$$\arg\min_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f}) \;=\; \frac{1}{2} \boldsymbol{f}^\top (\boldsymbol{I} - \boldsymbol{K}) \boldsymbol{f} + \frac{\mu}{2} ||\boldsymbol{y} - \boldsymbol{f}||^2, \tag{4}$$

where $\mu > 0$ is the regularization parameter, and $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)]^\top$. By setting the gradient $(\partial \mathcal{L})/(\partial \boldsymbol{f})$ to $\boldsymbol{0}$, we have the solution $\boldsymbol{f} = (1 - \alpha)(\boldsymbol{I} - \alpha \boldsymbol{K})^{-1} \boldsymbol{y} = \widetilde{\boldsymbol{K}} \boldsymbol{y}$ where $\alpha = \frac{1}{\mu + 1}$.

3. THE PROPOSED ALGORITHM

Based on the theory in Section 2, now we derive our method, which consists of the projection step and the training step.

3.1 Projection Step

In Section 1.1, we noted that obtaining the exact similarity matrix $\widetilde{\boldsymbol{K}}$ is infeasible for large N due to the inversion. Hence, we use the rank-R approximation of $\widetilde{\boldsymbol{K}}$, obtained from the R largest eigenvalues (denoted by $\lambda_1, \ldots, \lambda_R$) and

their eigenvectors $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R)$ of $\boldsymbol{K}$:

$$\widetilde{\boldsymbol{K}} = (\boldsymbol{I}_N - \alpha \boldsymbol{K})^{-1} \approx \sum_{i=1}^{R} \frac{1}{1-\alpha\lambda_i} \boldsymbol{v}_i \boldsymbol{v}_i^\top = \boldsymbol{V}_R (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-1} \boldsymbol{V}_R^\top, \quad (5)$$

where $\boldsymbol{I}_N$ denotes the $N \times N$ identity matrix, $\boldsymbol{\Lambda}_R$ denotes the $R \times R$ diagonal matrix such that $[\boldsymbol{\Lambda}_R]_{ii} = \lambda_i$, and $\boldsymbol{V}_R$ denotes the $N \times R$ matrix $[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R]$. Note that $\boldsymbol{K}$ is a sparse matrix, such that the number of non-zero elements doesn't exceed $\mathcal{O}(N)$. Hence, the principal components $\boldsymbol{\Lambda}_R$ and $\boldsymbol{V}_R$ can be efficiently computed by iterative algorithms such as *power iteration* and *Lanczos methods* [6].

In Section 1.2, we mentioned that the goal of the projection step is to obtain the projected data points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N \in \mathcal{M}$, where the proposed similarity $\widetilde{\boldsymbol{K}}$ is preserved as the inner product in the underlying low-dimensional space $\mathcal{M} \in \mathbb{R}^R$ such that $[\widetilde{\boldsymbol{K}}]_{ij} \approx \boldsymbol{z}_i^\top \boldsymbol{z}_j$ (i.e. $\mathcal{M}$ is the latent Euclidean space of web pages). Denoting by $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$, we have

$$\widetilde{\boldsymbol{K}} \approx \boldsymbol{Z}^\top \boldsymbol{Z}. \quad (6)$$

Combining Eq. (5) and Eq. (6), the optimal $\boldsymbol{Z}$ in the best rank-R sense is

$$\boldsymbol{Z} = (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-\frac{1}{2}} \boldsymbol{V}_R^\top. \quad (7)$$

In this way, we can obtain the latent space $\mathcal{M} \in \mathbb{R}^R$. Eq. (7) implies that the most important part in the projection step is to compute R principal components of $\boldsymbol{K}$. Now we describe how to obtain R principal components of $\boldsymbol{K}$ *without constructing* $\boldsymbol{K}$, as we emphasized at the first goal in Section 1.2.

Assume that m queries $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m$ are given, where $\boldsymbol{q}_i \in \{1, \ldots, N\}^T$ contains its top-T similar web pages. For example, $\boldsymbol{q}_i = [21, 1919, \ldots, 188]$ means that the 21st, 1919th, ..., and 188th web pages are the top-T results of $\boldsymbol{q}_i$. One can use any commercial search engine to determine those web pages for each query. In this paper, we used *Live search SDK*. Then, we define the i-th web page as $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{iD}]$, where

$$x_{ik} = \begin{cases} 1 & \text{if } i \in \boldsymbol{q}_k, \\ 0 & \text{otherwise,} \end{cases} \quad (8)$$

and the edge weight $[\boldsymbol{K}]_{ij}$ between two web pages $\boldsymbol{x}_i, \boldsymbol{x}_j$ as

$$[\boldsymbol{K}]_{ij} = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{||\boldsymbol{x}_i||\ ||\boldsymbol{x}_j||}, \quad (9)$$

where $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ represents the number of queries whose top-T results contain $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ simultaneously.

Note that Eq. (9) is a plausible base similarity measure: Each top-T list $\boldsymbol{q}_i$ is determined by a commercial search engine, which uses various state-of-the-art measures and techniques to obtain the list, thus a pair $\boldsymbol{x}_i, \boldsymbol{x}_j$ can be seen as closer if more queries contain both of them.

