Learning to Rank with Partially-Labeled Data

Kevin K. Duh

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2009

Program Authorized to Offer Degree: Electrical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Kevin K. Duh

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

Katrin Kirchoff

Reading Committee:

Katrin Kirchhoff

Mari Ostendorf

Jeffrey A. Bilmes

Date:

University of Washington

Abstract

Learning to Rank with Partially-Labeled Data

Kevin K. Duh

Chair of the Supervisory Committee:
Professor Katrin Kirchoff
Electrical Engineering

Ranking is a key problem in many applications. In web search, for instance, webpages are ranked such that the most relevant ones are presented to the user first. In machine translation, a set of hypothesized translations are ranked so that the correct one is chosen. Abstractly, the problem of ranking is to predict an ordering over a set of objects. Given the importance of ranking in many applications, "Learning to Rank" has risen as an active research area, crossing disciplines such as machine learning and information retrieval. The approach is to adapt machine learning techniques developed for classification and regression problems to problems with rank structure. However, so far the majority of research has focused on the supervised learning setting. Supervised learning assumes that the ranking algorithm is provided with labeled data indicating the rankings or permutations of objects. Such labels may be expensive to obtain in practice.

The goal of this dissertation is to investigate the problem of ranking in the framework of semi-supervised learning. Semi-supervised learning assumes that data is only partially labeled, i.e. for some sets of objects, labels are not available. This kind of framework seeks to exploit the potentially vast amount of cheap unlabeled data in order to improve upon supervised learning. While both *supervised learning for ranking* and *semi-supervised learning for classification* have become active research themes, the combination, *semi-supervised learning for ranking*, has been less examined. This thesis aims to fill the gap.

The contribution of this thesis is an examination of several ways to exploit unlabeled data in ranking. In particular, four assumptions commonly used in classification (Change of Represen-

tation, Covariate Shift, Low Density Separation, Manifold) are extended to the ranking setting. Their implementations are tested on six real-world datasets from Information Retrieval, Machine Translation, and Computational Biology. The algorithmic contributions of this work include (a) a Local/Transductive meta-algorithm, which allows one to plug in different unlabel data assumptions with relative ease, and (b) a kernel defined on lists, which allow one to extend methods which work with samples (i.e. classification, regression) to methods which work with lists of samples (i.e. ranking). We demonstrate that several assumptions about how unlabeled data helps in classification can be successfully applied to the ranking problem, showing improvements over the supervised baseline under different dataset-method combinations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

BLEU:	A popular machine translation evaluation metric, see Chapter 3
EM:	Expectation-Maximization Algorithm
FG:	Feature Generation approach for local/transductive ranking (Chapter 5)
FG+IW:	Combined Feature Generation and Importance Weighting for local/transductive ranking (Chapter 6)
GDT-TS:	Evaluation metric for Protein Structure Prediction (see [164])
IW:	Importance Weighting approach for local/transductive ranking (Chapter 5)
LETOR:	Learning to Rank dataset, published by Microsoft Research Asia. Consists of TREC and OHSUMED subsets.
MT:	Machine translation
MAP:	Mean average precision; A popular information retrieval evaluation metric, see Chapter 3
MERT:	Minimum Error Rate Training algorithm. A standard algorithm for training Machine Translation systems (see [119]).
NDCG:	Normalized discount cumulative gain; A popular information retrieval evaluation metric, see Chapter 3
OHSUMED:	Information Retrieval dataset (medical search task)

PM:	Pseudo Margin approach for local/transductive ranking (Chapter 7)
IR:	Information retrieval
RANKBOOST:	A supervised ranking algorithm, see Section 5.1.2
SVM:	Support vector machine
TREC:	Information Retrieval dataset (webpage ranking task)

ACKNOWLEDGMENTS

I am enormously grateful to my advisor Katrin Kirchoff for teaching me how to do research. More than anyone, she taught me how to frame a problem, how to devise my experiments, how to think about the results, and finally, how to present it clearly to the research community. Furthermore, I thank her for always being very supportive–I think I would not have endured graduate school while having a family without her encouragement and understanding.

I would also like to thank all the professors at the University who have had an important impact on me: Jeff Bilmes, for exciting my interest in new research directions (e.g. structured prediction, graphical models, social choice theory). Mari Ostendorf, for giving me the opportunity to co-teach with her, and for giving me encouragement when I need it the most. Marina Meila, whose clear lectures gave me a grounding in optimization and math. Bill Noble, who was ever so helpful in teaching me about computational biology. Efthi Eftimiadis, for his huge smile, which made me feel at home when I was new to IR conferences. Les Atlas, for being my mentor and showing me the inside workings of an academic career. Maya Gupta, for being willing to listen to my half-baked ideas and to give me feedback at various occasions. Jeng-Nenq Hwang, who is always so nice as to "adopt" our family during Thanksgiving and other times. Jeff, Mari, Marina, Bill, and Efthi are also on my thesis committee–I thank them for their time and effort.

I was also very fortunate to work with many colleagues outside of UW. Whether it be interning in industry or organizing a workshop, these experiences have given me new perspectives, new skills, and new connections. Sumit Basu, Marine Carpuat, Hal Daume, John Dunagan, Simon Corston-Oliver, Mo Corston-Oliver, Jianfeng Gao, Rebecca Hwa, Zhifei Li, Dekang Lin, Bob Moore, Patrick Nguyen, John Platt, Chris Quirk, Eric Ringger, Mike Schultz, Hisami Suzuki, Qin Wang–I enjoyed every minute working with you and hope we can continue keeping in touch. Sanjeev Khudanpur gave me the best advice at the beginning of my grad school career; to paraphrase: "Go to many talks and *always* ask questions. If you don't understand the talk, you should definitely ask a question. If you understood the talk, you will naturally have questions.” I find this advice useful even now.

Many friends have accompanied me along the way. SSLI Lab is so diverse that wherever I turn, I will find someone with the answer–whether it be a C++ tip, a linguistics question, a brain-storming session at the white board, or a solicitation for food. Thanks especially go to: Andrei Alexandrescu, Amittai Axelrod, Chris Bartels, Costas Boulis, Lee Damon, Karim Filali, Sangyun Hahn, Gang Ji, Jeremy Kahn, Xiao Li, Jon Malkin, Alex Marin, Tim Ng, Taka Shinozaki, Amar Subramanya, Sheila Reynolds, Mei Yang. I am also lucky to have many friends outside of SSLI lab, who continue to help me whenever I call or email. To name a few: Justin Brickell, Pichin Chang, Nels Jewell-Larson, Kristy Hollingshead, Hoifung Poon, Jared Tritz, Matt Walker, Fei Xia.

I would also like to acknowledge the National Science Foundation (NSF) Graduate Fellowship. The fellowship not only gave me generous financial support, but also allowed me the freedom to explore a variety of research topics, something I sincerely appreciated. I learned that research is not just about solving problems, but also about defining problems.

Finally, my utmost thanks go to my family–my parents, my grandparents, my brother, my wife, and my three children: I cannot say how much you all mean to me. You are the ones who make life worth living.

DEDICATION

To my family

Chapter 1
INTRODUCTION

1.1 Motivation

The problem of ranking, whose goal is to predict an ordering over a set of objects, is a key problem in many applications. In web search, for instance, ranking algorithms are used to order webpages in terms of relevance to the user. In speech recognition and machine translation, a set of candidate hypotheses is ranked such that the best transcription or translation emerges near the top[1]. In these applications as well as others (e.g. recommender systems, protein structure prediction, sentiment analysis, online ad placement), the ranking algorithm is a critical component that has important ramifications on final system output; a suboptimal ranking may render the entire system useless.

