Semi-supervised Ranking for Document Retrieval

Kevin Duh, Katrin Kirchhoff

University of Washington

Abstract

Ranking functions are an important component of information retrieval systems. Recently there has been a surge of research in the field of "learning to rank", which aims at using labeled training data and machine learning algorithms to construct reliable ranking functions. Machine learning methods such as neural networks, support vector machines, and least squares have been successfully applied to ranking problems, and some are already being deployed in commercial search engines.

Despite these successes, most algorithms to date construct ranking functions in a supervised learning setting, which assume that relevance labels are provided by human annotators prior to training the ranking function. Such methods may perform poorly when human relevance judgments are not available for a wide range of queries. In this paper, we examine whether additional unlabeled data, which is easy to obtain, can be used to improve supervised algorithms. In particular, we investigate the transductive setting, where the unlabeled data is equivalent to the test data.

We propose a simple yet flexible transductive meta-algorithm: the key idea is to adapt the training procedure to each test list after observing the documents that need to be ranked. We investigate two instantiations of this general framework: The Feature Generation approach is based on discovering more salient features from the unlabeled test data and training a ranker on this test-dependent feature-set. The Importance Weighting approach is based on ideas in the domain adaptation literature, and works by re-weighting the training data to match the statistics of each test list. We demonstrate that both approaches improve over supervised algorithms on the TREC and OHSUMED tasks from the LETOR dataset.

Key words: Information retrieval, Ranking algorithms, Semi-supervised Learning

1. Introduction

Ranking functions are integral to information retrieval systems: given a query that represents a user's information need, the ranking function orders the retrieved documents such that the most relevant ones appear first. The performance of such ranking functions often significantly influences the performance of the information retrieval system; thus the area of ranking has been at the forefront of information retrieval research.

A large class of ranking functions are human-engineered "metrics" that compute a score for each query-document pair. Documents are then ranked by their respective scores. For example, using the vector space model, one can measure the score of a query-document pair as the distance between suitably-defined query and document vectors. Particular methods include TF-IDF and BM25, which have been shown to work well in many evaluations.

Recently, the machine learning paradigm has emerged as a promising approach to solving ranking problems. In this "Learning to Rank" setup, one first prepares a training set comprising queries and documents labeled by their relevance. These query-document pairs are represented by feature vectors, which could include a variety of metrics (e.g. BM25). Then a machine learning algorithm learns the optimal combination of these features for predicting relevance rankings. The advantage of the machine learning approach is that it can automatically tune the ranking function for the given dataset. However, this also leads to a disadvantage: the ranking function can only be as good as the quality and the size of labeled training data.

The motivation of our work is to address the fundamental limitation of finite labeled training data. Ideally, the training data should cover a broad set of queries, but in practice, only the most important or prevalent queries are labeled with human judgments. Although approaches such as pseudo-relevance feedback and learning from click-through data may augment the finite labels, this information is inherently more noisy. Rather than examining whether potentially noisy labels can be usefully leveraged, as done in many previous work ([46, 47]), we focus on the orthogonal problem of whether data *without* labels can be used at all.

Our goal is to examine whether additional unlabeled data can be used to improve the ranking performance. Importantly, we consider the case when the test data is the unlabeled data. This is called transductive learning, whose core idea is that tailoring a learning algorithm to a specific test set should outperform a learner that has been trained to perform well on any

given dataset. Both transductive learning and the more general inductive semi-supervised learning (which will be explained further in Section 2) have become active research areas in the machine learning community, but most work has focused on classification problems. There is little work addressing semi-supervised or transductive learning for ranking, which is the aim of this paper.

We focus on the "dynamic ranking" problem, where one predicts ranking of documents given a particular query. Dynamic ranking is in contrast to static ranking, which predicts document importance regardless of query. Further, our algorithms work on the sub-problem of "subset ranking", which assumes that an initial retrieval engine has generated a finite subset of documents, and the dynamic ranker is only required to sort or re-rank this finite list. For example, the subset for each query may be defined as the set of documents that have a boolean term match with the query.

To be precise, let q = query, $\mathbf{d}$ = list of retrieved documents, and $\mathbf{y}$ = list of relevance judgments. Let $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}$ be the training set consisting of L tuples of query-document-labels. Documents within the set $\mathbf{d}_l$ will be indexed by superscripts, i.e. $d_l^{(j)}$ where $j = 1..N_l$ (N_l is the number of documents retrieved for query q_l). The traditional task of "supervised learning" is to learn a ranking function using S; the ranker is then evaluated on a previously unseen and unlabeled test set $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$, where U is the number of test queries. In transductive learning, both S and E are available when building the ranking function, which is also then evaluated on E. This has the potential to outperform supervised learning since (1) it has more data, and (2) it can adapt to the test set. A pictorial representation of our problem is shown is Figure 1.

The organization of the paper is as follows: we first discuss related work (Section 2). The proposed transductive meta-algorithm framework is presented in Section 3. Two particular algorithmic instantiations, Feature Generation and Importance Weighting, are detailed in Section 4 and Section 5, respectively. The experimental setup and main results are presented in Sections 6, 7. Finally, Section 8 discusses additional experiments and further detailed analyses of the proposed method.

2. Related Work

We first discuss prior work in *supervised ranking* and *semi-supervised/transductive classification* before focusing on the main topic, *semi-supervised/transductive*

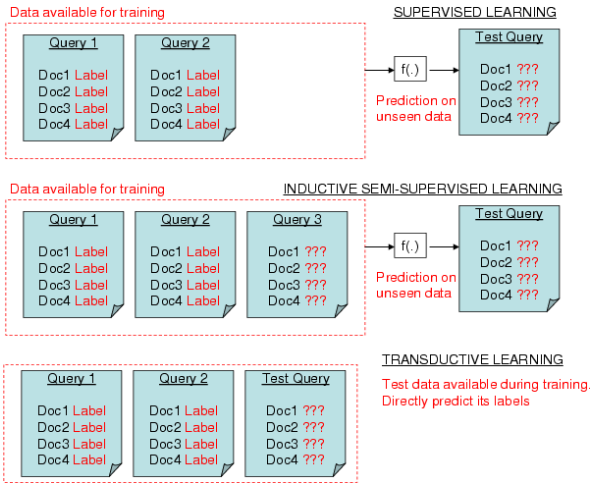


Figure 1: Supervised learning, inductive semi-supervised learning, and transductive learning: here we focus on the transductive setting, where test query is observed during training.

ranking and *domain adaptation*.

2.1. Supervised Learning for Ranking

A variety of approaches have been explored for the ranking problem in the supervised learning setting. Most algorithms learn a function f to predict a score for each query-document pair, then rank the documents according to this score. Methods differ by the objective used to obtain such a scoring function. These methods can roughly be taxonomized as regression-based, pairwise preference, and list-based methods.

In regression-based methods [24, 26], each document in the set has a target score value, and f is estimated by regression techniques to directly predict this value. The loss is measured by, for example, the residual between predicted and target scores. The idea of pairwise preference methods is to learn f such that $f(d_l^{(j)}) > f(d_l^{(i)})$ if $d_l^{(i)} \rhd d_l^{(j)}$, where $\rhd$ indicates "ranked higher than." For N_l documents, there can be $(N_l)(N_l - 1)/2$ such pairwise preferences. Popular pairwise methods include RankBoost [30] (to be explained in detail in later sections), RankSVM [37, 44], and RankNet [18].

The main advantage of regression and pairwise methods is that existing techniques in regression and classification can be applied to ranking. The main disadvantage is the optimization of an objective function that may not match popular information retrieval evaluation metrics (which generally consider the ordering of documents). Much recent work has therefore focused

on optimizing an objective that is closer to the true loss function (e.g. [17, 70, 55, 60]). Some of these methods are called list-based methods [78], because they operate with lists as the the fundamental unit and attempt to optimize objectives defined on lists, such as evaluation measures like NDCG/MAP [81, 80] or likelihoods of probabilistic permutation models [19].

2.2. Semi-supervised/Transductive Learning for Classification

The goal of semi-supervised learning and transductive learning is to combine unlabeled and labeled data such that the resulting learned function is superior than that trained on each individual dataset alone. The difference between (inductive) semi-supervised learning and transductive learning lies in the definition of the test data. In transductive learning, the test data is equivalent to the unlabeled data presented to the learning algorithm, meaning that the algorithm knows which points it should do well on to achieve low test error. In inductive semi-supervised learning, the unlabeled data used for training may be distinct from the test data. This means that the learned function must be defined over the entire space. Vapnik [71] has argued that transductive learning is inherently an easier problem and more gains can be achieved from the unlabeled test data than general unlabeled data.