From the above definitions, now we can compute the principal components of $\boldsymbol{K}$ using *power iteration* with deflation. The core part in the power iteration is the multiplication of $\boldsymbol{K}\boldsymbol{v}$. Let Q_i be a set of queries containing $\boldsymbol{x}_i$, i.e., $Q_i = \{k \mid i \in \boldsymbol{q}_k\}$. Then, the multiplication of $\boldsymbol{K}\boldsymbol{v}$ for a vector $\boldsymbol{v} \in \mathbb{R}^N$ can be computed as

$$[\boldsymbol{K}\boldsymbol{v}]_i = \sum_{[\boldsymbol{K}]_{ij}>0} [\boldsymbol{K}]_{ij} [\boldsymbol{v}]_j = \sum_{k \in Q_i} \sum_{j \in \boldsymbol{q}_k} \frac{[\boldsymbol{v}]_j}{||\boldsymbol{x}_i||\ ||\boldsymbol{x}_j||}, \quad (10)$$

Table 1: The eigen decomposition algorithm

For $r = 1, \ldots, R$, do the follows until $\boldsymbol{v}_r$ converges :
1. $\boldsymbol{b} \leftarrow \boldsymbol{0}$ /* $\boldsymbol{b} = \boldsymbol{K}\boldsymbol{v}_r$ */
2. For $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m$:
   $c = \sum_{j \in \boldsymbol{q}_k} [\boldsymbol{v}_r]_j / ||\boldsymbol{x}_j||$
   $[\boldsymbol{b}]_i \leftarrow [\boldsymbol{b}]_i + c/||\boldsymbol{x}_i||$ for all $i \in \boldsymbol{q}_k$
4. $\boldsymbol{v}_r \leftarrow \boldsymbol{b} - \sum_{t=1}^{r-1} \lambda_t (\boldsymbol{v}_r^\top \boldsymbol{v}_t) \boldsymbol{v}_t$
5. $\boldsymbol{v}_r \leftarrow \boldsymbol{v}_r / ||\boldsymbol{v}_r||$

After $\boldsymbol{v}_r$ converges, do 1∼2 and then $\lambda_r = \boldsymbol{v}_r^\top \boldsymbol{b}$

where the latter equation doesn't need the $\boldsymbol{K}$ matrix. Table 1 summarizes the eigen decomposition algorithm. It takes mT time to compute $\boldsymbol{K}\boldsymbol{v}$. Note that computing $\boldsymbol{K}\boldsymbol{v}$ takes $\mathcal{O}(kN)$ time if $\boldsymbol{K}$ is constructed by k-nearest neighbors. Our algorithm can save $\mathcal{O}(N^2)$ time for $\boldsymbol{K}$. Also, $\boldsymbol{K}\boldsymbol{v}$ can be more efficiently computed, since the number of queries m is generally smaller than the number of indexed web pages N.

We can also obtain the principal components of the normalized edge weight matrix $[\boldsymbol{K}]_{ij}/(\sqrt{d_i}\sqrt{d_j})$ as in Eq. (1), where

$$d_i = \sum_{[\boldsymbol{K}]_{ij}>0} [\boldsymbol{K}]_{ij} = \sum_{k \in Q_i} \sum_{j \in \boldsymbol{q}_k} \frac{1}{||\boldsymbol{x}_i||\ ||\boldsymbol{x}_j||}. \quad (11)$$

From the above equation, $d_1, \ldots, d_N$ can be pre-computed by Step 1∼2 in Table 1 with substituting $[\boldsymbol{v}_r]_j$ (in Step 2) by 1. After obtaining $d_1, \ldots, d_N$, we can obtain the principal components of the normalized version, using the normalizing term $\sqrt{d_i}||\boldsymbol{x}_i||$ rather than $||\boldsymbol{x}_i||$ in Step 2.

3.2 Training Step

After obtaining the latent feature vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N \in \mathcal{M}$ in the projection step, the ranking function $f : \mathcal{M} \to \mathbb{R}$, having the form

$$f(\boldsymbol{x}_i) = \boldsymbol{u}^\top \boldsymbol{z}_i, \quad (12)$$

is trained in this step. Note that the similarity measure $[\widetilde{\boldsymbol{K}}]_{ij}$ is approximated as the inner product $\boldsymbol{z}_i^\top \boldsymbol{z}_j$, which means that the latent space $\mathcal{M}$ can be seen as Euclidean. Hence, f doesn't need to be non-linear.