Due to its wide-spread applicability and importance, the problem of ranking has been gaining much attention in research communities ranging from machine learning to information retrieval and speech/language processing. However, most of the research so far has addressed ranking as a supervised learning problem. This is a restriction since supervised learning requires that all samples in the training set be labeled, which can be costly or prohibitive in real-world applications.

This thesis extends the study of ranking into semi-supervised learning, namely learning to rank using a dataset containing both labeled and unlabeled samples. This has the potential to improve the performance of ranking algorithms while keeping the manual labeling effort scalable. There has been little prior work in this area. Our goal is to study the following questions:

1. What information in unlabeled samples can be exploited in the context of ranking problems? In classification problems, ideas such as the manifold assumption and cluster assumption are used to justify the utility of large amounts of unlabeled data. What assumptions exist for ranking problems?

2. Is there an effective mechanism for adapting the wide range of methods developed for semi-

[1]This is referred to as "re-ranking" or "re-scoring" in the speech and language literature.

supervised classification into semi-supervised ranking? In particular, is there a general framework (i.e. meta-algorithm) that makes it straightforward to apply ideas in classification to ranking?

3. What can we learn by comparing the same semi-supervised ranking algorithm on different kinds of real-world datasets? The No Free Lunch Theorem states that no algorithm can be best on all datasets, but can we acquire rough intuition about what algorithms/assumptions match what datasets best?

In the following sections, I first formally formulate the problem of semi-supervised ranking (Section 1.2). Then, for concreteness of illustration, I briefly describe in Section 1.3 two of the applications of ranking. The contributions of the thesis are summarized in Section 1.4 and an outline of what follows is given in Section 1.5.

1.2 Problem Formulation

The problem of ranking involves learning a ranking model from a *training set* such that it generates a "good" ordering on the *test set*. Formally, let $\{x^{(l)}, y^{(l)}\}_{l=1..L}$ be the labeled training set consisting of L samples, and let $\{x^{(u)}\}_{u=1..U}$ be the unlabeled training set consisting of U samples. In many semi-supervised learning scenarios, unlabeled samples are significantly cheaper to acquire than labeled samples, so it is often the case that $U >> L$. For simplicity of notation, I will use the variable s to index the entire training set when the distinction between labeled and unlabeled data is not important: $s = 1..L, L+1..L+U$. In other words, I will use $x^{(s)}$ to refer to a sample in either the labeled or unlabeled training set. The variable t will be used to index the test set $\{x^{(t)}, y^{(t)}\}$.

A sample $x^{(s)} \in X$ consists of a set of N_s objects $\{x_n^{(s)}\}_{n=1..N_s}$ to be ranked or ordered. Often, an object is represented numerically as a feature vector with dimension d, so $x^{(s)}$ can be thought of as a matrix of N_s by d dimensions. Ranking can be thought of as "shuffling" the rows of this matrix. It is important to notice that N_s is dependent on s: some samples will naturally have more objects to be ordered than others.

The label $y^{(l)} \in Y$ encodes the ranking of $\{x_n^{(l)}\}$. Depending on the problem type, the label either represents a total ordering or a partial ordering. A general encoding for both would be to represent a total/partial ordering as a set of preference relations, $x_i^{(l)} \rhd x_j^{(l)}$ for a set of (i,j) pairs, where $\rhd$ represents $x_i^{(l)}$ is strictly "preferred to" or "ranked higher than" $x_j^{(l)}$. A weak preference $x_i^{(l)} \trianglerighteq x_j^{(l)}$ means that $x_i^{(l)}$ is either strictly preferred or equivalently preferred to $x_j^{(l)}$.[2] More formally, a preference relation $\trianglerighteq$ over a set X is considered a total ordering if it satisfies the following properties. For $a,b,c \in X$,

1. (Reflexivity) $a \trianglerighteq a$
2. (Antisymmetry) If $a \trianglerighteq b$ and $b \trianglerighteq a$, then $a \sim b$ ($\sim$ denotes "equivalently preferred")
3. (Transitivity) If $a \trianglerighteq b$ and $b \trianglerighteq c$, then $a \trianglerighteq c$
4. (Completeness) Either $a \trianglerighteq b$ or $b \trianglerighteq a$ is true.

If only the first three properties are satisfied, $\trianglerighteq$ is called a partial order. The set of preference relations can be shown in graph format where each $x_n^{(l)}$ is a node and each $\rhd$ links two nodes by a directed edge. A partial ordering forms a directed acyclic graph whereas a total ordering forms a linear chain. Clearly, a training set with total ordering contains more labeled information than one with partial ordering.[3]

The goal is to learn a function $h : X \to Y$ that performs well (i.e. minimizes loss) on the test set. While the performance measure is application-specific, there are two broad categories: In the general case, the loss function $l_{total} : Y \times Y \to R$ is based on differences between the true and predicted *total* orderings. In the more specific case, the loss l_{top} is only a function of the top-ranked object in the true and predicted rankings.

1.3 Example Applications

Information retrieval (IR) is a prominent example where ranking is central. Given a user-inputed query, the IR system returns a sorted list of documents that satisfies the user's information need. The

[2]The notations $a \succ b$ and $a \succeq b$ are also used to mean a is (strictly) preferred to b. I personally prefer $\rhd$ to avoid confusing $\succ$ with another similar-looking binary relation, $>$.

[3]A ranking problem with only partial ordering labels may also be thought of as semi-supervised ranking, but we do not use this definition in this thesis. Here, the term "semi-supervised" refers to the fact that some samples are labeled, whether they be total or partial orderings.

sorting should ideally be in order of relevance to the query. When the total number of documents scale up, such as in library search or web search, presenting a short list of relevant documents become an essential task.

The labeled training data $\{x^{(l)}, y^{(l)}\}_{l=1..L}$ for IR corresponds to the following: Each $x^{(l)}$ is a set of vectors, each vector $x_j^{(l)}$ representing a document. There are a total of L queries, thus L sets of documents. Labels $y^{(l)}$ can be thought of as a vector of relevance judgments, which are determined by a human annotator by seeing whether each document in the set is relevant to the given query. The unlabeled data $\{x^{(u)}\}_{u=1..U}$ refers to sets of documents in which there are queries but not associated relevance judgments.

Machine translation (MT) is another example where ranking can be applied. There are considerable differences with IR, however. In MT, the goal is to generate a translation for a given input sentence. The space of translations is theoretically infinite (compared to IR, the set of documents may be large but is finite). Therefore one can think of MT as a “generation” problem as opposed to the “selection” problem of IR. In this case, ranking is useful as a second-stage procedure in MT. The first stage generates a preliminary list of translation candidates; this usually involves algorithms not directly related to ranking. The second stage, which is often called a re-ranker or re-scorer, involves ranking the set of translations.

The labeled training data $\{x^{(l)}, y^{(l)}\}_{l=1..L}$ for MT corresponds to the following: Each $x^{(l)}$ is a set of vectors, each vector $x_j^{(l)}$ representing a hypothesized translation. There are a total of L input sentences, thus L sets of hypothesized translations. Labels $y^{(l)}$ can be thought of as a vector of fluency/adequacy judgments, which are determined by a comparing the translation to a human-generated translation. The unlabeled data $\{x^{(u)}\}_{u=1..U}$ refers to sets of translations from input sentences that have no correlating human-generated translations.