Regardless of transductive vs. inductive differences, semi-supervised classification is a broad field with a variety of techniques (see [84] for a concise and updated survey). Different technique makes different assumptions on how unlabeled data can help learning. Some important classes of algorithms include:

Bootstrapping: Assume that the predicted labels of unlabeled data can be used for learning. Methods such as self-training [27, 1], co-training [15], and mixture models with EM [56, 20] fall into this class. The challenges with these methods are to ensure that incorrectly classified samples do not corrupt the training data and that baseline classifiers need to have sufficiently high accuracy.

Low Density Separation (Cluster Assumption): Assume that the classification boundary exists in low density regions, and that unlabeled data can help identify those regions. For example, transductive support vector machines [9, 68, 32] achieve this by forcing a large distance between unlabeled samples and the decision boundary. Other techniques employing this assumption include null category noise model [50], information regularization [25], entropy minimization [35], and MarginBoost [10].

Graph-based Methods: Assume that samples similar to each other have the same label, and samples indirectly linked by a chain of close samples also have the same label. A graph defined over both labeled and unlabeled data captures this global and local closeness information. Prominent examples include: Mincut [13, 14], Spectral Graph Transducer [45], Discrete Markov Random Fields (MRF) [76], and its continuous relaxation: Gaussian Random Fields and Harmonic Functions [77], Manifold Regularization [6, 7], and Graph Kernels [65, 49, 40, 3]. The assumption used in graph-based method can also be called a "manifold assumption" since they all assume that data lie in some manifold defined by the graph, and that the decision function varies slowly over this manifold.

Change of Representation: Assume that a better feature representation (e.g. more parsimonious or expressive) for learning exists and that unlabeled data can help discover this representation. Our Feature Generation approach is based on the Change of Representation assumption, which essentially involve a two-step procedure:

1. Learn a better feature/kernel representation using large unlabeled data
2. Apply supervised learning on the new feature/kernel representation of labeled data

Many possibilities exist for learning better feature representations from unlabeled data. One approach is to cluster the samples and use the cluster identities as new features [51]. Alternatively, one may learn dependencies between the original features and collapse them into more parsimonious latent variables. [4, 5, 12] use multiple-task learning to find the dependent features; [57] learns the latent variables via principal components analysis or independent components analysis. Rather than learning features, one can also learn kernels from unlabeled data, as in Fisher kernels [41, 38, 34] and cluster kernels [21].

2.3. Semi-supervised Learning applied to Ranking

There are generally two interpretations of "learning to rank with partially-labeled data." In the scenario we consider here, the document lists in our dataset are either fully labeled or not labeled at all. The second scenario arises when a document list $\mathbf{d}$ is only partially-labeled, i.e. some documents in $\mathbf{d}$ have relevance judgments, while other documents in the *same* list $\mathbf{d}$ do not. This second problem can arise when, e.g. (a) the document list retrieved by one query is too long and the annotator can only label a few

documents, (b) one uses a implicit feedback mechanism [44] to generate labels and some documents simply cannot acquire labels with high confidence. Currently there is no precise terminology to differentiate the two problems. Here we will call Problem One *"Semi-supervised Rank Learning"* and Problem Two *"Learning to Rank with Missing Labels"*. See Figure 2 for a pictorial comparison.

Several methods have been proposed for the *Missing Labels* problem, e.g. [83, 74, 36, 73]: the main idea there is to build a manifold/graph over documents and propagate the rank labels to unlabeled documents. One can use the propagated labels as the final values for ranking [83] (transductive), or one can train a ranking function using these values as true labels [36, 73] (inductive). One important point about these label propagation methods is that they do not explicitly model the relationship that document $d^{(j)}$ is ranked above, say, $d^{(k)}$. Instead it simply assumes that the label value for $d^{(j)}$ is higher than that of $d^{(k)}$, and that this information will be preserved during propagation.

An alternative approach that explicitly includes pairwise ranking accuracy in the objective is proposed in [2]. It also builds a graph over the unlabeled documents, which acts as a regularizer to ensure that the predicted values are similar for closely-connected documents. [23] also proposes a graph-based regularization term, but in contrast to [2], it defines the graph nodes not as documents, but as *document pairs*. Just as the pairwise formulation allows one to extend Boosting to RankBoost, this formulation allows one to adopt any graph-based semi-supervised classification technique to ranking. However, generating all possible pairs of documents in a large unlabeled dataset quickly leads to intractable graphs.

Most prior work consist of graph-based approaches for the *Missing Labels* problem. However, they may be extended to address the *Semi-supervised Rank Learning* problem if one defines the graph across both $\mathbf{d}_l$ and $\mathbf{d}_u$. For instance, [73] investigates label propagation across queries, but concluded that it is computationally prohibitive. Beyond the computational issue, however, how to construct a graph across different queries (whose features may be at different scales and not directly comparable) is an open research question.

To the best of our knowledge, [69] is the only work that tractably addresses the *Semi-supervised Rank Learning* problem. First, it uses a supervised ranker to label the documents in an unlabeled document list; next, it takes the most confident labels as seeds for label propagaton. A new supervised ranker is then trained to maximize accuracy on the labeled set while

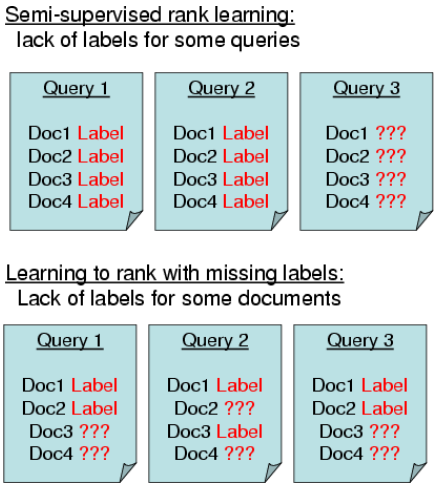


Figure 2: Two partially-labeled data problems in ranking. We focus here on semi-supervised rank learning, where labels are entirely lacking for some queries. A different problem is that of "missing labels", where not all documents retrieved by a query are labeled. Note that these two problems are not mutually-exclusive.

minimizing ranking difference to label propagation results. Thus this is a bootstrapping approach that relies on the initial ranker producing relatively accurate seeds. Our previous work [28] proposed another method using the change of Representation assumption; this paper is an extension of that work.

2.4. Relations to Domain Adaptation

Domain adaptation (c.f. [43] for a survey) is a field of machine learning that focuses on the problem of training and testing under different distributions. Our interest in domain adaptation stems from the fact that transductive learning can be considered an extreme form of domain adaptation, i.e. where one adapts to the given test set. Recent work such as [8, 58] applied the Change of Representation idea from semi-supervised learning to domain adaptation. Our work in Section 5 goes in the opposite direction, applying domain adaptation techniques to semi-supervised learning. In particular, our Importance Weighting approach treats each test list as a new domain and adapts the training procedure towards it.

Generally speaking, domain adaptation can be divided into supervised and unsupervised domain adaptation: for supervised adaptation, small amounts of labeled data are available for the test domain and the goal is to leverage the additional large amount of different but labeled data. In unsupervised

adaptation, no labels are available for the test domain. We will discuss only unsupervised adaptation since the use of unlabeled data relates more closely to the semi-supervised/transductive scenario.

The most common approach in domain adaptation is importance weighting [64], which involves re-weighting the training samples such that samples more representative of the test domain are emphasized during training. This approach is based on the assumption of "covariate shift", i.e. the sample distribution differs between train and test, but the functional relationships between input and output remain unchanged. To illustrate, consider a *classification* problem with labeled training set $\{(x_l, z_l)\}_{l=1..L}$ and unlabeled test set $\{(x_u)\}_{u=1..U}$.[1] Let $p_{train}(x)$ and $p_{test}(x)$ be the true training and test distributions, which are assumed to be significantly different.

It has been shown [64] that training on a dataset where each training sample $\{(x_l)\}_{l=1..L}$ is weighted by the ratio $w(x_l) = \frac{p_{test}(x_l)}{p_{train}(x_l)}$ corrects for covariance shift. In practice, computing the density estimates $\hat{p}_{test}(x)$ (from $\{(x_l)\}_{l=1..L}$) and $\hat{p}_{test}(x)$ (from $\{(x_u)\}_{u=1..U}$, is undesirable in high dimensions, so much recent work has focused on directly computing the importance weights $w(x_l)$ (without computing $\hat{p}_{test}(\cdot)$ and $\hat{p}_{test}(\cdot)$) [11, 39, 67]. A supervised algorithm applied to this weighted dataset would therefore focus on correctly classifying training samples close to the test distribution (i.e. high $w(x_l)$), while ignore samples far from it (low $w(x_l)$).

