

Learning to Rank with Partially-Labeled Data

Kevin Duh *
Dept. of Electrical Engineering
University of Washington
Seattle, WA, USA
kevinduh@u.washington.edu

Katrin Kirchhoff
Dept. of Electrical Engineering
University of Washington
Seattle, WA, USA
katrin@ee.washington.edu

ABSTRACT

Ranking algorithms, whose goal is to appropriately order a set of objects/documents, are an important component of information retrieval systems. Previous work on ranking algorithms has focused on cases where only labeled data is available for training (i.e. supervised learning). In this paper, we consider the question whether unlabeled (test) data can be exploited to improve ranking performance. We present a framework for transductive learning of ranking functions and show that the answer is affirmative. Our framework is based on generating better features from the test data (via KernelPCA) and incorporating such features via Boosting, thus learning different ranking functions adapted to the individual test queries. We evaluate this method on the LETOR (TREC, OHSUMED) dataset and demonstrate significant improvements.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

Information retrieval, Learning to rank, Transductive learning, Boosting, Kernel principal components analysis

1. INTRODUCTION

Ranking algorithms, whose goal is to appropriately order a set of objects/documents, are an important component of information retrieval (IR) systems. In applications such as web search, accurately presenting the most relevant documents to satisfy an information need is of utmost importance: a suboptimal ranking of search results may frustrate the entire user experience.

*Work supported by an NSF Graduate Research Fellowship

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGIR '08 July 20–24, 2008, Singapore

Copyright 2008 ACM 978-1-60558-164-4/08/07 ...\$5.00.

While many methods have been proposed to tackle the (document) ranking problem, a recent and promising research direction is to apply techniques in machine learning. In this approach, a training set consisting of user queries and the corresponding list of retrieved documents and relevance judgements is provided to the machine learning algorithm. The relevance judgments may be provided by a human annotator or an implicit feedback mechanism (e.g. query logs) [13]. The algorithm then learns a “ranking function” that predicts relevance judgments close to that of the training set. Thus far, much research has focused on (a) different learning algorithms [8, 7, 5, 11, 35, 17], and (b) the interplay between optimization objectives and IR evaluation measures [20, 28, 32, 4, 33].

We explore an orthogonal research direction here: We ask the question, “Can additional unlabeled data (i.e. query-document pairs without relevance judgments) be exploited to improve ranking performance?” In particular, we consider the case known as transductive learning, where such unlabeled data is the test data to be evaluated.

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-doc-labels. 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 $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$. 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.

Due to its promise, transductive learning (and more generally, semi-supervised learning¹) is an active area of research in machine learning. Previous work mostly focused on classification problems; work on semi-supervised ranking is beginning to emerge.

In this paper, we demonstrate that “learning to rank with both labeled and unlabeled data” is a research direction worth exploring. Our contribution is a flexible transductive framework that plugs in and improves upon existing supervised rankers. We demonstrate promising results on TREC and OHSUMED and discuss a variety of future research directions.

The paper is divided as follows: Section 2 outlines the

¹Semi-supervised (inductive) learning is more general in that the unlabeled data E need not be the test set; it can give predictions and be evaluated on another previously unseen and unlabeled data. See [37] for a good review.

proposed transductive framework. Section 3 presents the main experimental results. Then, Section 4 raises several questions and delves into more detailed analyses of results. Finally, Section 5 draws connections to various related work and Section 6 ends with a discussion on future research.

2. TRANSDUCTIVE META-ALGORITHM

2.1 General Framework

When designing a machine learning solution to a problem, there are several commonly chosen strategies:

- (a) Engineering better features to characterize the data.
- (b) Designing an objective function that more closely approximates the application-specific evaluation metric.
- (c) Developing a more effective algorithm for optimizing the objective.

In this work, we explore option (a). The key idea is to automatically derive better features using the unlabeled test data. In particular, an unsupervised learning method (i.e. a learning algorithm that does not require labels, e.g. clustering, principal components analysis) is applied to discover salient patterns (P_u) in each list of retrieved *test* documents \mathbf{d}_u . The training data is projected on the directions of these patterns and the resulting numerical values are added as new features. The main assumption is that this new training set better characterizes the test data, and thus should outperform the original training set when learning rank functions.