3.2.1 Label Vector Configuration

The goal of this training step is to find the optimal weight vector $\boldsymbol{u}$ for a given set of click log entries. Suppose that a click log entry is converted into a preference label "$\boldsymbol{x}_a$ is preferred over $\boldsymbol{x}_b$" (denoted by $\boldsymbol{x}_a \succ \boldsymbol{x}_b$), where $\boldsymbol{x}_a$ is a clicked web page and $\boldsymbol{x}_b$ is not. Then, we formulate the learning problem as *preference learning* [5, 8, 11], where the i-th label $[\boldsymbol{y}]_i$ indicates a pairwise preference between the i-th pair $(\boldsymbol{x}_{a_i}, \boldsymbol{x}_{b_i})$:

$$[\boldsymbol{y}]_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_{a_i} \succ \boldsymbol{x}_{b_i}, \\ -1 & \text{if } \boldsymbol{x}_{b_i} \succ \boldsymbol{x}_{a_i}, \\ 0 & \text{unlabeled (no click log).} \end{cases} \quad (13)$$

In classification, each $[\boldsymbol{y}]_i$ corresponds to $\boldsymbol{x}_i$ one by one. In preference learning, however, a label is defined by *one-to-two relations*, i.e., each $[\boldsymbol{y}]_i$ corresponds to a pair of data points $(\boldsymbol{x}_{a_i}, \boldsymbol{x}_{b_i})$. Note that the index i DOES NOT indicate "the i-th click log entry". It indicates the i-th pair, where the ordering of pairs is predefined as *lexicographical order*. More specifically,

- every pair (a_i, b_i) should satisfy $a_i < b_i$, and
- any two pairs (a_i, b_i) and (a_j, b_j), where $i < j$, should satisfy either "$a_i < a_j$" or "$a_i = a_j$ and $b_i < b_j$".

Here is an example for $N = 4$ (i.e. there are 4 web pages):

- The lexicographical order derives 6 pairs: $(1,2)$, $(1,3)$, $(1,4)$, $(2,3)$, $(2,4)$, and $(3,4)$.
- A label vector $\boldsymbol{y} = [1, -1, 0, 0, 0, 1]^\top$ means that there are 3 labeled pairs $\boldsymbol{x}_1 \succ \boldsymbol{x}_2$, $\boldsymbol{x}_3 \succ \boldsymbol{x}_1$, and $\boldsymbol{x}_3 \succ \boldsymbol{x}_4$.

For a given pair of indices (a, b), the corresponding element in $\boldsymbol{y}$ referfing to that pair, denoted by $[\boldsymbol{y}]_i$, can be found by the following formula:[2]

$$\begin{aligned} i &= b - a + \sum_{j=1}^{a-1}(N-j) \\ &= \{(2N-1)a - a^2\}/2 + b - N. \end{aligned} \tag{14}$$

For example, when $N = 10$ (i.e. there are 10 web pages), the corresponding label for the pair $(a = 4, b = 7)$ is $[\boldsymbol{y}]_{27}$.

Given N web pages, there are ${}_N C_2 = N(N-1)/2$ possible pairs of distinct web pages, thus the dimension of $\boldsymbol{y}$ scales as $\mathcal{O}(N^2)$, which is definitely large. However, since only a small fraction of those pairs are labeled, most of the elements in $\boldsymbol{y}$ are 0 as in Eq. (13). Thus, the actual space taken by $\boldsymbol{y}$ is quite small, which scales linearly with the number of labeled pairs.

3.2.2 Algorithm

Now we derive the algorithm to find the optimal weight vector $\boldsymbol{u}$, where $f(\boldsymbol{x}_i) = \boldsymbol{u}^\top \boldsymbol{z}_i$ is consistent with $\boldsymbol{y}$ such that

$$[\boldsymbol{y}]_i = f(\boldsymbol{x}_{a_i}) - f(\boldsymbol{x}_{b_i}) = (\boldsymbol{z}_{a_i} - \boldsymbol{z}_{b_i})^\top \boldsymbol{u} \tag{15}$$

for all i. Denoting by $\boldsymbol{g}_i$ an N-dimensional vector such that

$$[\boldsymbol{g}_i]_j = \begin{cases} 1 & \text{if } j = a_i, \\ -1 & \text{if } j = b_i, \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

then Eq. (15) can be rewritten as $[\boldsymbol{y}]_i = (\boldsymbol{Z}\boldsymbol{g}_i)^\top \boldsymbol{u}$, since $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$ and $\boldsymbol{Z}\boldsymbol{g}_i = \boldsymbol{z}_{a_i} - \boldsymbol{z}_{b_i}$. Hence, denoting by $\boldsymbol{G} = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_M]$, we have

$$\boldsymbol{y} = (\boldsymbol{Z}\boldsymbol{G})^\top \boldsymbol{u}. \tag{17}$$

Now we reformulate Eq. (4) for preference learning. First, we restrict f to be linear, i.e., $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)]^\top = \boldsymbol{Z}^\top \boldsymbol{u}$. Also, we replace the squared error term $||\boldsymbol{y} - \boldsymbol{f}||^2$ with $||\boldsymbol{y} - (\boldsymbol{Z}\boldsymbol{G})^\top \boldsymbol{u}||^2$. Then, Eq. (4) can be rewritten as[3]

$$\mathcal{L}(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u} + \frac{\mu}{2}||\boldsymbol{y} - (\boldsymbol{Z}\boldsymbol{G})^\top \boldsymbol{u}||^2. \tag{18}$$