We are aware of only a few recent works addressing the domain adaptation problem in ranking [75, 22]. However, their methods are under the supervised adaptation framework, and therefore are not directly applicable to the transductive problem we are interested in here.

3. A General Transductive Meta-Algorithm for Ranking

We now introduce our general framework for transductive ranking:

> Suppose we observe a particular test query q_u (let $u = 1$) and the corresponding list of $N_{u=1}$ retrieved documents that need to

[1] Regarding notation: In the ranking context, we use $\mathbf{d}$ and $\mathbf{y}$ to refer to lists of documents and labels; in the classification context, we use x and z to refer to input feature vectors and class labels. The subscript l in x_l and $\mathbf{d}_l$ is overloaded to index both labeled features vectors and labeled document lists, respectively. Similarly, the subscript u in x_u and $\mathbf{d}_u$ is overloaded to index for unlabeled feature vectors and unlabeled document lists.

be ranked (i.e. $\mathbf{d}_{u=1} \equiv \{d_{u=1}^{(j)}\}, j = 1..N_{u=1}$). Each document in this list is a k-dimensional feature vector comprising BM25, TF-IDF, etc. What information can we exploit from this $k \times N_{u=1}$ set of numbers in order to improve our ranking for this query?

It is important to note that we set up the problem so that only one test query/list is in focus at a time, even though there may be U test queries in total. The rationale is that different test queries are essentially independent problems from the perspective of the ranking function, and that it is likely easier to extract information that will be helpful for one list, rather than many lists. [2]

Algorithm 1 presents our general framework (meta-algorithm) for transductive ranking in pseudo-code. The methods in Sections 4 and 5 are two particular instantiations. For each test list u, first we obtain some information from the raw document feature vectors $\mathbf{d}_u$ (line 2). Then, we use this additional information, together with the original labeled training data, to obtain a ranking function (line 3). After the ranking function $F_u(\cdot)$ re-sorts the test list u, it can be discarded (line 4). The loop (lines 1-5) need not be a sequential operation, but can be computed in parallel since the ranking functions are trained independently.

Algorithm 1 Transductive Meta-Algorithm

Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Test set $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$
Output: Predicted rankings for test: $\{\mathbf{y}_u\}_{u=1..U}$

1: **for** $u = 1$ to U **do**
2: Observe the test documents $\mathbf{d}_u = \{d_u^{(j)}\}_{j=1..J_u}$ for query q_u.
3: Train a ranking function $F_u(\cdot)$ using the Train Set S and the additional observed information.
4: Predict the test ranking: $\mathbf{y}_u = F_u(\mathbf{d}_u)$
5: **end for**

Our proposed meta-algorithm can be contrasted with the established method of pseudo-relevance feedback, which can be seen as a kind of trans-

[2] The motivation here has some analogies to query classification (c.f. [31]), which believes that different classes of queries are best served by different ranking functions. We push this to the extreme by making *every* query be served by its own ranking function.

ductive ranking technique. Pseudo-relevance feedback (c.f. [79], [53] chapter 9) uses words in the initial top retrieved documents to generate a new query, which is then used to retrieve a new list of documents. Although this new query may contain some noise, as in the semi-supervised method of self-training, it may retrieve more relevant documents that have little match with the original query. Note that pseudo-relevance feedback occurs at query-time and the result is query-specific. In this respect it is similar to our transductive meta-algorithm. The three main differences are:

1. Pseudo-relevance feedback usually uses textual information from the test list, whereas our transductive meta-algorithm works purely from the document feature vectors
2. Pseudo-relevance feedback usually creates a new query, whereas our meta-algorithm creates new ranking function.
3. Pseudo-relevance feedback depends on a self-training/bootstrapping assumption, whereas our meta-algorithm leaves the assumption unspecified.

Finally, we can also compare our meta-algorithm to *local learning*. Local learning differs from traditional supervised learning in that it does not use the entire training set, but rather selects a subset of samples close to each test sample [16]. The intuition is that fitting a smooth function over a small partition of the feature space is easier than fitting a function over the entire space. Our meta-algorithm could be termed local, due to properties such as training at query-time and fitting test-specific functions; however, our meta-algorithm is more general in that it is not restricted to techniques that subsample the training data. In the information retrieval literature, local learning by k-nearest neighbors [33] and by query-time association rules [72] have achieved promising results.

4. The Feature Generation Approach to Transductive Ranking

The Feature Generation (FG) Approach is an instantiation of the general transductive meta-algorithm in Section 3. It employs the semi-supervised "Change of Representation" assumption and requires two components:

- First, an unsupervised method (e.g. principal components analysis) is applied to discover salient features for the test list.

- Second, a supervised method for learning to rank (e.g. RankBoost) is applied to labeled training data with this new representation, which ideally is more pertinent to the test list in question.

Algorithm 2 Feature Generation (FG) Approach to Transductive Ranking

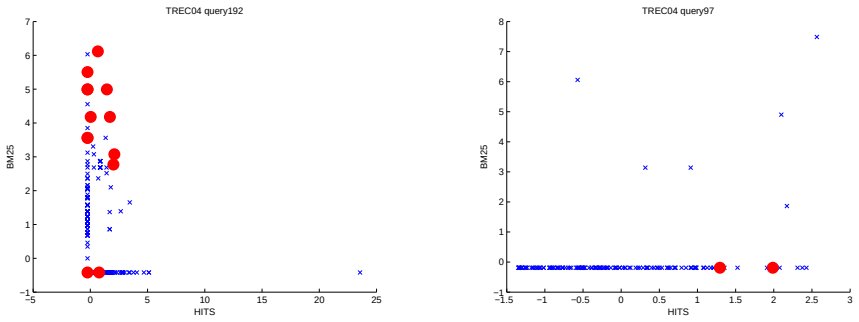
Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Test set $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$
Input: DISCOVER(), unsupervised algorithm for discovering salient patterns
Input: LEARN(), a supervised ranking algorithm
Output: Predicted rankings for test: $\{\mathbf{y}_u\}_{u=1..U}$

1: **for** $u = 1$ to U **do**
2: P_u = DISCOVER($\mathbf{d}_u$) # find transform on test data
3: $\hat{\mathbf{d}}_u = P_u \cdot \mathbf{d}_u$ # project test data along P_u
4: **for** $l = 1$ to L **do**
5: $\hat{\mathbf{d}}_l = P_u \cdot \mathbf{d}_l$ # project train data along P_u
6: **end for**
7: $F_u(\cdot)$ = LEARN($\{(q_l, \hat{\mathbf{d}}_l, \mathbf{y}_l)\}_{l=1..L}$)
8: $\mathbf{y}_u = F_u(\hat{\mathbf{d}}_u)$ # predict test ranking
9: **end for**

Algorithm 2 shows the pseudocode for this Feature Generation approach[3]. `DISCOVER()` is a generic unsupervised method that is applied to each test list $\mathbf{d}_u$ separately (line 2). `LEARN()` is a generic supervised method for learning rank functions. Since the feature-based representations of the training documents ($\{\mathbf{d}_l\}_{l=1..L}$) are enriched with additional test-specific features (line 5), we learn a different ranking function $F_u(\cdot)$ for each test query (line 7).

The usefulness of test-specific features and test-specific ranking functions is illustrated in Figures 3(a) and 3(b). These are plots of documents from two TREC'04 queries. The x-axis shows the (normalized) HITS Hub score of a document, while the y-axis shows the (normalized) BM25 score of the extracted title (both are important features for learning). Irrelevant documents are plotted as small crosses whereas relevant documents are large dots.

[3]Here, line 2 corresponds to line 2 in the Algorithm 1, lines 3-7 correspond to line 3 in Algorithm 1.



(a) (b)

Figure 3: Plots of documents for 2 different queries in TREC'04 (y-axis = BM25, x-axis = HITS score). Relevant documents are dots, irrelevant ones are crosses. Note that (a) varies on the y-axis whereas (b) varies on the x-axis, implying that query-specific rankers would be beneficial.

For the first query (Fig. 3(a)), we see that the data varies mostly along the y-axis (BM25); for the second query (Fig 3(b)), the variation is on the x-axis (HITS). These two document lists would be better ranked by two different rankers, e.g. one which ranks documents with BM25 > 2.5 as relevant, and the second which ranks documents with HITS > 1.25 as relevant. A single ranker would find it difficult to simultaneously rank both lists with high accuracy.