Algorithm 1 Transductive meta-algorithm

Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}$

Input: Test set $E = \{(q_u, \mathbf{d}_u)\}_{u=1..U}$

Input: DISCOVER(), unsupervised algorithm for discovering salient patterns

Input: LEARN(), a supervised ranking algorithm

Output: Predicted rankings for test: $\{\mathbf{y}_u\}_{u=1..U}$

```

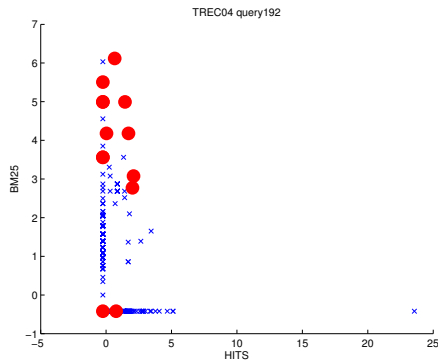
1: for  $u = 1$  to  $U$  do
2:    $P_u = \text{DISCOVER}(\mathbf{d}_u)$  # find patterns on test data
3:    $\hat{\mathbf{d}}_u = P_u(\mathbf{d}_u)$  # project test data along  $P_u$ 
4:   for  $l = 1$  to  $L$  do
5:      $\hat{\mathbf{d}}_l = P_u(\mathbf{d}_l)$  # project train data along  $P_u$ 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

```

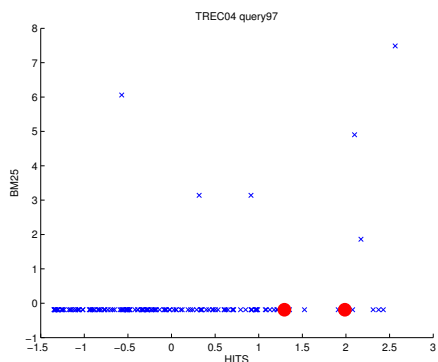
Algorithm 1 shows the pseudocode for this meta-algorithm. DISCOVER() is the generic unsupervised method. It is important to note that it is applied to each test document list \mathbf{d}_u separately (line 2). This is because queries are formulated independently and that different \mathbf{d}_u exhibit different salient patterns.² LEARN() is the 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 1(a) and 1(b).

²One can imagine extensions where successive queries are dependent. See Section 6.



(a)



(b)

Figure 1: 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.

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. For the first query (Fig. 1(a)), we see that the data varies mostly along the y-axis (BM25); for the second query (Fig 1(b)), the variation is on the x-axis (HITS). These 2 document lists would be better ranked by 2 different rankers, e.g. one which ranks documents with $\text{BM25} > 2.5$ as relevant, and the second which ranks documents with $\text{HITS} > 1.25$ 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)[24] as the unsupervised method and Rank-Boost[8] as the supervised ranker. In theory, any algorithm 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 will argue here

that Kernel PCA and RankBoost is one such good pair, but there may well be others.

2.2 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 most 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 [24] 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 \mathbf{d}^j be the original feature vector of document j .³ Then Kernel PCA can be seen as the following procedure:

1. Map each doc \mathbf{d}^j to a new space $\mathbf{d}^j \mapsto \Phi(\mathbf{d}^j)$, where $\Phi(\cdot)$ is the (non-linear/high-dimension) mapping.
2. Compute covariance matrix in this new space:
 $\mathbf{C} = \frac{1}{m} \sum_{j=1}^m \Phi(\mathbf{d}^j) \Phi(\mathbf{d}^j)^T$. ($T =$ transpose)
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 [24] for the derivation; here we only present the steps needed for this paper:

1. Define a kernel function $k(\cdot, \cdot) : (\mathbf{d}^j, \mathbf{d}^{j'}) \rightarrow \mathbb{R}$ which maps two document vectors to a real number indicating the similarity between the two documents.
2. There exist kernels of the form $k(\mathbf{d}^j, \mathbf{d}^{j'}) = \langle \Phi(\mathbf{d}^j), \Phi(\mathbf{d}^{j'}) \rangle$, (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. $K_{jj'} = k(\mathbf{d}^j, \mathbf{d}^{j'}) \forall j, j' \in \{1, 2, \dots, m\}$.
4. Kernel PCA reduces to solving the eigen-problem $m\lambda\alpha = K\alpha$. We pick only the α with the largest eigenvalues.
5. For a new document \mathbf{d}^n , its principal component is computed as $\sum_{j=1}^m \alpha_j k(\mathbf{d}^j, \mathbf{d}^n)$.