Setting $(\partial\mathcal{L})/(\partial\boldsymbol{u}) = \boldsymbol{0}$ and denoting by $\boldsymbol{W} = \boldsymbol{Z}\boldsymbol{Z}^\top$ and $\boldsymbol{w} = \sum_i \boldsymbol{z}_i$, the optimal weight vector $\boldsymbol{u}$ has the form (see Appendix A for details)

$$\boldsymbol{u} = \left(\frac{1}{1-\alpha}\boldsymbol{I}_R + N\boldsymbol{W} - \boldsymbol{w}\boldsymbol{w}^\top\right)^{-1} \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}. \tag{19}$$

Note that $\left(\frac{1}{1-\alpha}\boldsymbol{I}_R + N\boldsymbol{W} - \boldsymbol{w}\boldsymbol{w}^\top\right)$ is an $R \times R$ matrix, where R is the dimension of the latent feature space $\mathcal{M}$. Since R is generally much smaller than N (the number of web pages) or D (the number of original features of an web page — usually the number of terms), the matrix inversion is not so expensive. Also, the remaining part $\boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}$ can be computed efficiently as

$$\boldsymbol{Z}\boldsymbol{G}\boldsymbol{y} = \sum_{[\boldsymbol{y}]_i = 1} (\boldsymbol{z}_{a_i} - \boldsymbol{z}_{b_i}) + \sum_{[\boldsymbol{y}]_i = -1} (\boldsymbol{z}_{b_i} - \boldsymbol{z}_{a_i}). \tag{20}$$

3.2.3 Incremental Algorithm

In real web search services, users continuously give new clicks even after a ranking function is trained using the previous click logs. Thus, we should consider the update of the ranking function "in runtime" whenever new click log entries are given.

As we mentioned in Section 1.1, some methods [21] should discard the current model and then execute the whole algorithm again to reflect new query logs. It is too inefficient, so we propose an incremental version of Eq. (19).

First, we reformulate Eq. (19) as $\boldsymbol{u} = \boldsymbol{U}\boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}$, where $\boldsymbol{U} = \left(\frac{1}{1-\alpha}\boldsymbol{I}_R + N\boldsymbol{W} - \boldsymbol{w}\boldsymbol{w}^\top\right)^{-1}$. Note that $\boldsymbol{W} = \boldsymbol{Z}\boldsymbol{Z}^\top$ and $\boldsymbol{w} = \sum_i \boldsymbol{z}_i$ only depend on $\boldsymbol{Z}$, hence $\boldsymbol{U}$ is independent of $\boldsymbol{y}$. Denoting by $\widetilde{\boldsymbol{Z}} = [\widetilde{\boldsymbol{z}}_1, \ldots, \widetilde{\boldsymbol{z}}_N] = \boldsymbol{U}\boldsymbol{Z}$, we have

$$\boldsymbol{u} = \widetilde{\boldsymbol{Z}}\boldsymbol{G}\boldsymbol{y} = \sum_{[\boldsymbol{y}]_i \neq 0} [\boldsymbol{y}]_i (\widetilde{\boldsymbol{z}}_{a_i} - \widetilde{\boldsymbol{z}}_{b_i}). \tag{21}$$

Hence, whenever a new click log entry modifies $[\boldsymbol{y}]_i$, we can efficiently update $\boldsymbol{u}$ by simply adding $[\boldsymbol{y}]_i(\widetilde{\boldsymbol{z}}_{a_i} - \widetilde{\boldsymbol{z}}_{b_i})$, where the addition takes only $\mathcal{O}(R)$ time.

A decaying factor $0 < \kappa < 1$ can also be introduced to increase the influence of more recent click logs:

$$\boldsymbol{u} \leftarrow \kappa\boldsymbol{u} + [\boldsymbol{y}]_i(\widetilde{\boldsymbol{z}}_{a_i} - \widetilde{\boldsymbol{z}}_{b_i}). \tag{22}$$

3.2.4 Query-Dependent Ranking

The correct ranking of web pages can vary with queries. Thus, we use multiple weight vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_\mathcal{T}$ for the ranking function f such that

$$f(\boldsymbol{x}_i|\boldsymbol{q}) = \sum_{k=1}^{\mathcal{T}} \mathcal{P}(k|\boldsymbol{q})\boldsymbol{u}_k^\top \boldsymbol{z}_i, \tag{23}$$

where $\mathcal{P}(k|\boldsymbol{q})$ denotes the probability that the query $\boldsymbol{q}$ is related to the cluster k, where a cluster can be seen as a specific topic or something. That is, $0 \leq \mathcal{P}(k|\boldsymbol{q}) \leq 1$ for all k and $\boldsymbol{q}$, and $\sum_{k=1}^{\mathcal{T}} \mathcal{P}(k|\boldsymbol{q}) = 1$ for all $\boldsymbol{q}$. Now the training algorithm Eq. (22) is also modified as

$$\boldsymbol{u}_k \leftarrow \kappa\boldsymbol{u}_k + \mathcal{P}(k|\boldsymbol{q})[\boldsymbol{y}]_i(\widetilde{\boldsymbol{z}}_{a_i} - \widetilde{\boldsymbol{z}}_{b_i}). \tag{24}$$