In this paper, we use kernel principal components analysis (Kernel PCA) [62] as the unsupervised method and RankBoost [30] as the supervised ranker. Kernel PCA is advantageous in its flexibility in generating many different types of features by the use of different kernels. This is a good combination with RankBoost, which has been shown to be relatively robust to variations in tuning parameters and feature sets. For completeness, we present brief reviews of these two methods in the following subsections.

4.1. Unsupervised feature extraction: Kernel PCA

Principal components analysis (PCA) is a classical technique for extracting patterns and performing dimensionality reduction from unlabeled data. It computes a linear combination of features, which forms the direction that captures the largest variance in the data set. This direction is called the principal axis, and projection of a data point on it is called the principal component. The magnitude of the principal component values indicates how close a data point is to the main directions of variation.

Kernel PCA [62] is a powerful extension to PCA that computes arbitrary *non-linear* combinations of features. As such, it is able to discover patterns arising from higher-order correlations between features. We can imagine Kernel PCA as a procedure that first maps each data point into a (possibly) non-linear and higher-dimensional space, then performs PCA in that space. More precisely, let $\mathbf{d}$ be a list of m documents and $d^{(j)}$ be the original feature vector of document j.[4] Then Kernel PCA can be seen as the following procedure:

1. Map each document $d^{(j)}$ to a new space $d^{(j)} \mapsto \Phi(d^{(j)})$, where $\Phi(\cdot)$ is the (non-linear/high-dimension) mapping.
2. Compute covariance matrix in this new space: $\mathbf{C} = \frac{1}{m}\sum_{j=1}^{m} \Phi(d^{(j)})\Phi(d^{(j')})^T$. ($T$ = transpose. Φ should be centered at zero mean–if not, this can be achieved by some simple operations in kernel space [63])
3. Solve the eigen-problem: $\lambda\mathbf{v} = \mathbf{C}\mathbf{v}$.
4. The eigenvectors $\mathbf{v}$ with the largest eigenvalues λ form a projection matrix P. Datapoints can now be projected to the principal axes of the non-linear space defined by $\Phi(\cdot)$.

In practice, Kernel PCA uses the dual formulation to avoid solving the above eigen-problem in high dimensional space (this is known as the kernel trick). See [62] for the derivation; here we only present the steps needed for this paper:

1. Define a kernel function $k(\cdot,\cdot) : (d^{(j)}, d^{(j')}) \rightarrow \mathbb{R}$ which maps two document vectors to a real number indicating the similarity between the two documents.
2. There exist kernels of the form $k(d^{(j)}, d^{(j')}) = \langle\Phi(d^{(j)}), \Phi(d^{(j')})\rangle$, (i.e. dot product of the document mappings in high-dimensional space) such that the mapping does not need to be computed explicitly to get the kernel value.
3. Let the $m \times m$ matrix K be the kernel values of all pairs of documents in the list. i.e. $K_{jj'} = k(d^{(j)}, d^{(j')})\ \forall j, j' \in \{1, 2, \ldots, m\}$. This kernel matrix can be centered to ensure that the features Φ are zero-mean.

[4] In the context of Kernel PCA, we drop the subscript in $\mathbf{d}_u$ to avoid clutter. $\mathbf{d}_u$ or $\mathbf{d}$ is a document list; $d^{(j)}$ is one document vector within the list.

4. Kernel PCA reduces to solving the eigen-problem $m\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}$. We pick only the $\boldsymbol{\alpha}$ with the largest eigenvalues.
5. For a new document $d^{(n)}$, its principal component is computed as $\sum_{j=1}^{m} \alpha_j k(d^{(j)}, d^{(n)})$.

The kernel function defines the type of non-linear patterns to be extracted. In this work, we use the following kernels:

- **Polynomial**: Computes dot product of all monomials of order p, $k(d^{(j)}, d^{(j')}) = \langle d^{(j)}, d^{(j')} \rangle^p$.
- **Gaussian / Radial basis function**: $k(d^{(j)}, d^{(j')}) = \exp\left(-\frac{||d^{(j)} - d^{(j')}||}{2\sigma}\right)$. This is an isotropic kernel, with bandwidth σ adjusting for smoothness.
- **Diffusion kernel** [49]: This is suitable for graph data. We generate a k-nearest neighbor graph with documents as nodes and edges defined by the inverse Euclidean distance $1/||d^{(j)} - d^{(j')}||$. $k(d^{(j)}, d^{(j')})$ is defined by running a lazy random walk from $d^{(j)}$ to $d^{(j')}$. A time-constant parameter τ adjusts how long to run the random walk (e.g. larger τ leads to a more uniform distribution). Performing Kernel PCA with diffusion kernels is equivalent to running PCA on a non-linear manifold.
- **Linear**: $k(d^{(j)}, d^{(j')}) = \langle d^{(j)}, d^{(j')} \rangle$. Equivalent to PCA.

Kernel PCA scales as $O(m^3)$, due to solving the eigen-problem on the $m \times m$ kernel matrix K. Nevertheless, extremely fast versions have been proposed; for instance, Sparse kernel feature analysis [66] is based on sparsity constraints and can extract patterns in $O(m)$.

4.2. Supervised Ranking Algorithm: RankBoost

RankBoost [30] is an extension of the boosting approach [61] for ranking. In each iteration, RankBoost searches for a weak learner that maximizes the (weighted) pairwise ranking accuracy (defined as the number of document pairs that receive the correct ranking). A weight distribution is maintained for all pairs of documents. If a document pair receives an incorrect ranking, its weight is increased, so that the next iteration's weak learner will focus on correcting the mistake.

It is common to define the weak learner as a non-linear threshold function on the features (decision stump). For example, a weak learner $h(\cdot)$ may be

$h(d^{(j)}) = 1$ if "BM25 score > 1" and $h(d^{(j)}) = 0$ otherwise. The final ranking function of RankBoost is a weighted combination of T weak learners:

$$F(d^{(j)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(j)}), \tag{1}$$

where T is the total number of iterations. θ_t is computed during the RankBoost algorithm and its magnitude indicates the relative importance of a given weak learner (feature). Finally, a ranking over a document list $\mathbf{d}$ is obtained by calculating $y^j = F(d^{(j)})$ for each document and sorting the list by the value of y^j.

Algorithm 3 RankBoost

Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Initial distribution $D(i, j)$ over (i,j)
Output: Ranking function $F(\cdot)$.

1: **for** $t = 1$ to T **do**
2: Find weak ranker $h_t(\cdot)$ on weighted data D.
3: Choose step size θ_t
4: Update weights $D(i, j) = D(i, j) \exp \theta_t(h_t(d^{(i)}) - h_t(d^{(j)}))$. Normalize.
5: **end for**
6: Output final ranking function $F(d^{(n)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(n)})$.

In theory, many algorithms can be plugged in for `DISCOVER()` and `LEARN()`. In practice, it is important to consider the interaction between feature and learning, and to ensure that `DISCOVER()` generates features that `LEARN()` is able to exploit. We believe that there are several advantages to using RankBoost with Kernel PCA in our transductive framework:

1. Inherent feature selection: RankBoost selects T features that are most conducive to good rankings. Since there are no guarantees that the Kernel PCA's directions of high variance always correspond to directions of good ranking, RankBoost's inherent feature selection reduces the need for tuning. For a `LEARN()` algorithm without inherent feature selection, we may have to tune for (a) number of Kernel PCA features, (b) relative importance of Kernel PCA features compared to original features.

2. Non-linear thresholding in weak learners $h(\cdot)$: One could define the weak learner to be simply the feature values (e.g. $h(\cdot) =$ raw BM25 score). This assumes that good ranking is directly correlated to the feature values (e.g. large BM25 implies more relevance). Kernel PCA, however, may generate features that have a non-linear relationship to ranking (e.g. large positive *and* negative deviation from the principal axes implies less relevance). Non-linear rankers can handle this possibility more robustly.
3. "Anytime" training: Boosting can be seen as gradient descent in function space [54] and each iteration improves on the training accuracy. If training time is a concern (e.g. in practical deployment of the transductive framework), then RankBoost can be stopped before reaching T iterations. The resulting ranker may be less optimized, but it should still give reasonable predictions.

5. The Importance Weighting Approach to Transductive Ranking

The Importance Weighting (IW) Approach is another instantiation of the general transductive meta-algorithm. The main idea is to treat each test list as a new domain for domain adaptation. It requires two components:

- An domain adaptation algorithm, `ADAPT()`, that generates importance weights specific to each test list.
- A supervised learning to rank algorithm, `WEIGHTED-LEARN()`, that can train on weighted data. Essentially, only a weighted subset of the training data most similar to the test list will be used in computing the ranking function.