A third example is protein structure prediction in computational biology. The goal is to predict 3-D structure given an amino acid sequence. The ranking problem is to sort a set of candidate 3-D structures (generated by different techniques) such that the ones most likely to be correct are at the top of the list. The setup is in many ways similar to the machine translation problem. The labeled training data $\{x^{(l)}, y^{(l)}\}_{l=1..L}$ corresponds to the following: Each $x^{(l)}$ is a set of vectors, each vector $x_j^{(l)}$ representing a hypothesized 3-D structure. There are a total of L input amino acid sequences, thus L sets of hypothesized structures. Labels $y^{(l)}$ can be thought of as a vector of similarity values,

Table 1.1: Example applications and their relation to semi-supervised ranking

	Information Retrieval (IR)	Machine Translation (MT)
Goal	For a user query, return the most relevant documents	For an input sentence, return the correct translation
Where Ranking Applies	Ranking is the central operation used to sort documents, which potentially come from a large set	Ranking is used at an (optional) second-stage to re-sort a set of hypothesized translations
$\{x^{(l)}\}_{l=1..L}$	sets of documents	sets of candidate translations
$\{y^{(l)}\}_{l=1..L}$	sets of relevance judgments for each document as determined by a human annotator	sets of fluency/adequacy ratings for each translation, computed by matching with human-generated translations
$\{x^{(u)}\}_{u=1..U}$	sets of documents without relevance judgments	sets of translations without the associated human translation

which are determined by a comparing the hypothesized structures to a true reference structure. The unlabeled data $\{x^{(u)}\}_{u=1..U}$ refers to sets of structures that have no true 3-D reference.

A summary of the information retrieval and machine translation applications is summarized in Table 1.3.

1.4 Contributions

Our answers to the questions posed at the beginning of this chapter (Section 1.1) are briefly summarized below:

1. What information in unlabeled samples can be exploited in the context of ranking problems?

   Answer: We demonstrate that some of the assumptions from semi-supervised classification can apply to ranking. In particular: the manifold assumption, low density separation (cluster) assumption, and change of representation assumption can be exploited in certain dataset scenarios. We also introduce a new assumption based on domain adaptation for ranking.

2. Is there an effective mechanism for adapting the wide range of methods developed for semi-supervised classification into semi-supervised ranking?

   Answer: We propose a local/transductive meta-algorithm which trains a ranker for each test point individually. This makes it straightforward to incorporate semi-supervised classification assumptions, as one does not need to take into account dependencies among different sets of objects to be ranked. Further, this has the added benefit of building test-dependent rankers, which has the potential to improve over general-purpose rankers.

   In addition, we develop a kernel defined on lists (as opposed to points), which allows one to modify kernel-based or graph-based classification methods for ranking.

3. What can we learn by comparing the same semi-supervised ranking algorithm on different kinds of real-world datasets?

   Answer: We experiment with six real-world datasets: three are in Information Retrieval, two are in Machine Translation, and one in Computational Biology. As expected, most methods show mixed results, since each of the dataset has different characteristics. Some issues that influence what method works include: (a) the amount of tied ranks, (b) the correlation between the optimized loss function and the true application-specific loss. We also observe that some methods tend to give slight improvements to all datasets, while other methods are high-risk/high-reward. A concise summary of the results is in the Conclusion section, while detailed analysis about why something worked (and did not work) are in the respective experiments sections.

1.5 Outline of Thesis

- Chapter 2: reviews related work, such as semi-supervised classification and supervised ranking.

- Chapter 3: describes the three tasks (and corresponding datasets) used in this work (Information Retrieval, Machine Translation, Protein Prediction).

- Chapter 4: presents a general local/transductive framework for examining various assumptions in semi-supervised learning in ranking.
- Chapter 5: investigates how unlabeled data can be used to learn better features for ranking.
- Chapter 6: investigates how unlabeled data can be used to match the training distribution to the test distribution in order to improve ranking.
- Chapter 7: investigates whether the low density separation assumption in semi-supervised classification can be applied to ranking.
- Chapter 8: introduces a novel kernel based on lists and its application in semi-supervised ranking.
- Chapter 9: compares all presented methods and summarizes the main contributions of this work.

Chapter 2
RELATED WORK

2.1 *Supervised Learning for Ranking*

A variety of approaches have been explored for the ranking problem in the supervised learning setting. The majority of algorithms can be seen as instantiations of the following abstraction, which I call the *score-and-sort approach*:

1. Learn a function f that maps each individual object $x_n^{(s)}$ to a real number, a *score*.

2. Rank a set of objects $\{x_n^{(s)}\}$ by ordering the scores from maximum to minimum. An object with the maximum score will be ranked first, followed by an object with the next largest score, etc. In other words, the ranking model $h : X \rightarrow Y$ is equivalent to `argsort` $f(x_n^{(s)})$.

This can be contrasted with the structured learning approach which directly estimates $h : X \rightarrow Y$, i.e. scores are given for entire permutations, rather than on individual objects. In this case, ranking is similar to a structured prediction problem [100]:

1. Define $h(x^{(s)}) = \arg\max_{y \in Y} g(x^{(s)}, y)$

2. Estimate g by minimizing a rank-based loss, e.g. $\sum_{l=1}^{L} l_{total}(y^{(l)}, h(x^{(s)}))$

The *score-and-sort* approaches have been investigated to a greater extent because it is often difficult to directly optimize the ranking loss function on the space of all permutation of orderings (For N_l objects, the $\arg\max$ in the above formulation needs to search over $N_l!$ orderings). The challenge, however, is to design an algorithm for learning f such that the argsort thereafter induces an ordering with minimal loss. There are roughly three categories of methods for learning f: point-wise methods (regression), pairwise methods, and list-wise methods.

2.1.1 Point-wise methods (Regression-based methods)

In regression-based methods [45], each object 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 advantage of regression-based methods is that the large body of work on regression can be exploited for ranking. However, the disadvantage is that predicting a target score for each object may be a harder problem than simply ordering the objects. In addition, like the pairwise methods, there is no guarantee that optimizing for regression loss will optimize for l_{total} or l_{top}. Yet, recently [49] has shown promising asymptotic results that in the limit of large samples, regression can optimize loss functions such as l_{top}.

Regression-based methods may be most suitable in cases where a meaningful scoring function exists. For instance, in protein structure prediction [55, 124, 159], the target score quantifies the quality of protein fold. To predict these scores, [124] proposed a modification of support vector regression that gives smaller slack (ε-tube) for top-ranked objects than lower-ranked objects, which ensures that the scores of the top-ranked objects are predicted with higher accuracy. The automatic metrics used for speech recognition and machine translation may also serve as meaningful targets for regression.

A related approach is ordinal regression [113], where one attempts to predict the ordinal numbers, which may directly represent the ranks of each object. [131] provides a framework for large margin ordinal regression applied to ranking.

2.1.2 Pairwise preference methods

The idea of pairwise preference methods is to learn f such that $f(x_i^{(l)}) > f(x_j^{(l)})$ if $x_i^{(l)} \rhd x_j^{(l)}$. Even if the labels $\{y^{(l)}\}$ are given as total orderings, pairwise preference methods would nevertheless extract and learn from the corresponding set of pairwise orderings (for a total ordering of N_l objects, there are $(N_l)(N_l-1)/2$ such pairwise preferences).