To compute the probability $\mathcal{P}(k|\boldsymbol{q})$, we use *soft k-means clustering* [14]. Note that a query $\boldsymbol{q}$ can be represented as a feature vector using its top T results, i.e.,

$$\widetilde{\boldsymbol{q}} = \sum_{i \in \boldsymbol{q}} \boldsymbol{z}_i. \tag{25}$$

[2] Note that $(N - \frac{1}{2})a - \frac{1}{2}a^2$ involves floating point operations, so $\{(2N-1)a - a^2\}/2$ is more efficient to compute.

[3] One may argue that the objective function Eq. (18) doesn't need the smoothness term $\boldsymbol{u}^\top \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u}$, since $f : \mathcal{M} \to \mathbb{R}$ is "already smooth" by defining on the latent space $\mathcal{M}$ derived as in Section 3.1. Actually, the term does not smooth f any longer. Instead, it plays a role as the *L_2-norm (ridge) regularizer* [9] in $\mathcal{M}$ (see Appendix B for details).

Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{\mathcal{T}}$ denote $\mathcal{T}$ clusters of given queries $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m$. Then, using Eq. (25), we have

$$\mathcal{P}(k|\boldsymbol{q}) = \frac{\exp\{-\beta||\widetilde{\boldsymbol{q}} - \boldsymbol{\mu}_k||^2\}}{\sum_{\ell=1}^{\mathcal{T}} \exp\{-\beta||\widetilde{\boldsymbol{q}} - \boldsymbol{\mu}_\ell||^2} \quad (26)$$

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{m} \mathcal{P}(k|\boldsymbol{q}_i)\widetilde{\boldsymbol{q}}_i}{\sum_{i=1}^{m} \mathcal{P}(k|\widetilde{\boldsymbol{q}}_i)}, \quad (27)$$

where $\beta > 0$ is a parameter. The results close to "hard" clustering (i.e. $\mathcal{P}(k|\boldsymbol{q})$ would be 1 or 0) if $\beta \to \infty$.

3.3 Summary

In this section, we proposed several algorithms to achive three goals in Section 1.2: Table 1 for the first goal, Eq. (7) for the second goal, and Eq. (21) for the last goal. Here is the whole process of our method:

[Learning] Step 1: Projection

1. Obtain principal components $\boldsymbol{\Lambda}_R = \text{diag}(\lambda_1, \ldots, \lambda_R)$ and $\boldsymbol{V}_R = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R]$ by Table 1.
   [Parameters] T: # of relevant web pages per query
2. Obtain the projected data points $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$ by Eq. (7).
   [Parameters] α: decaying factor, R: dimension of the latent space
3. Compute $\boldsymbol{U}$ as the above of Eq. (21) and then obtain $\widetilde{\boldsymbol{z}}_i = \boldsymbol{U}\boldsymbol{z}_i$.
4. Obtain the query clusters $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{\mathcal{T}}$ by repeating Eq. (26) and (27) until converge.
   [Parameters] β: hardness factor, $\mathcal{T}$: # of clusters

[Learning] Step 2: Training

1. Initilize $\mathcal{T}$ weight vectors $\boldsymbol{u}_1 = \cdots = \boldsymbol{u}_{\mathcal{T}} = \boldsymbol{0}$.
2. Whenever a query $\boldsymbol{q}$ and its click log entries $\boldsymbol{x}_a \succ \boldsymbol{x}_b$ are given, update $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{\mathcal{T}}$:
   (a) Compute the probability $\mathcal{P}(k|\boldsymbol{q})$ as in Eq. (26).
   [Parameter] $\beta, \mathcal{T}$ in Step 1.4
   (b) Update $\boldsymbol{u}_k$ as in Eq. (24).
   [Parameter] κ: decaying factor

[Ranking]

1. When a query is given, obtain the list of the top-20 web pages (denoted by $\boldsymbol{q}$) from a commercial search engine, e.g. *Live Search*.
2. Compute the probability $\mathcal{P}(k|\boldsymbol{q})$ as in Eq. (26).
   [Parameter] $\beta, \mathcal{T}$ in Step 1.4
3. Obtain the ranking function value $f(\boldsymbol{x}_i)$ as in Eq. (23) for all $i \in \boldsymbol{q}$.
4. Sort the web pages in descending order of $f(\boldsymbol{x}_i)$.

4. EXPERIMENTS

We applied our method to the real-world web search service, *MSN Live Search*. As Table 1 shows, our projection algorithm needs (1) a list of the whole queries, and (2) the top T search results of each query $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m$. Thus, we first gathered all distinct query strings from *Live Search query log*, and assigned the unique ID $(1, \ldots, m)$ to each query. There are m = 3,875,427 distinct queries in *Live Search query log*.