Algorithm 4 Importance Weighting (IW) Approach to Transductive Ranking

Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Test set $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$
Input: ADAPT(), a domain adaptation algorithm
Input: WEIGHTED-LEARN(), a supervised ranking algorithm that handles weighted data
Output: Predicted rankings for test: $\{\mathbf{y}_u\}_{u=1..U}$

1: **for** $u = 1$ to U **do**
2: W = ADAPT($\mathbf{d}_u, \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}$) # find weighting over training samples such that samples close to test have high weights
3: $F_u(\cdot)$ = WEIGHTED-LEARN($W, \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}$)
4: $\mathbf{y}_u = F_u(\mathbf{d}_u)$ # predict test ranking
5: **end for**

Algorithm 4 shows the pseudo-code for the Importance Weighting (IW) approach. In our instantiation, `WEIGHTED-LEARN()` is the AdaCost version of RankBoost [29] and `ADAPT()` is the Kullback-Liebler Importance Estimation Procedure (KLIEP) [67]. KLIEP is currently a state-of-the-art method in importance weighting, its main advantages being its automatic model selection procedure and proven convergence properties. The main issue here is how to adjust the importance weighting method developed for classification to a ranking problem. The samples to which importance weights are applied depends on `WEIGHTED-LEARN()`. Since AdaCost-Rankboost is a pairwise ranking algorithm, our importance weights will be applied to samples consisting of document pairs. If `WEIGHTED-LEARN()` were a regression-based method, then we would define importance weights for each training document; for listwise methods, the importance weights would be defined on the level of each query/list.

5.1. Computing the Importance Weights

Our domain adaptation method `ADAPT()` works in the following steps:

1. Extract all pairs of documents from the training set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$ where there are rank differences (i.e. $y^j_{l=1} \neq y^k_{l=1}$). This is the same set of document pairs that would be extracted from a pairwise ranking algorithm which maximizes pairwise accuracy, such as RankBoost. Suppose there are L_{pair} such document pairs.

2. Extract all pairs of documents from the test list $\mathbf{d}_{u=1} \equiv \{d_{u=1}^{(j)}\}, j = 1..N_{u=1}$. There will be a total of $U_{pair} = N_{u=1} * (N_{u=1} - 1)$ such pairs.
3. For each train/test document pair, derive a single vector representation by taking the difference of the original document feature vectors. For instance, for the document pair $(d_l^{(j)}, d_l^{(k)})$ we derive the difference vector $x \equiv d_l^{(j)} - d_l^{(k)}$. The set of difference vectors from the training set will be $\{x_l\}_{l=1..L_{pair}}$; the set of difference vectors from the test list will be $\{x_u\}_{u=1..U_{pair}}$.[5]
4. Run the KLIEP importance weighting algorithm [67] using $\{x_u\}_{u=1..U_{pair}}$ as samples from the target domain. The method will generate weights $w(x_l)$ for each sample in $\{x_l\}_{l=1..L_{pair}}$.

For completeness, we briefly review KLIEP; for details, refer to [67]. KLIEP computes importance weights $w(x_l)$ without directly estimating the densities $\hat{p}_{test}(x)$ and $\hat{p}_{test}(x)$. The main idea is to minimize the Kullback-Liebler divergence between the test distribution $p_{test}(x)$ and the weighted training distribution $w(x) * p_{train}(x)$:

$$KL(p_{test}(x)//w(x) * p_{train}(x)) = \int p_{test}(x) \log \frac{p_{test}(x)}{w(x) * p_{train}(x)} dx \quad (2)$$

$$= \int p_{test}(x) \log \frac{p_{test}(x)}{p_{train}(x)} dx - \int p_{test}(x) \log w(x) dx \quad (3)$$

The first term does not depend on w and can be dropped in the following objective:

$$\mathcal{O}_{KLIEP} = \int p_{test}(x) \log w(x) dx \quad (4)$$

$$\approx \frac{1}{U_{pair}} \sum_{u=1}^{U_{pair}} \log w(x_u) \quad (5)$$

$$= \frac{1}{U_{pair}} \sum_{u=1}^{U_{pair}} \log \sum_{b=1}^{B} \beta_b \psi_b(x_u) \quad (6)$$

$$(7)$$

[5]Note that for notational simplicity we have again overloaded the indexes u and l to index both lists and individual vectors.

where the last line follows from parameterizing the weights w as a weighted average of basis functions: $w(x) = \sum_{b=1}^{B} \beta_b \psi_b(x)$. In this work, these bases are Gaussian kernels centered at the test samples: $\psi_b(x) == \exp\left(-\frac{||x - x_{u=b}||}{2\sigma}\right)$. The kernel bandwidth σ are set by KLIEP's automatic model selection procedure. In addition, we will need the constraints that $\beta \geq 0$ (so that the weights w are positive) and $1 = \int w(x) p_{train}(x) dx \approx \frac{1}{L_{pair}} \sum_{x=1}^{L_{pair}} \sum_{b}^{B} \beta_b \psi(x_l)$ (so that it is a proper distribution). The resulting problem can be solved by linear programming. In the end, we have a weights $w(x_l)$ for each document pair in the training data, where large values represent training document pairs that are similar to test document pairs.

5.2. AdaCost: RankBoost with Importance Weights

The weights $\{w(x_l)\}_{l=1..L_{pair}}$ are given to the AdaCost-Rankboost learning algorithm. AdaCost will ensure that training document pairs with large weights are ranked correctly during training, possibly at the expense of other document pairs with smaller importance weights.

Algorithm 5 shows the AdaCost modification. Note that the only change from traditional RankBoost (Algorithm 3 is the cost factor $c(i, j)$ in Line 4 and its incorporation into the update equation in Line 5. The cost factor $c(i, j)$ is computed such that:[6]

- If importance weight $w(d^{(i)} - d^{(j)})$ is large and prediction is incorrect (i.e. $h_t(d^{(i)} > h_t(d^{(j)})$), then $D(i, j)$ is increase much (i.e. $c(i, j)$ is large)
- If importance weight is small and prediction is incorrect, then $D(i, j)$ increases only slightly.
- If importance weight is large and prediction is correct, then $D(i, j)$ decreases only slightly.
- If importance weight is small and prediction is correct, then $D(i, j)$ decreases much.

[6]We use the formula $c(i, j) = 0.5 * \tilde{w}(d^{(i)} - d^{(j)}) + 0.5$ if the pair is correctly ranked, and $c(i, j) = -0.5 * \tilde{w}(d^{(i)} - d^{(j)}) + 0.5$ otherwise. $\tilde{w}$ is w normalized to $[0, 1]$.

Algorithm 5 RankBoost - AdaCost version

Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Initial distribution $D(i, j)$ over (i,j)
Input: Weights on each training document pair w
Output: Ranking function $F(\cdot)$.

1: **for** $t = 1$ to T **do**
2: Find weak ranker $h_t(\cdot)$ on weighted data D.
3: Choose step size θ_t
4: Compute cost factor $c(i, j)$ depending on importance weight w
5: Update weights $D(i, j) = D(i, j) \exp\left(c(i, j)\theta_t(h_t(d^{(i)}) - h_t(d^{(j)}))\right)$. Normalize.
6: **end for**
7: Output final ranking function $F(d^{(n)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(n)})$.

6. Data and Experimental Setup

We perform experiments on the LETOR dataset (version 2) [52], which contains three sets of document retrieval data: TREC'03, TREC'04, and OHSUMED. This is a re-ranking (subset ranking) problem, where an initial set of documents have been retrieved and the goal is to sort the set in an order most relevant to the query. The TREC data is a Web Track Topic Distillation task. The goal is to find webpages that are good entry points to the query topic in the `.gov` domain. The OHSUMED data consists of medical publications and the queries represent medical search needs. For TREC, documents are labeled {`relevant`,`irrelevant`}; an additional label {`partially relevant`} is provided for OHSUMED.

The LETOR dataset conveniently extracts many state-of-the-art features from documents, including BM25 [59], HITS [48], and Language Model [82]. Table 1 summarizes the data (e.g. in TREC'03, the ranker needs to sort on average 983 documents per query, with only 1 document in the set being relevant); see [52] for details.