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 , $k(\mathbf{d}^j, \mathbf{d}^{j'}) = \langle \mathbf{d}^j, \mathbf{d}^{j'} \rangle^p$.

³In 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; \mathbf{d}^j is one document vector within the list.

- **Radial basis function:** $k(\mathbf{d}^j, \mathbf{d}^{j'}) = \exp(-\frac{\|\mathbf{d}^j - \mathbf{d}^{j'}\|}{2\sigma})$. This is an isotropic kernel, with bandwidth σ adjusting for smoothness.
- **Diffusion kernel** [15]: 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 $1/\|\mathbf{d}^j - \mathbf{d}^{j'}\|$. $k(\mathbf{d}^j, \mathbf{d}^{j'})$ is defined by running a lazy random walk from \mathbf{d}^j to $\mathbf{d}^{j'}$. 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:** $k(\mathbf{d}^j, \mathbf{d}^{j'}) = \langle \mathbf{d}^j, \mathbf{d}^{j'} \rangle$. 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 [26] is based on sparsity constraints and can extract patterns in $O(m)$.

2.3 Rankboost

RankBoost[8] is an extension of the boosting philosophy [23] 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 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 $h(\mathbf{d}^j) = 1$ if "BM25 score > 1" and $h(\mathbf{d}^j) = 0$ otherwise. The final ranking function of RankBoost is a weighted combination of T weak learners:

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

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 $y^j = F(\mathbf{d}^j)$ for each document and sorting the list by the value of y^j .

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 (e.g. RankSVM), 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 [19] 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.

Finally, for clarity, we present a concrete walk-through of a search scenario using our transductive framework:

0. A training set is prepared and stored in memory/disk.
1. User submits a query q_u
2. Initial IR engine returns possibly relevant documents \mathbf{d}_u .
3. Run Algorithm 1, lines 2 to 8. Output \mathbf{y}_u .
4. Rank the documents \mathbf{d}_u by \mathbf{y}_u . Return results to user.

In contrast to the supervised approach, the practical challenges here are: (a) scalable storage of the training set, (b) fast computation of Algorithm 1 (during user wait). We leave these computational issues as future work.

3. MAIN RESULTS

3.1 Data and Setup

We perform experiments on the LETOR dataset [18], 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 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 [22], HITS [14], and Language Model [34]. It is a good dataset for our experiments since we will be discovering patterns from features that have already been proven to work. 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 [18] for details.

Our experimental setup compares three rankers. The **baseline** is a supervised RankBoost, trained on the original training data. This is compared with **transductive**, which is Algorithm 1 with **LEARN()** being RankBoost and **DISCOVER()** being Kernel PCA run 5 times with different kernels:

- Polynomial kernel (**poly**), order=2
- Radial basis function kernel (**rbf**), bandwidth=1
- Diffusion kernel (**diff**), time constant=1, 10-nn graph
- Diffusion kernel (**diff**), time constant=10, 10-nn graph
- Linear kernel (**linear**) (corresponds to linear PCA)

We create 25 additional features representing the first 5 principal components from each of the above 5 kernels.