Then, we obtained top 20 URLs for each query (i.e. T = 20) using *Live Search SDK*. All distinct URLs were also collected with unique IDs, but there are too many URLs (more than 40 million) to use all of them. Thus, we selected a portion of URLs occurring in top-20 lists of more than 10 queries, and reassigned unique IDs $(1, \ldots, N)$ to those selected URLs, where N = 627,819.

For training or testing, we used 4,175,543 click log entries among the whole 12,251,067 entries in *Live Search query log*, choosing a click log entry if (1) the corresponding clicked URL is one of the N selected URLs and (2) the given rank of the clicked URL is smaller than or equal to 20. For our training algorithm, we converted each click log entry into a set of labeled samples of the form $\boldsymbol{x}_a \succ \boldsymbol{x}_b$, where $\boldsymbol{x}_a$ is a clicked URL, and $\boldsymbol{x}_b$ is not clicked but in the top 20 results. That is, one click log entry yields 19 (if $\boldsymbol{x}_a$ is in the top-20) or 20 labeled samples.

4.1 Search Accuracy

For performance evaluation, we performed "re-ranking" (as in Section 3.3) the list of top 20 search results given by *Live Search*, including the clicked URL, then compared between the predicted rank and the given rank of the clicked URL. If the predicted rank (determined by our model) is higher than the given rank (determined by *Live Search*), we can conclude that semi-supervised learning is actually effective.

For example, suppose that a clicked URL $\boldsymbol{x}_a$, whose given rank is the 3rd, is ranked as the 1st by our method. Then, we say that our algorithm is "better", and the "improvement" is 2. Suppose that the URL is ranked as the 6th by our method. Then, we say that our algorithm is "worse", and the "improvement" is –3. The below table is the summary of comparison:

Group	Avr. Imprv.	% Better	% Worse
1∼5	–2.343	6.48	48.2
6∼20	0.8027	62.9	32.6
6∼10	0.7924	62.8	32.6
11∼20	8.273	87.8	9.96

Each group is a set of click log entries, where the given ranks of the clicked URLs are within the specified range. For example, the second group contains click log entries where the given ranks of the clicked URLs are within 6∼20. We trained our model with the first group (thus the second group is the test set), where the parameters were set as follows: $T = 20$, $\alpha = 0.95$, $R = 200$, $\beta = 1000$, $\mathcal{T} = 100$, and $\kappa = 1$.

The result of the first group disappointed us, implying that it is very hard to improve the accuracy of well-ordered search results. For the rest of the groups, however, our algorithm was better than *Live Search*, especially for poorly-ordered search results. It means that our method can complementary cooperate with *Live Search* as an additional ranking measure.

However, we still feel that the ranking performance of our method should be further improved, since only a few click log entries are in the second group. More specifically, 3,957,292 entries are in the first group (1∼5), and 218,251 entries are in the second group (6∼20). For the last group (11∼20)

in which our method worked very well, there are only 302 entries.

4.2 Time

Now we evaluate the learning and testing time of our algorithm. The experiments were done on a 3.2GHz Pentium 4 CPU and 2.0GB of RAM.

Note that the projection step (Step 1 in learning, see Section 3.3) can be pre-processed, since we need not recompute the outputs $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$, $\widetilde{\boldsymbol{Z}} = [\widetilde{\boldsymbol{z}}_1, \ldots, \widetilde{\boldsymbol{z}}_N]$ and $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{\mathcal{T}}$ once they are obtained. We performed 500 power iterations with deflation for each eigenvector, and 50 iterations of soft k-means for query clusters. The below table is the running time of the projection step:

Operation	Time (min.)
$R = 200$ principal components $\boldsymbol{\Lambda}_R, \boldsymbol{V}_R$	2383
Projected data points $\boldsymbol{Z}$ and $\widetilde{\boldsymbol{Z}}$	≤ 1
$\mathcal{T} = 100$ query clusters $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{\mathcal{T}}$	1486
Total	3870

Here is the running time of the rest of the steps in our method:

Step	Time (sec.)	Time/click[4](ms.)
Training step	≈ 3265.64	≈ 0.8252
Ranking	≈ 2325.68	≈ 0.5570

Note that the recent re-ranking framework for semi-supervised rank learning [7] took several hundred seconds for each query on the LETOR dataset [13]. Our method is 5 or 6 orders of magnitude faster than that method in query response time even on much larger data set.

5. CONCLUSIONS

In this paper, we proposed a semi-supervised learning method for web search, which can use both labeled (user click logs) and unlabeled data to train a ranking function, by utilizing top-T search results given by *Live Search*. The algorithm is applicable to real-world web search engines by solving scalability problems of existing semi-supervised learning methods (Section 1).

Based on the theory of Markov random walks, the diffusion kernel $[\widetilde{\boldsymbol{K}}]_{ij}$ was derived as a similarity measure, and an efficient algorithm (Table 1) was developed for computing the rank-R approximation of $[\widetilde{\boldsymbol{K}}]_{ij}$ without constructing the edge-weight matrix $\boldsymbol{K}$. Also, we incorporated the graph regularization framework into preference learning (Eq. (18)), and then developed an incremental training algorithm (Eq. (24)).