The evaluation metrics are mean average precision (MAP) and normalized discount cumulative gain (NDCG@n) [42]. MAP is defined using precision, the percentage of relevant documents up to a given rank. For a set of R relevant documents, average precision (AP) is:

$$AP = \frac{1}{|R|} \sum_{d^{(j)} \in R} precision@rank(j)$$

Table 1: Data characteristics

	TREC'03	TREC'04	OHSUMED
#query	50	75	106
#document	49k	74k	16k
avg #document/query	983.4	988.9	152.3
#relevant documents	516	1600	4.8k
avg #relevant/query	1	0.6	28
avg #document pairs	302k	262k	369k
#feature (original)	44	44	25
#feature (transductive)	69	69	50

For example, for a set of documents with the following ranking: {relevant, irrelevant, relevant}, the precision at rank 1, 2, 3 are 1/1, 1/2, 2/3, respectively, and the AP is $\frac{1}{2}(1/1 + 2/3) = 0.8$. MAP is the mean of AP over all queries. NDCG is an alternative metric that takes into account multiple levels of judgment (not only relevant vs. irrelevant). Similar to precision, NDCG is measured at a given position:

$$NDCG(n) = Z_k \sum_{j=1}^{n} \frac{2^{r(j)} - 1}{\log(j)}$$

Here $r(j) = 0, 1, ..M$ represents the M-level numerical rating for document $d^{(j)}$ (Higher value indicates more relevance). The log is base 2 and we set $\log(1) = 1$ (not 0) in the above equation. Z_k is a normalization constant that represents the score of the best possible ranking and allows NDCG to be bounded by $[0, 1]$.

In this paper, we report results from the average of 5-fold cross-validation and judge statistical significance using the dependent t-test. We used our own implementation of Kernel PCA, RankBoost, and AdaCost-Rankboost; the KLIEP software is available from [67].

7. Main Results: Transductive vs. Supervised Learning

Our experimental setup compares three rankers. The Baseline is a supervised RankBoost, trained on the original training data. This is compared with the two proposed transductive approaches: Feature Generation (FG) and Importance Weighting (IW). Figure 4 compares the three systems

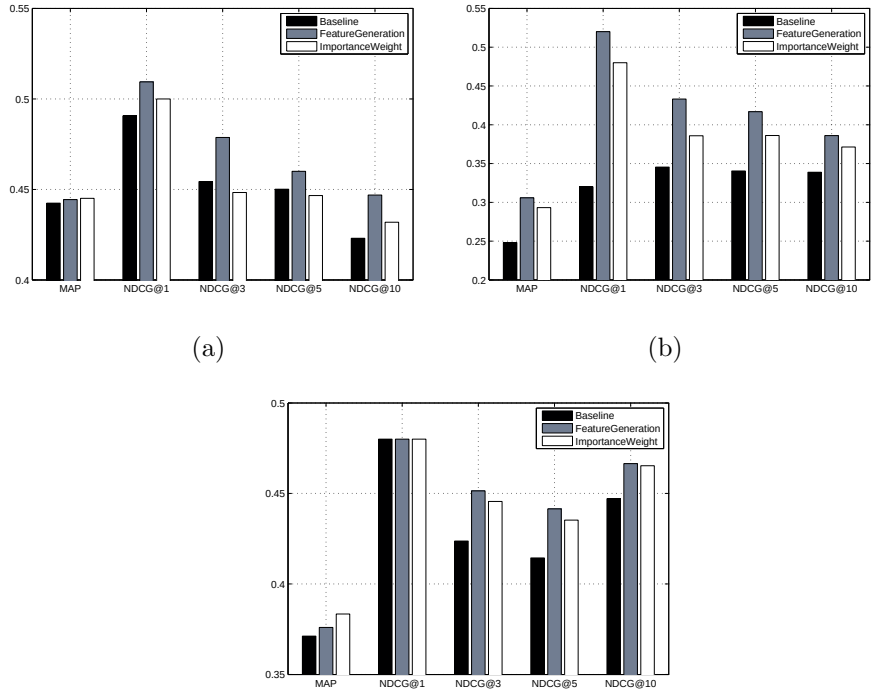


Figure 4: Main result (MAP and NDCG@n) for (a) OHSUMED, (b) TREC'03, (c) TREC'04. Both Feature Generation (FG) and Importance Weighting (IW) systems outperform the supervised Baseline in all datasets and most metrics.

on the three datasets. Table 2 shows the same results in numbers (boldfaced MAP/NDCG numbers indicate a statistically significant improvement ($p < 0.05$) over Baseline.) [7]

The main observations from Figure 4 are:

1. In general, both tranductive approaches (Feature Generation and Importance Weighting) outperform the Baseline on all datasets and most metrics. The improvements are statistically significant for some but not all metrics.
2. Between the two transductive methods, Feature Generation usually

[7] Our Feature Generation results here are slightly different to our previous work [28] due to a fix in a minor bug in our RankBoost implementation. The general results trend and conclusions of that paper are still replicated here.

Table 2: Main result (Figure 4 in table form). Both transductive methods improve over baseline. Statistically significant improvements are bold-fonted. The Combined FG+IW row is a combination of the transductive methods and will be explained in Section 8.

	MAP	N@1	N@3	N@5	N@10	N@14
TREC'03						
Baseline (supervised)	.2482	.3200	.3455	.3404	.3388	.3401
Feature Generation	**.3058**	**.5200**	.4332	**.4168**	.3861	**.3994**
Importance Weighting	**.2932**	**.4800**	.3858	**.3862**	.3713	.3755
Combined FG+IW	**.3219**	**.5250**	.4321	**.4138**	**.4023**	**.3990**
TREC'04						
Baseline (supervised)	.3712	.4800	.4237	.4144	.4471	.4686
Feature Generation	.3760	.4800	.4514	.4415	.4665	**.4910**
Importance Weighting	**.3834**	.4800	.4456	.4353	**.4653**	.4810
Combined FG+IW	**.3891**	**.4833**	.4487	**.4483**	.4554	.4873
OHSUMED						
Baseline (supervised)	.4424	.4906	.4543	.4501	.4230	.4218
Feature Generation	.4444	.5094	.4787	.4600	**.4469**	**.4377**
Importance Weighting	**.4451**	.5000	.4483	.4466	**.4319**	**.4280**
Combined FG+IW	**.4497**	.5010	**.4897**	.4765	**.4431**	**.4422**

performs better than Importance Weighting.

We also performed data ablation experiments to see how the three rankers compare for low data scenarios. For each fold, we artificially limited the training data by taking the first 40%, 60%, and 80% of the training data (i.e. TREC'03 has 30 queries for training in each fold, so we would take the first 12, 18, and 24 queries as ablated data). The results for MAP and NDCG@10 are shown in Figure 5. The Importance Weighting approach consistently improves over the Baseline and can therefore be considered a relative safe/stable algorithm. On the other hand, Feature Generation performs well for 80% and 100% but is usually worse than Baseline for 40% and 60% cases. (This corresponds to 18 and 27 training queries in TREC'04; 25 and 38 training queries in OHSUMED.) We believe this is due to the fact that Feature Generation creates more features, and is therefore more sensitive to the amount of training data.

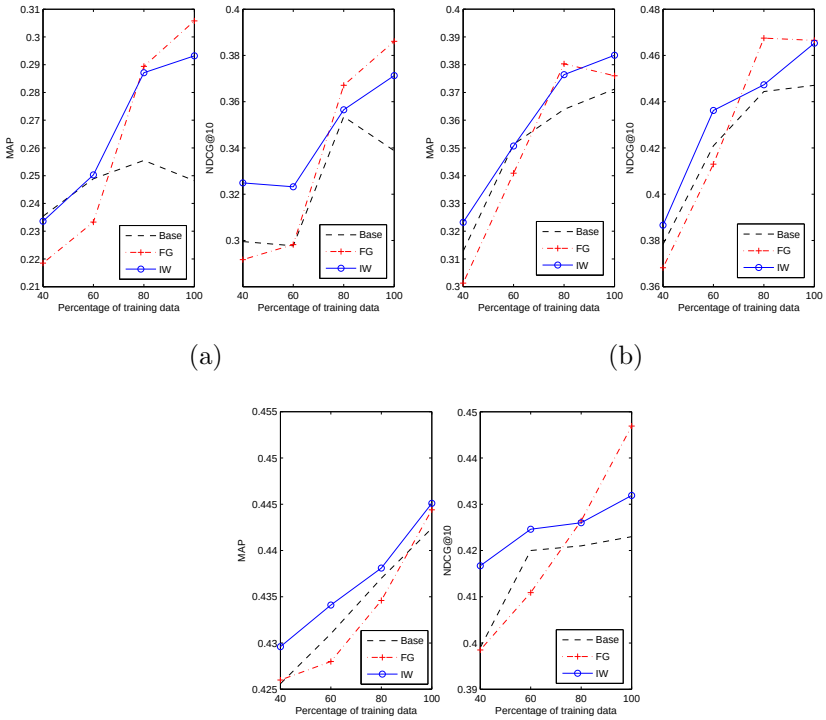


Figure 5: Data ablation results (MAP and NDCG@10) of (a) OHSUMED, (b) TREC'03, (c) TREC'04 for 40%, 60%, and 80% subsets of training data. Importance Weighting consistently improves over the Baseline. Feature Generation performs well for larger data but poorly in the 40% and 60% cases.