Table 1: Data characteristics

| | TREC’03 | TREC’04 | OHSUMED |
|-------------------------|---------|---------|---------|
| #query | 50 | 75 | 106 |
| #document | 49k | 74k | 16k |
| avg #doc/query | 983.4 | 988.9 | 152.3 |
| #relevant doc | 516 | 1600 | 4.8k |
| avg #relevant/query | 1 | 0.6 | 28 |
| avg #doc pairs | 302k | 262k | 369k |
| #feature (original) | 44 | 44 | 25 |
| #feature (transductive) | 54 | 69 | 50 |

The third ranker (**combined**) performs a simple system combination of **baseline** and **transductive**. For each test query, it first normalizes the outputs \mathbf{y}_u of **baseline** and **transductive**, then returns the averages of these normalized outputs. While more sophisticated combination methods have been proposed (c.f. [16]), here we investigate whether a simple unweighted average is sufficient to give improvements.

Following LETOR convention, each dataset is divided into 5 folds with a 3:1:1 ratio for training, validation, and test set. We use the validation set to decide which kernels to use in the **transductive** system. The **transductive** system employs all 5 kernels (25 features) in TREC’04 and OHSUMED, but only **poly** and **rbf** kernels (10 features) in TREC’03. We decided not to use the validation set to pick features in a more fine-grained fashion (e.g. how many principal components, what kernel parameters) since we expect the optimal settings to vary for each test query, and relying on RankBoost’s inherent feature selection ability is a more efficient and effective solution. For both **transductive** and **baseline**, RankBoost is run for a total of 150 iterations, determined from initial experiments. We conjecture that **transductive** may be improved by automatically tuning for the best number of boosting iterations for each test query, but we leave that to future work.

The evaluation metrics are mean averaged precision (MAP) and normalized discount cumulative gain (NDCG@n) [12]. We report results from the average of 5-fold cross-validation and judge statistical significance using the dependent t-test.

3.2 Overall Evaluation

Figure 2 compares the 3 systems on the 3 datasets. Table 2 shows the same results in numbers (boldfaced MAP/NDCG numbers indicate a statistically significant improvement ($p < 0.05$) over **baseline**.)⁴ The main observations are:

1. Both **transductive** and **combined** outperform **baseline** on all datasets and all metrics.
2. For TREC’03, all improvements are statistically significant. For TREC’04 and OHSUMED, only **combined** gives a statistically significant improvement.

⁴Our RankBoost baseline is comparable but different from LETOR[18], mainly due to different feature normalization (mean-variance vs. [0,1] scaling). For reference, we report the baseline score comparison in the following format:

MAP(ours/LETOR) NDCG@1(ours/LETOR)
TREC’03: MAP(.1880/.2125) N@1(.3200/.2600)
TREC’04: MAP(.3524/.3835) N@1(.4400/.4800)
OHSUMED: MAP(.4427/.4403) N@1(.5252/.4977)

All results here are compared to our version of the baseline.

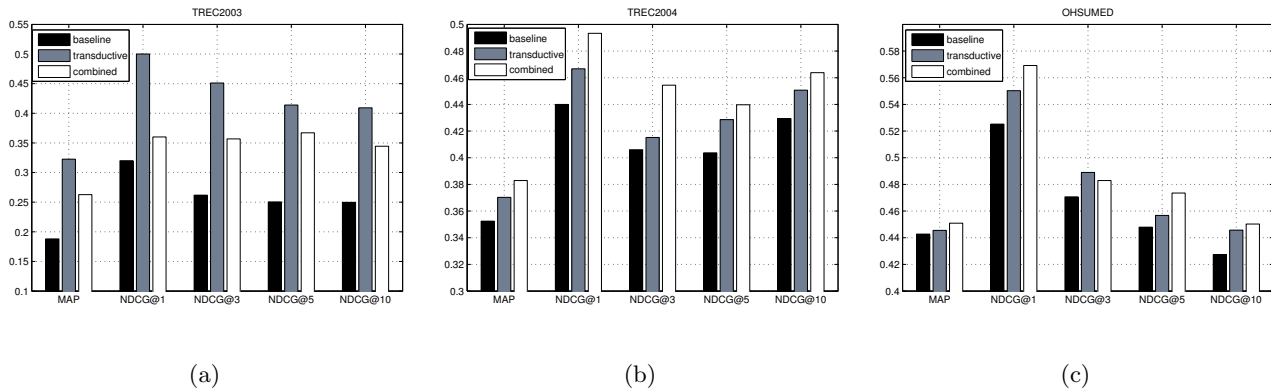


Figure 2: Main result (MAP and NDCG@n) for (a) TREC'03, (b) TREC'04, (c) OHSUMED. Both transductive and combined systems outperform baseline in all datasets and all metrics.