The advantages of this pairwise preference approach are (1) existing classification methods, with some modifications, can be applied, and (2) it works on partial orderings and can be used on applications where total ordering labels are difficult to obtain. The main disadvantage is that the learning objective is more naturally cast as minimizing the number of incorrect pairwise orderings, which may not correspond to the true loss function on total orderings. Other disadvantages include the i.i.d. assumption of different pairs and the computational complexity arising from generating all pairwise preferences, but these issues can be solved to some extent, by e.g. placing more emphasis on some pairs over others (c.f. [31] [84]).

One of the first pairwise preference approaches is RankBoost [58]. RankBoost maintains weights on each pair of objects and learns weak rankers that reduce the number of incorrect pairwise orderings. Following the boosting philosophy, a weak ranker is learned in each round and the final ranker is a combination of weak rankers. Similar to RankBoost, RankSVM [67, 82] attempts to minimize the number of incorrect pairwise orderings by formulating $(f(x_i^{(l)}) > f(x_j^{(l)})$ for all pairs $x_i^{(l)} \rhd x_j^{(l)})$ as constraints in a support vector machine objective. RankSVM minimizes the hinge loss over the margin $f(x_i^{(l)}) - f(x_j^{(l)})$ of incorrectly ordered pairs. The idea of pairwise preferences is given a probabilistic formulation in RankNet [27]. The probability that $x_i^{(l)} \rhd x_j^{(l)}$ is defined as: $P_{ij} = \frac{e^{o_{ij}}}{1+e^{o_{ij}}}$ where $o_{ij} = f(x_i^{(l)}) - f(x_j^{(l)})$, and a neural network is trained to optimize the cross-entropy between the desired P_{ij} and the predicted P_{ij}. Much recent work in this area has focused on improving the above algorithms and attempting to optimize an objective that is closer to the true loss function (e.g. [26, 147, 128, 52]).

Many applications that use the loss function l_{top} may also learn the scoring function from a pairwise preference method [42]. Rather than generating all possible pairwise preferences from a total ordering, these methods only ensure that pairwise preferences between the top-ranked object and other objects are predicted correctly. In other words, we estimate f such that the score of the top-ranked object is higher than any other object, but do not care about the ordering among non-top-ranked objects, i.e. $f(x_{i'}^{(l)}) > f(x_j^{(l)})$ for i' indexing the top-ranked object and $j \neq i'$. This line of work has an interesting connection with structured prediction [144, 148], since the argsort function of $h : X \to Y$ essentially becomes an argmax. Most re-rankers in machine translation and other natural language processing systems employ this argmax approach. Prominent examples include: parsing [43, 92], machine translation [133], part-of-speech tagging [74], information extraction [78].

One thing to note about pairwise methods such as RankBoost is that although the training phase uses pairs of objects, the testing phase operates on individual objects. That is, $f(\cdot)$ is learned by comparing pairs of objects (i.e. $f(x_{i'}^{(l)}) > f(x_j^{(l)})$) but during prediction, we apply $f(\cdot)$ on individual objects independently, then sort by the resulting values. As such, there are no issues of intransitivity for score-and-sort pairwise methods. (Intransitivity occurs if we make independent pairwise decisions during test time, i.e. $A > B$, $B > C$, and $C > A$, which leads to an inconsistent ordering.) A few alternative methods (not in the score-and-sort approach) operate on pairs at test time and therefore is required to solve the challenge of combining partial (pairwise) orderings into a total order. For instance, [41] proposes a greedy method where one first builds a directed weighted graph, where each vertex is an object and each edge indicates the strength in which the starting node ranks over the ending node. Then they compute the potential for each node by the weighted sum of outgoing edges minus weighted sum of incoming edges. The node with largest potential ranks first, is deleted from the graph, and we recurse to obtain the second-ranked node. Another example is [3], which advocates using the QuickSort algorithm to combine partial orderings during test time.

2.1.3 Listwise methods

The third class of ranking methods is called list-wise approaches, due to the emphasis on treating the list as the basic object of optimization. This avoids the problems in regression and pairwise approaches, which artificially forces independence assumptions among objects in the same list. Listwise approaches can use information about rank positions and information at the query level. It can potentially optimize a loss function that more closely approximates the true loss function, but the cost is usually more intensive (sometimes intractable) computation.

Listwise approaches can generally be divided into two categories. The first directly optimizes the loss function one cares about, or some smoothed version thereof. Loss functions for ranking are usually non-smooth and non-differentiable, which is a considerable challenge to the optimization. Examples include [115, 158, 163]. The other approach defines a loss function on the list, but the loss function may not necessarily be inspired by the loss function used for evaluation. For example, [156, 32] define probabilistic permutation models based on the Luce-Plackett model and training involves optimizing the model likelihood (or minimizing KL-divergence).

2.2 Semi-supervised Learning for Classification

A wide variety of techniques have been proposed for semi-supervised learning in the *classification* literature. See [170] for a concise and updated survey. Here we group the various techniques based on the assumptions used. Each technique makes different assumptions on how unlabeled data can help learning. The four broad assumptions are:

Bootstrapping: Assume that the predicted labels of unlabeled data can be used for learning. Methods such as self-training [162, 1], co-training [21], and mixture models with EM [118, 34] fall into this class.

Low Density Separation: 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 [15, 81, 61] (also known as semi-supervised SVMs) achieve this by forcing a large distance between unlabeled samples and the decision boundary. The assumption used by these methods is sometimes also called the "cluster assumption."

Manifold/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. 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.

One important note is that there is no clear-cut way of categorizing the various algorithms by their assumptions, since many of the assumptions are related and many algorithms employ more than one assumption. For example, one may also think of graph-based methods as falling under the Low Density Separation assumption or the Change of Representation assumption.

Since these assumptions are very relevant to this work, we describe their related work in much more detail as follows.

2.2.1 Bootstrapping Assumption

The Bootstrapping Assumption assumes that the predicted labels of unlabeled data can be used for supervised learning. Techniques that assume this include: self-training [162, 1], co-training [21], and generative models with EM [118, 34].

In self-training, first an initial classifier trained on small amounts of labeled data predicts the labels of unlabeled data. Then, confident predictions are added into the training set, and the classifier is re-trained. Self-training assumes that the additional labels are accurately predicted; its accuracy degrades when noisy labels are added to the training set.

In co-training [21], two classifiers are trained on different feature splits of the labeled data. Then the classifiers teach each other about their respective high-confidence predictions on unlabeled data (i.e. confident predictions by classifier A are added to the training set of classifier B, and vice versa). Theoretical and experimental results show that co-training performs well when feature splits are sufficiently good and are conditionally independent of each other given the class [21, 117]. In general, co-training can be seen a method that enforces multiple classifiers to agree on both the labeled and unlabeled data. It works because the version space is reduced when classifiers are forced to agree on the large unlabeled data.

Generative models with the EM algorithm [53] can be seen as a soft version of self-training. They model the joint distribution $p(x,y) = p(y)p(x|y)$ where $p(x|y)$ is a mixture component that can be identified by large amounts of unlabeled data. [118] uses multinomial mixture components for semi-supervised text classification. Castelli and Cover [33, 34] proved that if the model form is correct, unlabeled data is guaranteed to improve accuracy. In practice, unlabeled data are often downweighted [30, 46] in generative models to prevent excessive bias. Fujino et. al. [60] derives a hybrid algorithm that attempts to correct the bias. Generative models with EM are also subject to the difficulty of getting stuck in local optima.