Experiments on *Live Search query log* showed that our method was definitely inferior to existing search engines, while it can still be useful as a cooperative ranking measure for refining some poor search results. Such failure may be due to the rank-R approximation or the small number of the selected URLs, or something else. As a future work, we will investigate the reasons for the failure to improve the ranking performance.

6. ACKNOWLEDGMENTS

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D00939) and Microsoft Research Asia.

[4]We used 4,175,543 click log entries for ranking, and 3,957,292 entries among them for training.

7. REFERENCES

[1] S. Agarwal. Ranking on graph data. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

[2] M. Amini, V. Truong, and C. Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *Proceedings of the ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR)*, 2008.

[3] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003.

[4] M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine Learning*, 56:209–239, 2004.

[5] 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 International Conference on Machine Learning (ICML)*, pages 89–96, Bonn, Germany, 2005.

[6] D. Calvetti, L. Reichel, and D. C. Sorensen. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 2:1–21, 1994.

[7] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *Proceedings of the ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR)*, 2008.

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

[9] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[10] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, 2002.

[11] R. Herbrich, T. Graepel, P. Bollman-Sdorra, and K. Obermayer. Learning preference relations for information retrieval. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI)*, 1998.

[12] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 271–279, 2003.

[13] T. Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of the SIGIR-2007 Workshop on Learning to Rank for Information Retrieval*, Amsterdam, Netherlands, 2007.

[14] D. J. C. MacKay. *Information Theory, Inferecne, and Learning Algorithms*. Cambridge University Press, 2003.

[15] C. D. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2007.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd. The

pagerank citation ranking: Brining order to the web. Technical report, Stanford University, 1998.

[17] F. Qiu and J. Cho. Automatic identification of user interest for personalized search. In *Proceedings of the 15th International Conference on World Wide Web*, pages 727–736, 2006.

[18] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 239–248, 2005.

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

[20] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16, pages 321–328. MIT Press, 2004.

[21] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16, pages 169–176. MIT Press, 2004.

[22] X. Zhu. Semi-supervised learning literature survey. Technical report, Computer Science TR 1530, University of Wisconsin-Madison, 2007.

APPENDIX

A. OPTIMAL WEIGHT VECTOR

We have the following objective function as in Eq. (18)

$$\mathcal{L}(\boldsymbol{u}) \;=\; \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u} + \frac{\mu}{2}||\boldsymbol{y} - (\boldsymbol{Z}\boldsymbol{G})^\top \boldsymbol{u}||^2,$$

and its gradient with respect to the weight vector $\boldsymbol{u}$:

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{u})}{\partial \boldsymbol{u}} &= \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u} - \mu \boldsymbol{Z}\boldsymbol{G}\Big(\boldsymbol{y} - (\boldsymbol{Z}\boldsymbol{G})^\top \boldsymbol{u}\Big) \\ &= \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u} - \mu \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y} + \mu \boldsymbol{Z}\boldsymbol{G}\boldsymbol{G}^\top \boldsymbol{Z}^\top \boldsymbol{u} \\ &= \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K} + \mu \boldsymbol{G}\boldsymbol{G}^\top)\boldsymbol{Z}^\top \boldsymbol{u} - \mu \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}. \qquad (28)\end{aligned}$$

Setting $(\partial \mathcal{L})/(\partial \boldsymbol{u}) = \boldsymbol{0}$ and dividing by μ, we have

$$\frac{1}{\mu}\boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K} + \mu \boldsymbol{G}\boldsymbol{G}^\top)\boldsymbol{Z}^\top \boldsymbol{u} = \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}. \qquad (29)$$

With the lexicographical ordering mentioned in Section 3.2, $\boldsymbol{G}$ has the following useful property:

$$\boldsymbol{G}\boldsymbol{G}^\top = (N+1)\boldsymbol{I}_N - \boldsymbol{1}_N \boldsymbol{1}_N^\top,$$

where $\boldsymbol{I}_N$ denotes the $N \times N$ identity matrix, and $\boldsymbol{1}_N$ denotes an N-dimensional constant vector such that $\boldsymbol{1}_N = [1, \ldots, 1]^\top$. Then, the former term in Eq. (28) can be rewritten as follows:

$$\begin{aligned}&\frac{1}{\mu}\boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K} + \mu \boldsymbol{G}\boldsymbol{G}^\top)\boldsymbol{Z}^\top \\ =\;& \frac{1}{\mu}\boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K} + \mu(N+1)\boldsymbol{I}_N - \mu \boldsymbol{1}_N \boldsymbol{1}_N^\top)\boldsymbol{Z}^\top \\ =\;& \frac{1}{\mu}\boldsymbol{Z}\big((1+\mu)\boldsymbol{I}_N - \boldsymbol{K}\big)\boldsymbol{Z}^\top + N\boldsymbol{Z}\boldsymbol{Z}^\top - \boldsymbol{Z}\boldsymbol{1}_N \boldsymbol{1}_N^\top \boldsymbol{Z}^\top \\ =\;& \frac{\alpha}{1-\alpha}\boldsymbol{Z}\left(\frac{1}{\alpha}\boldsymbol{I}_N - \boldsymbol{K}\right)\boldsymbol{Z}^\top + N\boldsymbol{Z}\boldsymbol{Z}^\top - (\boldsymbol{Z}\boldsymbol{1}_N)(\boldsymbol{Z}\boldsymbol{1}_N)^\top \\ =\;& \frac{1}{1-\alpha}\boldsymbol{Z}(\boldsymbol{I}_N - \alpha \boldsymbol{K})\boldsymbol{Z}^\top + N\boldsymbol{Z}\boldsymbol{Z}^\top - (\boldsymbol{Z}\boldsymbol{1}_N)(\boldsymbol{Z}\boldsymbol{1}_N)^\top,\end{aligned}$$

where $\alpha = \frac{1}{\mu+1}$ (i.e. $\mu = \frac{1-\alpha}{\alpha}$). From Eq. (5) and (7), we have

$$\begin{aligned}&\boldsymbol{Z}\left(\boldsymbol{I}_N - \alpha \boldsymbol{K}\right)\boldsymbol{Z}^\top \\ =\;& (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-\frac{1}{2}} \boldsymbol{V}_R^\top \boldsymbol{V}_N (\boldsymbol{I}_N - \alpha \boldsymbol{\Lambda}_N) \boldsymbol{V}_N^\top \boldsymbol{V}_R (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-\frac{1}{2}} \\ =\;& (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-\frac{1}{2}} (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)(\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda}_R)^{-\frac{1}{2}} \\ =\;& \boldsymbol{I}_R.\end{aligned}$$

Note that $\boldsymbol{V}_R^\top \boldsymbol{V}_N = [\boldsymbol{I}_R \;\; \boldsymbol{0}_{N-R}]$ since the column vectors in $\boldsymbol{V}_N$ are orthonormal.

Denoting by $\boldsymbol{W} = \boldsymbol{Z}\boldsymbol{Z}^\top$ and $\boldsymbol{w} = \boldsymbol{Z}\boldsymbol{1}_N$. From Eq. (7), we have

$$\boldsymbol{Z}\boldsymbol{Z}^\top = (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda})^{-\frac{1}{2}} \boldsymbol{V}_R^\top \boldsymbol{V}_R (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda})^{-\frac{1}{2}} = (\boldsymbol{I}_R - \alpha \boldsymbol{\Lambda})^{-1},$$

where both $\boldsymbol{I}_R$ and $\boldsymbol{\Lambda}$ are diagonal, so $\boldsymbol{W}$ is diagonal such that $[\boldsymbol{W}]_{ii} = 1/(1 - \alpha\lambda_i)$.

Consequently, Eq. (29) can be rewritten as

$$\left(\frac{1}{1-\alpha}\boldsymbol{I}_R + N\boldsymbol{W} - \boldsymbol{w}\boldsymbol{w}^\top\right)\boldsymbol{u} = \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y},$$

hence the optimal weight vector $\boldsymbol{u}$ has the form

$$\boldsymbol{u} = \left(\frac{1}{1-\alpha}\boldsymbol{I}_R + N\boldsymbol{W} - \boldsymbol{w}\boldsymbol{w}^\top\right)^{-1} \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}. \qquad (30)$$

B. WITH/WITHOUT REGULARIZATION

Without the smoothness term, the optimal $\boldsymbol{u}$ has the form

$$\begin{aligned}\boldsymbol{u} &= \Big((N+1)\boldsymbol{Z}\boldsymbol{Z}^\top - \boldsymbol{w}\boldsymbol{w}^\top\Big)^{-1} \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y} \\ &\approx (N^2 \mathbf{Cov}_{\mathcal{M}}(\boldsymbol{x}))^{-1} \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}, \qquad (31)\end{aligned}$$

where $\boldsymbol{w} = \sum_{i=1}^N \boldsymbol{z}_i$, and $\mathbf{Cov}_{\mathcal{M}}(\boldsymbol{x})$ denotes the covariance matrix of the web pages in the latent space $\mathcal{M} \in \mathbb{R}^R$.

With the smoothness term, the optimal $\boldsymbol{u}$ is as in Eq. (30) in Appendix A, which can be rewritten as

$$\boldsymbol{u} = \left(\frac{1}{1-\alpha}\boldsymbol{I} + N^2 \mathbf{Cov}_{\mathcal{M}}(\boldsymbol{x})\right)^{-1} \boldsymbol{Z}\boldsymbol{G}\boldsymbol{y}. \qquad (32)$$

By comparing Eq. (31) and (32), we conclude that the smoothness term $\boldsymbol{u}^\top \boldsymbol{Z}(\boldsymbol{I}_N - \boldsymbol{K})\boldsymbol{Z}^\top \boldsymbol{u}$ plays a role as the *L_2-norm (ridge) regularizer* [9] in the latent space.