Table 2: Main result (Figure 2 in table form). All transductive/combined improve on baseline. Statistically significant improvements are bold-faced.

| | MAP | N@1 | N@3 | N@5 | N@10 |
|--------------|--------------|--------------|--------------|--------------|--------------|
| TREC'03 | | | | | |
| baseline | .1880 | .3200 | .2619 | .2504 | .2498 |
| transductive | .3226 | .5000 | .4512 | .4140 | .4092 |
| combined | .2627 | .3600 | .3568 | .3670 | .3444 |
| TREC'04 | | | | | |
| baseline | .3524 | .4400 | .4060 | .4037 | .4294 |
| transductive | .3703 | .4667 | .4152 | .4286 | .4507 |
| combined | .3829 | .4933 | .4544 | .4397 | .4638 |
| OHSUMED | | | | | |
| baseline | .4427 | .5252 | .4707 | .4479 | .4274 |
| transductive | .4455 | .5503 | .4890 | .4567 | .4457 |
| combined | .4509 | .5692 | .4829 | .4735 | .4503 |

- In TREC'04 and OHSUMED, combined improves on both transductive and baseline, implying that transductive and baseline make different kinds of errors and thus benefit from system combination. In TREC'03, combined is worse than transductive—a simple averaging is less likely to work if one system (baseline) is significantly worse in performance than the other.

We believe these are very promising results which demonstrate that transductive learning (and the framework we propose in Algorithm 1) has the potential to improve ranking.

4. DETAILED ANALYSIS

We ask several questions in order to examine the properties of the transductive framework in more detail. These questions seek to answer what is beneficial and not beneficial in the proposed framework.

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

Are we seeing gains in transductive simply because Kernel PCA extracts good features per se, or particularly because the features are extracted on the test set (i.e. the transductive aspect)? To answer this, we built a new system, KPCA on train, which runs Kernel PCA on each training

list (as opposed to projecting the training lists to principal directions of the test lists). We then train RankBoost on this data, which is the same training data as baseline except for the additional Kernel PCA features. This new RankBoost is then evaluated on the test set. The results (Table 3) show that KPCA on train is worse than transductive (e.g. .2534 vs. .3226 MAP for TREC'03), implying that transductive aspect of adapting to each test query is essential.

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

| | TREC'03 | TREC'04 | OHSUMED |
|---------------|---------|---------|---------|
| Transductive | .3226 | .3703 | .4435 |
| KPCA on train | .2534 | .3558 | .4420 |
| Baseline | .1880 | .3523 | .4427 |

2. How can Kernel PCA features be interpreted?

Kernel PCA features are in general difficult to interpret because they involve non-linear combinations and the α generated from the eigen-problem represents weights on samples, not features. We can only get a rough answer by computing the correlation between the values of a Kernel PCA feature and an original feature. Table 4 lists some features that correlate with particular kernel PCA features (e.g. in TREC'04 query10, the Diffusion feature correlated highly with HITS). It is important to note, however, that this kind of analysis only serves as extra reference to help us understand particular test queries: most Kernel PCA features have little correlation to original features. The average correlation on TREC'04 is less than 0.15.

3. What are the most useful features?

What weak learners $h(\cdot)$ in transductive's 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 transductive 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

Table 4: Some examples of original features that correlate highly with Kernel PCA features (coeff. of determination in parentheses). However, most features (not listed) have low correlation due to their non-linear relationship.

| | Polynomial | Diffusion | Linear |
|-----------------|--------------------|---|------------------------------|
| TREC'03 query2 | none | Hyperlink feature prop, weighted out-link (.66) | LMIR.JM of anchor (.70) |
| TREC'04 query10 | dl of anchor (.99) | HITS authority (.89) | LMIR.ABS of title (.68) |
| OHSUMED query1 | none | BM25 of title+abstract (.78) | BM25 of title+abstract (.82) |