2.2.2 Low Density Separation Assumption

The Low Density Separation Assumption assumes that the classification boundary exists in low density regions of the feature space, and that unlabeled samples can help identify such regions. Such an assumption is reasonable if one assumes that positive and negative samples form two separable clusters (i.e. the so-called *cluster assumption*).

Transductive SVMs (TSVMs), also known as semi-supervised SVMs (S3VMs), [15, 81, 61] achieve low density separation by maximizing the margin of both labeled and unlabeled data. Zhang and Oles [166] questions the notion of margin for unlabeled samples, and suggests that TSVMs may "maximize the wrong margin." Nevertheless, much research has focused on TSVM's difficult discrete optimization problem by methods such as gradient descent on an approximate continuous objective [37], deterministic annealing [135], and the concave-convex procedure (CCCP) [44]. Importantly, [35] applied a Branch-and-Bound optimization procedure to obtain exact global optima on small datasets. Their excellent results, which outperformed other TSVM implementations and some graph-based algorithms, validated the importance of a good optimization procedure for the TSVM objective. [160] proposed an alternative SVM formulation based on semi-definite programming (SDP); their formulation allows for both binary and multi-class problems in semi-supervised and unsupervised learning.

Other techniques that employ the Low Density Separation assumption include Gaussian Processes with null category noise model [99], information regularization [142, 47, 48], entropy minimization [64, 80, 102], and maximum entropy discrimination [76].

2.2.3 Manifold Assumption

In graph-based methods, one first constructs a graph over both labeled and unlabeled data; then a function that is both smooth over the graph and incurs small loss on the labeled samples is estimated. The graph can be seen as a data-dependent regularizer.

This can also be considered as similar to a Change of Representation Assumption because the distances in the original Euclidean feature space are discarded in favor for the geodesic distance induced by the graph. This geodesic distance is assumed to be more accurate since the large amount of unlabeled data can help induce the true underlying subspace or manifold of the data. To make this clearer, imagine we have 10 labeled points in a feature space of dimension 1000. This is a high dimensional space, and it is likely that we would suffer from overfitting. Now suppose we have many more unlabeled points, which we use to generate a data graph–the edge weights of a node, for example, can be computed from the Euclidean distances of its closest neighbors. Therefore, the distance between two faraway points is no longer the Euclidean distance computed directly on its feature vectors, but instead the summed distance of traveling through the paths of nearest neighbors in the graph. If it turns out that the real data lies in the manifold and not the original feature space, then the geodesic distances would be a more accurate distance measure and the graph-based method may achieve improvements.

The variety of graph-based semi-supervised algorithms differ primarily in the particular form of loss function and regularizer. Prominent examples include: Mincut [19, 20], Spectral Graph Transducer [83], Discrete Markov Random Fields (MRF) [171], and its continuous relaxation: Gaussian Random Fields and Harmonic Functions [172], Manifold Regularization [12, 13], and Graph Kernels [138, 91, 95, 5]. An open area here is the question of optimal graph construction, since empirical evidence suggests that accuracy may depend more on the graph than the particular learning algorithm. Some works have begun to address this, e.g. modifying graph spectrum [173, 85], convex combination of graphs [8], and classifier-derived distances [4].

2.2.4 Change of Representation Assumption

The Change of Representation Assumption assumes that a better representation (e.g. more parsimonious or expressive) for learning exists, and that unlabeled data can help discover this hidden representation.

The idea of feature/kernel learning is to use vast amounts of unlabeled data to learn a better feature or kernel representation of the data. The new feature or kernel is assumed to be a better distance measure, just as the geodesic distance is assumed to be better when the data lies on a manifold. Feature/kernel learning methods differ from graph-based methods in terms of the emphasis: whereas graph-based methods focus on ways to exploit unlabeled data once a graph is constructed, feature/kernel learning methods focus on learning a better distance metric, which *could* be used to construct graphs [4]. However, the strongest advantage of feature/kernel learning is that one is not restricted to graphs but is free to choose from the toolkit of any supervised and inductive classification algorithm once the new feature/kernel is learned. It is essentially a two-step procedure:

1. Learn a better feature/kernel representation using both labeled and unlabeled data

2. Apply supervised learning to the new feature/kernel representation of labeled data

How does one learn better features from unlabeled data? One approach is to cluster the samples and use the cluster identities as new features [105]. Alternatively, one may learn dependencies between the original features and collapse them into more parsimonious latent variables. Works by

Ando and Blitzer [6, 7, 18] use multiple-task learning to find the dependent features; [121] learns the latent variables via principal components analysis or independent components analysis. An alternative to learning better features is to learn distances or kernels between points directly (since many learning algorithms work by comparing distances, even if the initial representative is based on features). Methods for learning better kernels include Fisher kernels [75, 70, 63] and cluster kernels [36]. There are also many distance metric learning algorithms (e.g. [96, 69]); the application of them to this problem is still relatively unexplored. An open research area is the question of how to use the labels as well as the unlabeled data to learn better features; the above techniques essentially ignore the labels and learn features/kernels in an unsupervised fashion.

Finally, we note that *theory* for semi-supervised classification is still an open problem, and has been identified as one of the ongoing challenges in machine learning [93]. One of the first works in this area was [34], which established that if we view classification as a mixture of class conditional distributions and that the mixtures are identifiable (cluster assumption), then error can converge exponentially fast in the number of labeled examples. For co-training, [103] presents a generalization error bound which shows that forcing agreement among multiple learners lead to tighter bounds. Importantly, [11] formulates a PAC model for unlabeled data, which is the first unified theory for semi-supervised learning (as opposed to algorithm-specific analyses). More recently, [94] analyzed the Manifold Assumption and indicate that some methods based on graph Laplacians actually do not achieve faster convergence rates. Singh et. al. [137] provide an interesting analysis of when unlabeled data helps under the cluster assumption. Theory for semi-supervised classification is still an unsolved problem; we will not address these issues in this work.

2.3 Semi-supervised learning for Ranking

There are generally two interpretations of "learning to rank with partially-labeled data." For concreteness, in this section we will describe these interpretations as information retrieval problems, where the objects to be ranked are documents **d**.

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 **d** is only partially-labeled, i.e. some documents in **d** have relevance judgments, while other documents in the *same* list **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 [82] 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 2.1 for a pictorial comparison.

Several methods have been proposed for the *Missing Labels* problem, e.g. [168, 153, 66, 151]: the main idea 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 [168] (transductive), or one can train a ranking function using these values as true labels [66, 151] (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 objects, which acts as a regularizer to ensure that the predicted values are similar for closely-connected objects. Specifically, for a combined training and testing set of n objects, one can learn a score vector $\mathbf{f} \in R^n$ that represents the score value for each object. One minimizes the following objective function with respect to $\mathbf{f}$:

$$\min_{\mathbf{f} \in R^n} \mathbf{f}^T \mathbf{L} \mathbf{f} + C \sum_{(i,j) \in E} \tau(x_i, x_j) \xi_{ij} \qquad (2.1)$$
$$\text{subject to} f_i - f_j \geq \xi_{ij}, \quad \xi_{ij} \geq 0 \quad \forall (x_i, x_j) \in E$$

where $(i, j) \in E$ is the set of labeled pairwise preferences, $\mathbf{L}$ is the Laplacian of the data graph consisting of all objects as nodes, τ is the loss for misordering x_i and x_j, ξ are slack variables, and C is an adjustable parameter that trades off between loss and regularization.

[40] also proposes a graph-based regularization term, but in contrast to Equation 2.1, it defines the graph nodes not as objects, but as *object 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 objects in a large unlabeled dataset quickly leads to intractable graphs.