8. Analysis and Discussion

We perform additional experiments in order to analyze the properties of the proposed approaches.

1. Are the Feature Generation and Importance Weighting approaches mutually compatible?

The samples used in training Rankboost in Feature Generation can be weighted by importance weights. Therefore one may wonder whether the two transductive approaches are mutually compatible. To test this, we enhance the Feature Generation method by weighting each training sample (document pair) with the importance weights obtained by running KLIEP on the KPCA features. This combined method allows us to select (soft) subsets of the training data while training with test-specific features. The results in the (Combined FG+IW) rows of Table 2 show that the combined method nicely outperforms both Feature Generation and Importance Weighting in all tasks for the MAP metric. The results are more varied for NDCG: for instance, in OHSUMED, NDCG@3 and NDCG@5 showed improvments, but NDCG@1 and NDCG@10 only outperformed one of FG or IW approaches it was built upon. In general, for virtually all NDCG metrics and all task, FG+IW did not perform worse than FG or IW individually.

2. How important is it to adapt to the test query?

Does the Feature Generation approach obtain gains because Kernel PCA extracts good features per se, or particularly because the features are extracted on the *test* set (i.e. the local/transductive aspect)? In order to answer this question, a new system (`KPCA on train`) was built based on feature transformations estimated from training data alone: Kernel PCA was run on each training list (as opposed to projecting the training lists to principal directions of the test lists). The subsequent rank learner and evaluation remain identical: we train RankBoost on this data, which is the same training data as Baseline except for the additional Kernel PCA features and evaluated this new ranking function on the test set. The results (Table 3) show that `KPCA on train` is worse than Feature Generation (e.g. .2511 vs. .3058 MAP for TREC'03), implying that the transductive aspect of adapting to each test query is essential.

Table 3: Feature Generation (transductive) outperforms KPCA on train (inductive); adapting to test queries is a useful strategy.

	TREC'03	TREC'04	OHSUMED
Feature Generation	.3058	.3760	.4444
KPCA on train	.2511	.3625	.4418
Baseline	.2482	.3712	.4424

3. What are the most useful features?

For the Feature Generation system, what weak learners $h(\cdot)$ in the multiple ranking functions ($F_u(\cdot) = \sum_{t=1}^{T} \theta_t h_t(\cdot)$) achieve large $|\theta_t|$? For instance, how often are Kernel PCA features chosen compared to the original features? To analyze this, we look at the 25 FG ranking functions in TREC'04 that improve more than 20% over the Baseline. For each ranking function, we look at the top 5 features and note their type: {`original, polynomial, rbf, diffusion, linear`}. 24 of 25 functions have both original and Kernel PCA features in the top 5, indicating that Kernel PCA features are quite useful. It is even more interesting to note the distribution of top 5 feature combinations (Figure 4): no single combination is more prevalent than others. This again supports the intuition that test-specific rankers are better than a single general ranker.

Table 4: Top 5 feature combinations employed in RankBoost, by count. There is a diversity of feature combinations, indicating that different test queries require different rankers.

Chosen Features	Count
Original + Diffusion	7
Original + Polynomial	4
Original + Polynomial + Linear	4
Original + Polynomial + Diffusion	4
Original + Linear	3
Orignal + Diffusion + Linear	1
Orignal + Diffusion + Gaussian	1
Orignal only	1

4. What are the statistics for the importance weights?

We are interested in seeing how well-matched is the training set to the test query, and whether KLEIP is indeed selecting subsets of the training

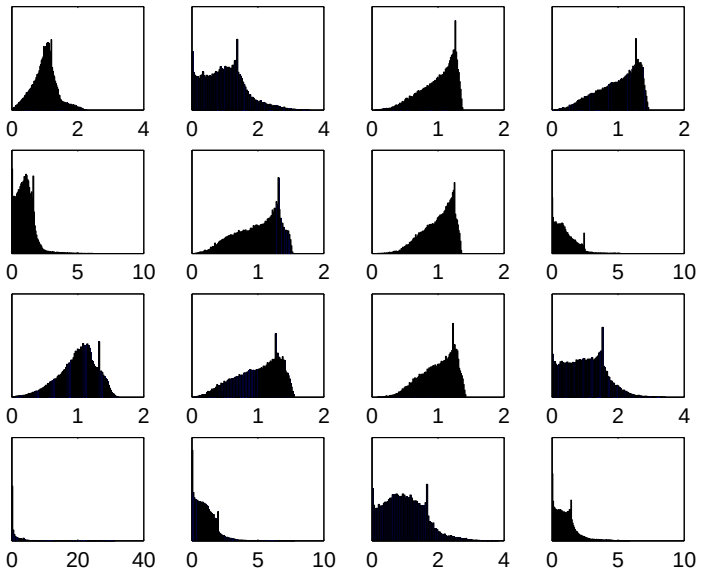


Figure 6: Importance weight histogram from some OHSUMED queries. The x-axis is the importance weight value; y-axis is the histogram count. The large variety in distribution implies that the target test statistics differ drastically.

data. Figure 6 shows histograms of importance weight values associated with a random set of test queries. Note that the histograms vary widely, i.e. for the same pairs of training documents, the corresponding importance weight varies a lot depending on the test list in question. This supports our rationale for treating each test list as a new domain adaptation problem.

Further, we compute general statistics on the importance weights, shown in Table 5. First, note that the cardinality of the importance weight distribution is similar for both OHSUMED and TREC'03: there are roughly 150k-230k training pairs for OHSUMED and 130k-270k training pairs for TREC'03 (the variation is due to different folds). Therefore we may compare weight values across datasets to draw some conclusions. The median value for importance weights is 0.91 for OHSUMED and 0.51 for TREC'03. The 25th Quantile value is also much lower for TREC'03, implying that KLIEP assigns relatively more low values to TREC'03 as compared to OHSUMED. Similarly, the higher standard deviation in TREC'03 suggests that the weight distribution for TREC'03 is more skewed and broad than OHSUMED. In

Table 5: Importance weight statistics. Median respresents the average median value of importance weights, across all test lists. Similarly, the 25th/75h quantile capture the value of the 25th and 75th portion of the weight's cumulative distribution function (CDF). Standard deviation and entropy shows how much the importance weight differs from uniform. Uniform distribution would achieve an entropy of 2.48 (entropy is calculated discretely by dividing the weight histogram into 12 bins).

	OHSUMED	TREC'03	TREC'04
Median	0.9142	0.5140	0.0461
75th Quantile	1.2808	1.3642	0.5137
25th Quantile	0.6104	0.1420	0.0011
Mode	0.9031	0.0416	0.3041
Std Deviation	0.5712	1.2951	3.2136
Entropy	1.8582	1.9653	1.2753

other words, we can say that the Importance Weighting approach ignores more training data in TREC'03 than in OHSUMED. We are not certain why this is so, but this may be due to the fact that the TREC data has more features, thus more chance of domain variety.

9. Conclusions

In this paper, we proposed a flexible transductive meta-algorithm for learning ranking functions (Algorithm 1). The main question we addressed is how information extracted from a test list can be exploited to learn better test-specific ranking functions. We presented two particular instantiations of the general transductive meta-algorithm: The Feature Generation approach (Algorithm 2) builds test-specific rankers by incorporating salient features discovered from the test list. The Importance Weighting approach (Algorithm 4) treats each test list as a new domain in domain adaptation and trains a ranking function only from similarly distributed training samples. In both cases, we have demonstrated how existing methods from semi-supervised classification, supervised ranking, and domain adaptation can be modified and combined to achieve transductive ranking.

In our experiments with the LETOR dataset, we demonstrate that both transductive ranking approaches outperform the supervised baseline. This points to promising directions on other ways to exploit test data in ranking. This particular problem formulation (Semi-supervised Rank Learning in Figure 2) is pertinent because there is usually a long tail in search queries (i.e.

many test queries in practice will not have been seen in the training set). Future work will include exploring other ways to use unlabeled data, such as using the low density separation assumption under the general transductive framework.

References

[1] S. Abney. Understanding the Yarowsky Algorithm. *Computational Linguistics*, 30(3):365–395, 2004.

[2] S. Agarwal. Ranking on graph data. In *International Conference on Machine Learning*, 2006.