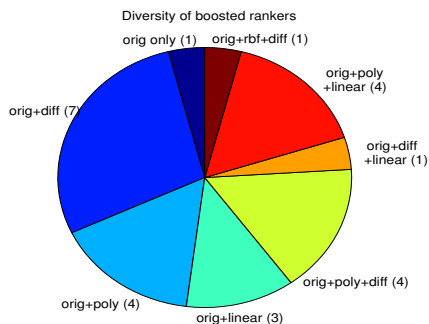


Figure 3: Top 5 feature combinations employed in RankBoost, by count. There is a diversity of feature combinations that lead to good performance, indicating that different test queries require different rankers.

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 3): no single combination is more prevalent than others. This again supports the intuition that test-specific rankers are better than a single general ranker.

4. Linear vs. Non-linear PCA

How important is the non-linear aspect of Kernel PCA? Would we have achieved similar gains if we restrict **transductive** to perform only linear PCA? To test this, we trained new systems consisting of original features plus 5 linear PCA features, vs. original features + 5 polynomial, rbf, or diffusion kernel features. On TREC'04, we observe the MAP scores, in order: .3701 (rbf), .3670 (poly), .3627 (diff), .3614 (linear). However, on TREC'03, linear is not the worst: .3032 (diff), .2996 (linear), .2895 (poly), .2754 (rbf). Thus, non-linearity is important in most cases, but one should not expect non-linear kernels to always outperform linear ones. The best strategy is to employ multiple kernels.

5. How does performance vary across queries?

In Section 3, we present overall results averaged over all test queries. A more detailed analysis would include per-query MAP and NDCG. Figure 4 reports a histogram of queries that are improved vs. degraded by **transductive**.

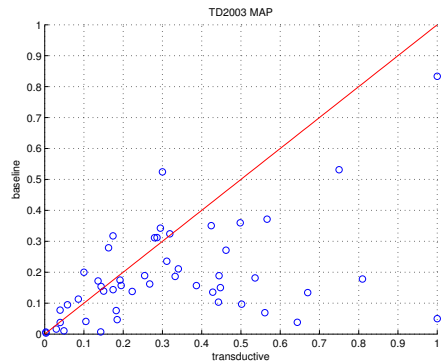


Figure 5: Scatterplot of TREC'03 MAP results for transductive (x-axis) vs. baseline (y-axis).

For each plot, the bars on the right side indicates the number of queries that improved more than 1%, 20%, and 50% over **baseline**. Bars on the left side indicate the number of queries that become more than 1%, 20%, and 50% worse than **baseline**.

One observes that our transductive approach does not give improvements across all queries. We are seeing gains in Section 3 because the proportion of improved queries is greater than that of degraded queries (especially for TREC'03).

It would be helpful to understand exactly the conditions where the transductive approach is beneficial vs. harmful. On TREC'03, there is slight evidence showing that **transductive** seems to benefit queries with poorer baselines (See Figure 5, scatterplot of baseline and transductive MAP scores). One hypothesis is that the original features of more difficult queries are not sufficiently discriminative, so Kernel PCA has more opportunity to show improvements. However, this trend is not observed in TREC'04.

We also attempted to see if differences at the query level correlates with e.g. (a) number of documents, (b) number of relevant documents, (c) pairwise ranking accuracy in training, but no single factor reliably predicts whether a query will be benefited by the transductive ranker.

6. What is the computation time?

Ideally, both DISCOVER() and LEARN() need to operate at real-time since they are called for each test query. In our experiments on a Intel x86-32 (3GHz CPU), KernelPCA (implemented in Matlab/C-MEX) took on average 23sec/query for TREC and 4.3sec/query for OHSUMED; Rankboost (implemented in C++) took 1.4sec/iteration for TREC and 0.7sec/iteration for OHSUMED. The total compute time per query (assuming 150 iterations) is around 233sec/query for TREC and 109sec/query for OHSUMED. It remains to be seen whether real-time computation can be achieved by better code optimization or novel distributed algorithms.