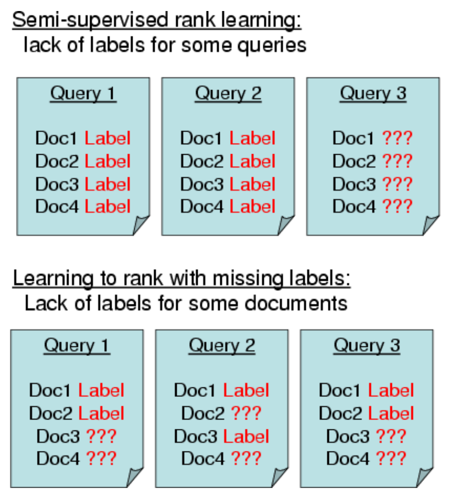


Figure 2.1: Two partially-labeled data problems in ranking. We focus here on semi-supervised rank learning, where labels are entirely lacking for some queries. A different problem is that of "missing labels", where not all documents retrieved by a query are labeled. Note that these two problems are not mutually-exclusive.

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, [151] 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, [145] 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 previous work [54] proposed another method using the Change of Representation assumption.

The *Semi-supervised Rank Learning* problem is important in practice because it may be difficult to obtain any labels for some queries/lists. For example, in information retrieval, this problem manifests itself in the long tail of search queries. A quote by Udi Manber (Google Vice President of

Engineering) demonstrates how many types of queries are received at Google: "Twenty to twenty-five percent of the queries we will see today [at Google], we have never seen before." [1] It is therefore impractical to obtain labels for many of the queries in practice. For machine translation and protein structure prediction, the *Semi-supervised Rank Learning* problem is actually the only scenario that occurs. In these applications, a one-time sunk cost is associated with obtaining a reference translation or 3-D structure. Once this reference is obtained, labels for each hypothesis translation or candidate structure is computed via automatic procedures. Therefore, the additional cost

2.4 Relations to Domain Adaptation

Domain adaptation (c.f. [79] 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 [14, 125] applied the Change of Representation idea from semi-supervised learning to domain adaptation. Our work in Chapter 6 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.

One of the popular approaches in domain adaptation is importance weighting [134], 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

[1] He made this statement at Google Searchology conference, May 2007. A video of the event is available online at `http://www.youtube.com/watch?v=SD0cyYUEE1Y`; his talk is roughly 35 minutes into the 53 minute video.

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 $\{(x_u)\}_{u=1..U}$. 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 [134] 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 $\hat{p}_{test}(\cdot)$) [17, 73, 141]. 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 [155, 38]. However, their methods are under the supervised adaptation framework, and therefore are not directly applicable to the transductive problem we are interested in here.

2.5 Related Work in Statistics and Economics

There is a large body of literature in statistics on modeling rank data, as well as in economics on modeling choice decisions. These are related to the ranking problem, and insights from these fields could potentially benefit our machine learning techniques. We do not directly apply any of the statistics or economic work here, but for completeness we will briefly describe some background in these areas. These may be useful in identifying possible avenues of interdisciplinary work.

One of the main goals of statistics is to analyze, model, and interpret data. Ranking data is a kind of data that often results from surveys, for example. Surveys could be given to a large population, and insights on what the population prefers could be inferred and hypotheses about population preferences could be tested. For instance, suppose a survey asks a group of students to give their preference for a set of drinks: (Coke, Pepsi, Sprite)[2]. There are 6 possible rank vectors: (3,2,1), (3,1,2), (2,3,1), (2,1,3), (1,3,2), (1,2,3). Based on the population survey (size N), we obtain a histogram over these 6 possible rank vectors. The questions that statistics seek to answer are, e.g.:

1. Is there an average (mean/median) rank vector that describes the average preference? If so,

[2] Example taken from [111]

how to compute it?

2. What is the spread of rank vectors to the average (i.e. variance)?

3. Are there clusters of sub-populations that show different kinds of preferences?

4. How to model the population, and how to fit the model to data?

The average rank vector $\bar{y}$ could be obtained by minimizing, e.g. the following objective:

$$\arg\min_{\bar{y}} \sum_{i=1}^{N} d(y^{(i)}, \bar{y}) \tag{2.2}$$

where d is a distance between two rank vectors. Possible distances include:

- Spearman, Footrule, and Hamming: $d(y,\bar{y}) = \sum_d |y_d^p - \bar{y}_d^p|$ where p$\rightarrow$ 0 (Hamming), p=1 (Spearman) or p=2 (Footrule)

- Kendall: $d =$ miniminum number of consecutive swaps required to make y become $\bar{y}$.

- Cayley: $d =$ miniminum number of arbitrary swaps required to make y become $\bar{y}$.

There are many ways to model the data with a probabilistic model. To give a flavor, here are some examples:

- Mallows ϕ model: $P(y|\bar{y},\sigma) = \frac{1}{Z(\sigma)} \exp(-\sigma d(y,\bar{y}))$. This is similar to a Gaussian model, where we have a "mean" vector $\bar{y}$ and a constant σ indicating the spread around the mean. The farther a rank vector y is from the mean (in the $d()$ sense), the lower the probability.

- Babington Smith models: Define p_{ij} to be the probability that object i is ranked above object j, independent of other potential choices.[3]. The probability model is then obtained by ranking a set of objects constrained to the orderings where all p_{ij}'s are consistent.

[3]The Bradley-Terry model, which is an instance of this, assumes $p_{ij} = \frac{v_i}{v_i + v_j}$, where $v > 0$.

- Multi-stage models (e.g. Plackett-Luce model): Define μ_i to be the probability that object i is the first choice out of all candidates. Then we can imagine a multi-stage process where a ranking is obtained by drawing the first choice, then deleting the first choice and drawing the second choice, etc. We may wish the values μ_i to satisfy some conditions such as the Luce Axioms.[4]

For a broad review of ranking models, refer to [57, 111]. One important note about these ranking models is that they often work with a fixed set of objects. As such, analysis of ranked data is often termed "analysis of permutations." For example, in our survey, each participant produces a preference over the same set of objects (different participants give a different permutation over the same set). In contrast, in the ranking problems we are interested, different "participants" may be producing preferences for different sets of objects. For example, in Information Retrieval, each list of documents represent *different* documents retrieved via different queries. While this might imply that prior work on permutations may not always be directly applicable, we already begin to see recent innovations in machine learning which utilize these ideas (c.f. [28, 101, 72]).

Related work in economics include social choice theory, utility theory, and game theory. All these fields are concerned with "making choices" out of a set of candidates, and one way this decision manifests itself is via a preference relation. Social choice theory [9] studies how preferences can be aggregated to make a collective decision. Utility theory (c.f. [109]) is a branch of decision theory and game theory which models choice and preference. Each object has an utility, and the preference relation is induced from the ordering of these utility values. This is very similar to the score-and-sort approach of ranking. In fact, we may consider many of the methods for ranking as procedures for estimating the utility of each object.

Chapter Summary

We reviewed various work related to semi-supervised ranking, in particular semi-supervised classification and supervised ranking. While there are much work in these related fields, semi-supervised ranking is still a relatively unexplored area. Here we briefly identify some potential challenges and

[4]Essentially, these axioms formalize intuitions such as: if the probability that Coke is the first choice out of all 3 drinks is 0.9, then the probability that Coke is the first choice out of (Coke,Pepsi) should be no less than 0.9.

open problems in semi-supervised ranking:

- What kinds of unlabeled data assumptions in classification can be converted to ranking? Classification algorithms can be converted to ranking algorithms via, e.g. pairwise formulation, but would the unlabeled assumptions be convertible as well?