[3] Y. Altun, D. McAllester, and M. Belkin. Maximum margin semi-supervised learning for structured variables. In *Advances in Neural Information Processing Systems*, 2005.

[4] R. Ando and T. Zhang. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. Technical report, IBM T.J. Watson Research Labs, 2004.

[5] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. In *Journal of Machine Learning Research*, volume 6, pages 1817–1853, 2005.

[6] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold Regularization: a Geometric Framework for Learning from Examples. Technical Report TR-2004-06, University of Chicago, 2004.

[7] M. Belkin, P. Niyogi, and V. Sindhwani. On Manifold Regularization. In *International Workshop on Artificial Intelligence and Statistics*, 2005.

[8] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems*, 2006.

[9] K. P. Bennett and A. Demiriz. Semi-Supervised Support Vector Machines. In *Advances in Neural Information Processing Systems 12*, pages 368–374, 1998.

[10] K. P. Bennett, A. Demiriz, and R. Maclin. Exploiting Unlabeled Data in Ensemble Methods. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, 2002.

[11] S. Bickel, M. Brckner, and T. Scheffer. Discriminative learning for differing training and test distributions. In *ICML*, 2007.

[12] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *EMNLP*, 2006.

[13] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proc. 19th International Conference on Machine Learning (ICML-2001)*, 2001.

[14] A. Blum, J. Lafferty, M. Rewbangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proc. International Conference on Machine Learning (ICML-2004)*, 2004.

[15] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of Computational Learning Theory*, 1998.

[16] L. Bottou and V. N. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[17] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems*, 2006.

[18] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, 2005.

[19] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise to listwise approach. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[20] V. Castelli and T. M. Cover. The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory*, pages 2102–2117, 1996.

[21] O. Chapelle, J. Weston, and B. Schoelkopf. Cluster kernels for semi-supervised learning. In *Proceedings of Neural Information Proccessing Systems*, 2003.

[22] D. Chen, J. Yan, G. Wang, Y. Xiong, W. Fan, and Z. Chen. Transrank: A novel algorithm for transfer of rank learning. In *IEEE International Conference on Data Mining (ICDM), Workshop on Domain Driven Data Mining*, 2008.

[23] W. Chu and Z. Ghahramani. Extensions of Gaussian processes for ranking: semi-supervised and active learning. In *NIPS Workshop on Learning to Rank*, 2005.

[24] W. S. Cooper, F. C. Gey, and D. P. Dabney. Probabilistic retrieval based on staged logistic regression. In *SIGIR*, 1992.

[25] A. Corduneanu and T. Jaakkola. Distributed information regularization on graphs. In *Proceedings of Neural Information Proccessing Systems*, 2004.

[26] D. Cossock and T. Zhang. Subset ranking using regression. In *Conference on Learning Theory (COLT)*, 2006.

[27] D. Yarowsky. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1995.

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

[29] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. Adacost: misclassification cost-sensitive boosting. In *In Proc. 16th International Conf. on Machine Learning*. Morgan Kaufmann, 1999.

[30] 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.

[31] A. Fujii. Modeling anchor text and classifying queries to enhance web document retrieval. In *WWW*, 2008.

[32] G. Fung and O. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. Technical Report TR 99-05, University of Wisconsin, Data Mining Institute, 1999.

[33] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR*, 2008.

[34] C. Goutte, H. Déjean, E. Gaussier, N. Cancedda, and J.-M. Renders. Combining labelled and unlabelled data: a case study on Fisher kernels and transductive inference for biological entity recognition. In *Proceedings of the 6th Conference on Natural language learning (CoNLL)*, pages 1–7, 2002.

[35] Y. Grandvalet and Y. Bengio. Semi-supervised Learning by Entropy Minimization. In *Proceedings of Neural Information Proccessing Systems*, 2004.

[36] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Conference on Multimedia*, 2004.

[37] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *International Conference on Artificial Neural Networks (ICANN)*, 1999.

[38] A. Holub, M. Welling, and P. Perona. Exploiting unlabelled data for hybrid object classification. In *NIPS 2005 Workshop in Inter-Class Transfer*, 2005.

[39] J. Huang, A. J. Smola, A. Gretton, K. M. Borgwardt, and B. Schölkopf. Correcting sample selection bias by unlabeled data. In *NIPS*, 2007.

[40] J. Lafferty and X. Zhu and Y. Liu. Kernel Conditional Random Fields: Representation and Clique Selection. In *Proceedings of International Conference on Machine Learning*, 2004.

[41] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, 1998.

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

[43] J. Jiang. A literature survey on domain adaptation of statistical classifiers.

[44] T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2002.

[45] T. Joachims. Transductive learning via spectral graph partitioning. In *International Conference on Machine Learning (ICML)*, 2003.

[46] T. Joachims and F. Radlinski. Search engines that learn from implicit feedback. *IEEE Computer*, 40(8), 2007.

[47] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: A bibliography. *ACM SIGIR Forum*, 37(2):1828, 2003.

[48] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999.

[49] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. of the International Conference on Machine Learning*, 2002.

[50] N. Lawrence and M. Jordan. Semi-supervised learning with gaussian processes. In *Advances in Neural Information Processing Systems*, 2005.

[51] W. Li and A. McCallum. Semi-supervised sequence modeling with syntactic topic models. In *AAAI-05, The 20th National Conference on Artificial Intelligence*, 2005.

[52] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR Workshop on Learning to Rank for IR (LR4IR)*, 2007.

[53] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[54] L. Mason, J. Baxter, P. Bartless, and M. Frean. Boosting as gradient descent. In *Advances in Neural Information Processing Systems*, 2000.

[55] D. Metzler. Direct maximization of rank-based metrics. Technical Report 432, Univ. of Massachusetts, Amherst, CIIR, 2006.

[56] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 30(3), 2000.

[57] C. Oliveira, F. Cozman, and I. Cohen. Splitting the unsupervised and supervised components of semi-supervised learning. In *ICML 2005 Workshop on Learning with Partially Classified Training Data*, 2005.

[58] R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: transfer learning from unlabeled data. In *International Conference on Machine Learning*, 2007.

[59] S. Robertson. Overview of the Okapi projects. *Journal of Documentation*, 53(1), 1997.

[60] C. Rudin. Ranking with a p-norm push. In *Conference on Learning Theory (COLT)*, 2006.

[61] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3), 1999.

[62] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as kernel eigenvalue problem. *Neural Computation*, 10, 1998.

[63] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge Univ. Press, 2004.

[64] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inferenc*, 90, 2000.

[65] A. Smola and R. Kondor. Kernels and regularization on graphs. In *Conference on Learning Theory (COLT)*, 2003.

[66] A. Smola, O. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. Technical Report 99-03, University of Wisconsin, Data Mining Institute, 1999.

[67] M. Sugiyama, T. Suzuki, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanbe. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics*, 60(4), 2008.

[68] T. Joachims. Transductive Inference for Text Classification using Support Vector Machines. In *International Conference on Machine Learning*, 1999.

[69] T. Truong, M.-R. Amini, and P. Gallinari. Learning to rank with partially labeled training data. In *International Conference on Multidisciplinary Infomation Science and Technology*, 2006.

[70] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: A ranking method with fidelity loss. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[71] V. Vapnik. *Statistical Learning Theory*. Springer, 1998.

[72] A. Veloso, H. Almeida, M. Goncalves, and W. M. Jr. Learning to rank at query-time using association rules. In *SIGIR*, 2008.

[73] J. Wang, M. Li, Z. Li, and W.-Y. Ma. Learning ranking function via relevance propagation. Technical report, Microsoft Research Asia, 2005.

[74] J. Weston, R. Kuang, C. Leslie, and W. S. Noble. Protein ranking by semi-supervised network propagation. *BMC Bioinformatics*, 7, 2006.

[75] Q. Wu, C. J. Burges, K. Svore, and J. Gao. Ranking, boosting, and model adaptation. Technical report, Microsoft Research, 2008.

[76] X. Zhu and Z. Ghahramani. Towards semisupervised classification with Markov Random Fields. Technical Report CMU-CALD-02-106, Carnegie Mellon University, 2002.

[77] X. Zhu and Z. Ghahramani and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proc. of ICML-2003*, 2003.

[78] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank - theory and algorithm. In *ICML*, 2008.

[79] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 1996.

[80] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[81] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[82] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001.

[83] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schökopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems*, 2004.

[84] X. Zhu. Semi-Supervised Learning Literature Survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. http://www.cs.wisc.edu/∼jerryzhu/pub/ssl_survey.pdf.