5. RELATED WORK

There are generally two types of problems in “learning to rank with partially-labeled data.” In the problem we consider here, the document lists in our dataset are either fully labeled $\{\mathbf{d}_l\}_{l=1..L}$, or not labeled whatsoever $\{\mathbf{d}_u\}_{u=1..U}$. The second type of problem is concerned with the case when a document list \mathbf{d} is only partially-labeled, i.e. some docu-

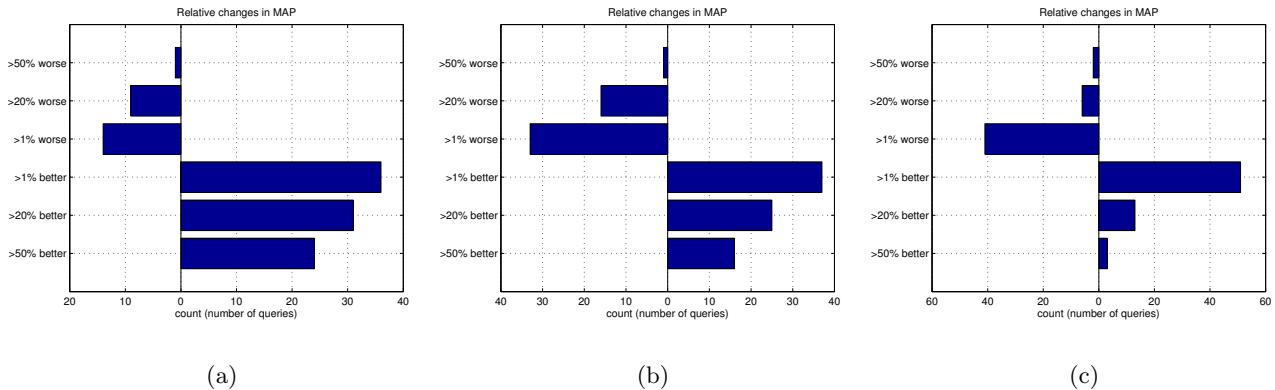


Figure 4: Query-level changes in MAP: We show the number of queries (in transductive) that improved/degraded compared to baseline. In TREC'03 (a), the majority of queries improved, but in TREC'04 (b) and OHSUMED (c) a significant proportion degraded. See text for more explanation.

ments 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 an implicit feedback mechanism [13] to generate labels and some documents simply cannot acquire labels with high confidence. So far the research community does not yet have 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*”.

Several methods have been proposed for the *Missing Labels* problem, e.g. [36, 30, 10, 29]: 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 [36] (transductive), or one can train a ranking function using these values as true labels [10, 29] (inductive). One important point about these label propagation methods is that they do not explicitly model the relationship that document \mathbf{d}^j is ranked above, say, \mathbf{d}^k . Instead it simply assumes that the label value for \mathbf{d}^j is higher than that of \mathbf{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 [1]. 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. [6] also proposes a graph-based regularization term, but in contrast to [1], 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, [29] investigates label propagation across queries, but concluded that it is computationally prohibitive. Beyond the compu-

tational 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, [27]⁵ 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 propagation. A new supervised ranker is then trained to maximize accuracy on the labeled set while minimizing ranking difference to label propagation results. Thus this is a bootstrapping approach that relies on the initial ranker producing relatively accurate seeds.

Our feature-based approach differs from the above graph-based and bootstrapping approaches, and is more similar to work in feature extraction for domain adaptation [3, 21] or multi-task learning [2]. In fact, one can consider transductive learning as an extreme form of domain adaptation, where one adapts only to the given test set.

Finally, we wish to note that various approaches incorporating similar semi-supervised ideas have been explored in the IR community. For instance, query expansion by pseudo-relevance feedback [31] can be thought as a query-specific transductive techniques that uses the bootstrapping assumption (i.e. top retrieved documents are relevant and can be exploited).

6. CONCLUSIONS

We present a flexible transductive framework for learning ranking functions. The main intuition is based on extracting features from test documents and learning query-specific rankers. Our experiments using Kernel PCA with RankBoost demonstrate significant improvements on TREC and OHSUMED, and point to a promising area of further research. Possible extensions and future work include:

- Investigating different algorithm pairs for DISCOVER() and LEARN().