- What new algorithmic and computational challenges are introduced by the ranking structure? How can operations (e.g. loss function, kernels) on samples be extended to operations on lists?

- What sorts of theoretical guarantees are possible with semi-supervised ranking?

The first two challenges will be addressed in this dissertation.

Chapter 3
APPLICATIONS AND DATASETS

This thesis focuses on three applications: Information Retrieval, Machine Translation, and Computational Biology. The idea is to utilize a variety of real-world datasets to test the effectiveness of my algorithms. An algorithm may work well on one dataset but not the other, and investigation into the reasons for such differences may lead to further insight about the algorithm and its assumptions.

3.1 Ranking in Information Retrieval

The goal of Information Retrieval (IR) is to help a user find the information he/she desires. The user's need can be very diverse. For example, in web search, the user's need can be classified as informational, navigational, and transactional [25]. The first, informational query, may range from simple factoid questions such as "What is the highest mountain in the world?" to complex ones such as "How have different parts of the world responded to the President's new plan for Afghanistan". Users with navigational needs are interested in going to particular webpages on the web (e.g. the query "NAACL 2009" may indicate a need for the user to find a conference website, which will eventually help in satisfying other needs. Transactional needs represent a user's wish to accomplish some action on the web, such as the "Form 1040 download".

The IR field is therefore very broad, encompassing studies of human interaction, document representation, scalable architectures, etc. (Refer to a recent book by [110] for an overview). Ranking is one of the core areas of study in IR due to its importance in presenting documents to users in an efficient manner. Information is essential to making knowledgeable decisions, but information is useless if the one who needs it cannot find it.

Ranking in IR is classified into two main categories: Static Ranking and Dynamic Ranking. Suppose we have a large but fixed and finite document collection. Static ranking is the problem of ordering documents without regard to a specific user query. This ordering can represent the a priori "quality" of a document; high quality documents should have a prior probability of ranking higher than a low quality document, independent of any user query. For example, Google's famous PageRank algorithm [24] assumes that webpages with many in-links are likely to be of much higher quality than those with few in-links, i.e. $p(d) = (1-\alpha) + \alpha(\sum_{d' \in Incoming} \frac{p(d')}{c(d')})$. Here, $p(d)$ is the PageRank of document d (the higher the better), $c(d')$ is the number of out-links for document d', and α is a damping factor that is useful for computational issues and modeling a random surfer. We see here that this recursive equation generates a static ranking of all webpages on the web, where a webpage that is often linked by others with high pagerank is itself higher in rank. Importantly, note that this ranking is only a function of the underlying web graph and is independent of any user query.

In contrast, Dynamic ranking is the problem of ordering documents *after* a user-given query has been observed, i.e. a user need has been stated. A large class of dynamic 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.

This thesis focuses on the dynamic ranking problem, where features derived from static ranking may be employed.

Type of feature	Examples
Document characteristics	document length
Term matching of query and web document	BM25[126], LMIR[165], tfidf
Term matching of query and metadata (anchor, title, URL)	BM25, LMIR, tfidf
Static ranking	PageRank[24], HITS[87]
Hyperlink-based scores	Hyper-link feature propagation[123]

Table 3.1: Examples of TREC features

3.1.1 Information Retrieval Datasets

Our IR experiments are performed on the LETOR dataset (version 2) [107], which contains three sets of document retrieval data: TREC'03, TREC'04, and OHSUMED. This is a dynamic 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. This is a commonly-used benchmark task in IR.

The TREC data is a Web Track Topic Distillation task [50]. The goal is to find webpages that are good entry points to the query topic. Figure 3.1 gives an example of a query and a subset of webpages that ought to be returned by the ranker. As seen in the example, webpages such as "USDA Cotton Program" provide a good entry point reference for a variety of information that will answer the query; the webpage itself may not contain much information, but should contain pointers to information. The original data consists of all webpages from a crawl of the `.gov` domain in 2004. There are a total of 1,053,110 HTML webpages and 11,164,829 hyperlinks. The LETOR dataset conveniently extracts many state-of-the-art features from query-document pairs, including BM25 [126], HITS [87], and Language Model [165], and hyperlink propagation [123]. A summary of features is given in Table 3.1.

The OHSUMED data [68] consists of medical publications and the queries represent medical search needs. It is a clinically-oriented MEDLINE subset, consisting of 348,566 references (out of a total of over 7 million), covering all references from 270 medical journals over a five-year period (1987-1991). An example query and relevant document is shown in Figure 3.2. Note that the document fields are the title, abstract, MeSH indexing terms, author, source, and publication type.

QUERY:

Title: cotton industry

Description: Where can I find information about growing, harvesting cotton and turning it into cloth?

EXAMPLE ANSWERS/HOMEPAGES

- Cotton Pathology Research Unit (cpru.usda.gov/)
- FAS Cotton Group (ffas.usda.gov/cots/cotton.html)
- Office of Textiles and Apparel (otexa.ita.doc.gov/)
- USDA Cotton Program (www.ams.usda.gov/cotton/)

Figure 3.1: Example TREC query and webpages

The features are in general of the term-matching type, ie. BM25, LMIR, and tfidf.

For TREC, documents are labeled {`relevant`,`irrelevant`}; an additional label {`partially relevant`} is provided for OHSUMED. Table 3.2 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 [107] for details.

3.1.2 *Information Retrieval Evaluation*

The evaluation metrics are mean average precision (MAP) and normalized discount cumulative gain (NDCG@n) [77]. 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:

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

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

QUERY:

Patient Info: 60 year old menopausal woman without hormone replacement therapy

Information request: Are there adverse effects on lipids when progesterone is given with estrogen replacement therapy

---

DOCUMENT:

Source: J Obstet Gynaecol 8707; 94(2):130-5

MeSH Keywords: Drug Combinations; Estrogens/AE/*TU; Female;Hemorrhage/CI; ...

Title: Continuous oestrogen-progestogen treatment and serum lipoproteins in postmenopausal women.

Publication type: JOURNAL ARTICLE.

Abstract: Serum lipids and lipoproteins were examined in 44 healthy postmenopausal women every 3 months during 1 year of treatment with either continuous oestrogen-norethisterone acetate or placebo. Total serum cholesterol and LDL-cholesterol levels were reduced by approximately 15% and 20% (P less than 0.001), respectively in the hormone group but were unchanged in the placebo group.... Moreover, the low frequency of bleeding with continuous oestrogen-proge stogen therapy would make this an appropriate alternative in postmenopausal replacement therapy.

Author: Jensen J; Riis BJ; Strom V; Christiansen C.

Figure 3.2: Example OHSUMED query and document

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

Table 3.2: IR Data characteristics

	TREC'03	TREC'04	OHSUMED
#queries	50	75	106
#documents	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
#features (original)	44	44	25

is a normalization constant that represents the score of the best possible ranking and allows NDCG to be bounded by $[0,1]$.