⁵Although this paper has the same title as ours, the approach is entirely unrelated.

- Implementing kernels that directly capture semantic relationships between documents, as opposed to relying on a vectorial bag of features representation of documents.
- Automatic tuning of parameters (e.g. kernel parameters, number of features) on a per-query basis. One possibility is to employ stability analysis [25] and calculate residual errors for tuning Kernel PCA.
- Explicitly incorporating a feature selection component (e.g.[9]).
- Understanding the exact conditions in which test-specific ranking functions work well.
- Computational speed-ups for practical deployment: for instance, can we re-use previously-trained ranking functions?
- Modeling dependent queries in an interactive search scenario: (1) user issues initial query, (2) transductive system returns ranked documents, (3) user reformulates query, (4) transductive system now learns from both previous and new document list.

7. REFERENCES

- [1] S. Agarwal. Ranking on graph data. In *ICML*, 2006.
- [2] 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, 2005.
- [3] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *Neural Information Processing Systems*, 2006.
- [4] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2006.
- [5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. of the 22nd International Conference on Machine Learning*, 2005.
- [6] W. Chu and Z. Ghahramani. Extensions of gaussian processes for ranking: semi-supervised and active learning. In *NIPS Wksp on Learning to Rank*, 2005.
- [7] K. Crammer and Y. Singer. Pranking with ranking. In *Neural Information Processing Systems (NIPS)*, 2001.
- [8] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4, 2003.
- [9] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature selection for ranking. In *SIGIR*, 2007.
- [10] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Conf. on Multimedia*, 2004.
- [11] R. Herbrich, T. Graepel, and K. Obermayer. *Advances in Large Margin Classifiers*, chapter Large margin rank boundaries for ordinal regression. MIT Press, 2000.
- [12] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.
- [13] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, 2003.
- [14] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999.
- [15] I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002.
- [16] J. H. Lee. Analysis of multiple evidence combination. In *SIGIR*, 1997.
- [17] P. Li, C. Burges, and Q. Wu. McRank: Learning to rank using classification and gradient boosting. In *NIPS*, 2007.
- [18] 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.
- [19] L. Mason, J. Baxter, P. Bartless, and M. Frean. Boosting as gradient descent. In *NIPS*, 2000.
- [20] D. Metzler. Direct maximization of rank-based metrics. Technical report, University of Massachusetts, Amherst CIIR, 2005.
- [21] R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: transfer learning from unlabeled data. In *ICML*, 2007.
- [22] S. Robertson. Overview of the Okapi projects. *Journal of Documentation*, 53(1), 1997.
- [23] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3), 1999.
- [24] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as kernel eigenvalue problem. *Neural Computation*, 10, 1998.
- [25] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge Univ. Press, 2004.
- [26] A. Smola, O. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. Technical Report 99-03, University of Wisconsin, Data Mining Institute, 1999.
- [27] T. Truong, M.-R. Amini, and P. Gallinari. Learning to rank with partially labeled training data. In *Int'l Conf. on Multidisciplinary Info Sci/Tech*, 2006.
- [28] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: a ranking method with fidelity loss. In *SIGIR*, 2007.
- [29] J. Wang, M. Li, Z. Li, and W.-Y. Ma. Learning ranking function via relevance propagation. Technical report, Microsoft Research Asia, 2005.
- [30] J. Weston, R. Kuang, C. Leslie, and W. S. Noble. Protein ranking by semi-supervised network propagation. *BMC Bioinformatics*, 7, 2006.
- [31] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR*, 1996.
- [32] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. In *SIGIR*, 2007.
- [33] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, 2007.
- [34] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR*, 2001.
- [35] Z. Zheng, H. Zha, K. Chen, and G. Sun. A regression framework for learning ranking functions using relative relevance judgements. In *SIGIR*, 2007.
- [36] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *NIPS*, 2004.
- [37] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, University of Wisconsin, Madison, Computer Science Dept., 2005.