As reported by [107], some of the state-of-the-art results based on RankBoost and RankSVM are as follows:

- TREC'03: RankBoost - 0.212 MAP, RankSVM - 0.256 MAP
- TREC'04: RankBoost - 0.383 MAP, RankSVM - 0.350 MAP
- OHSUMED: RankBoost - 0.440 MAP, RankSVM - 0.447 MAP

3.2 Ranking in Machine Translation

In machine translation systems, ranking algorithms are used as a "2nd-pass" decision maker that improves upon the outputs of a "1st-pass" translation system. The 1st-pass system, due to constraints in computing power, usually employs simpler features and knowledge sources when translating foreign sentences. For each input sentence, it generates a set of likely hypotheses (N-best lists), which is then re-ordered by a ranking algorithm that employs more powerful features or knowledge sources. In a variant of this setup called "system combination", multiple 1st-pass systems are simultaneously employed to generate multiple N-best lists, which are then combined and ranked. This coarse-to-fine approach of ranking a hypothesis set generated by the 1st pass system can be very effective in improving system accuracy. The assumption is that good translations lie in the set of the hypotheses, and can be picked out by powerful ranking algorithms.

In machine translation, the ultimate output of the entire system is a single translation, so the more specific loss function l_{top} that only measures the quality of the top hypothesis is used. In addition, the labels $\{y^{(l)}\}$ are usually total orderings. Automatic metrics, which measure the distance between translations and their corresponding human-produced references, provide an effective way to label each hypothesis with a score. These scores (e.g. the BLEU metric for machine translation) can be sorted to provide a total ordering. Further, multiple hypotheses may have the same loss, so in practice we have ties. Finally, machine translation systems tend to have low-dimensional feature vectors for representing hypotheses–this in turn impacts issues such as separability of data. We note that there are research systems that operate on hundreds or thousands of features, but the conventional systems usually work with on the order of 10 features. The features, which include likelihood scores from language models, translation models, acoustic models, etc., are sometimes called "sub-models" because they are each quite complex functions and encompass a significant amount of information.

3.2.1 Machine Translation Datasets

We employ two machine translation datasets from the International Workshop on Spoken Language Translation (IWSLT) shared task of 2007. One dataset is for Arabic-English translation, while the other is for Italian-English translation. IWSLT evaluations, in general, focus on translation of spontaneous speech. The Arabic-English task involves translation of transcribed read speech of the Basic Travel Expression Corpus (BTEC) [143], which contain sentences similar to those found in a traveler's phrase book. The training data for the Italian-English task consisted of transcriptions of read dialogs, whereas the test data contains spontaneous dialogs (e.g., between a travel agent and his clients regarding transactions about ticket purchases and hotel reservations), so there is a data mismatch condition in the Italian-English case.

The baseline systems used for this task are described in the System Description paper of University of Washington's entry in IWSLT [86]. The work in this thesis is based on the University of Washington system, so note that the following brief descriptions are related to the system and do not represent the *only* approach for machine translation. Nevertheless, the University of Washington system is a mainstream and competitive system in the framework of phrase-based statistical machine translation. For a good recent survey of various approaches to machine translation, see [108].

The first-pass machine translation system is a statistical phrase-based MT system based on a log-linear probability model:

$$e* = \arg\max_e p(e|f) = \arg\max_e \{\sum_{k=1}^{K} \lambda_k \phi_k(e,f)\} \tag{3.1}$$

where e is the English sentence, f is the foreign (Italian or Arabic) sentence, λ_k are k weights to be trained, and $\phi_k(e,f)$ are features such as phrase-based translation scores, lexical translation scores, word count penalty, and language model score.

The weights are trained by Minimum Error Rate Training (MERT) [119] and decoding is done via the Moses decoder (without factored models) [89]. The decoding generates N-best lists of size up to 2000 hypotheses, which is then de-duplicated to remove identical hypotheses. This dataset is the input to our second-pass ranker. For more details about the system, refer to the system description [86]. A brief summary of the data characteristics are shown in Table 3.3.

Table 3.3: MT Data characteristics

	Arabic-English (ar)	Italian-English (it)
#lists (dev set)	482	500
#lists (test set)	499	494
avg hypothesis per list	260	362
minimum # of hypotheses in a list	2	1
maximum # of hypotheses in a list	1676	1626
#manual references	7	1
#features (original)	9	10

3.2.2 Machine Translation Evaluation

Traditionally, MT evaluation measures required examination of results by human judges, who annotated the *adequacy* of translation in capturing the meaning and *fluency* of the expression in the target language. This is expensive, however, so the community has since moved towards automatic evaluation metrics such as word error rate BLEU [122], PER [120], TER [140], and METEOR [98]. A common element of these automatic metrics is the requirement of one or more human-generated translation of the test set, called references. The idea is to match the system output with these human references; the larger the match, the better the translation.

In this work, we use BLEU [122], the most common evaluation metric to date. BLEU considers n-gram matches of the system output with human references up to a maximum n (usually n=4, as employed in this work). This rewards sentences where the local word order matches that of the reference. BLEU is a precision-oriented measure in that it calculates the percentage of n-gram matches out of the total number of n-grams in the sentence. A brevity penalty is introduced to capture recall: it penalizes sentences that are significantly shorter than the reference. The BLEU metric is computed over the whole corpus using the following equation:

$$BLEU = BP \cdot \exp(\sum_n \log p_n) \tag{3.2}$$

where BP is the brevity penalty and p_n is the precision of each n-gram.

Traditionally, researchers report the BLEU score of the top hypotheses chosen by the ranker. In addition to this, here we report what we call *Top-k BLEU oracle scores*, which is a metric useful for MT system combination scenarios. In this scenario, the 2-nd pass ranker not only outputs the top hypothesis after re-ranking, but the Top-k hypotheses. The idea is that a downstream system combination engine will gather these k-best lists from multiple MT systems and do an additional re-ranking step. System combination has become an effective strategy in MT research due to, e.g. advances in sentence/lattice alignment (c.f. [127]), and therefore we believe Top-k oracle is a meaningful metric to report.

The Top-k BLEU oracle is best illustrated by an example. In Figure 3.3, we see seven hypothesized translations (of the Arabic-English dataset) in the order ranked by the re-ranker. If we only output a single sentence (“please press the button yellow), the BLEU score would be .53. If we instead output the top-2 hypotheses (“please press the button yellow”, “please press button yellow”), the best possible (*oracle*) BLEU score is still .53. If we output the top-4 hypotheses, the best possible BLEU score is 1.0. In this sense, the Top-k BLEU oracle represents the best possible achievable BLEU score if the Top-k list is given to a system combination engine.[1] Note that in this definition, the Top-k BLEU oracle is a non-decreasing curve with respect to *k*.

In addition to BLEU, we report PER (position-independent word error rate) for the first hypothesis in the re-ranked list.

REFERENCES

please press the yellow button

please push the yellow button

Hypotheses in ranked order and BLEU of individual hypothesis

(BLEU=.53) 1. please press the button yellow

(BLEU=.42) 2. please press button yellow

(BLEU=.41) 3. please press the button the yellow

(BLEU=1.0) 4. please press yellow button

(BLEU=.65) 5. please press the yellow the button

(BLEU=.43) 6. please press yellow the button

(BLEU=.35) 7. please press a yellow the button

Figure 3.3: Illustration of Top-k BLEU oracle score. Top-1 oracle=.53, Top-2 oracle=.53, Top-3 oracle=.54, Top-4 oracle=1.0, Top-5 oracle=1.0.

3.3 Ranking in Computational Biology (Protein Structure Prediction)

Protein structure prediction is an important research area within computational biology. The goal is to predict the 3-D structure of a protein based on sequence (e.g. amino acid) information. This

[1]This assumes the system combination is simply choosing among sentences; more advanced system combination methods merge different sentences and generate new ones, in which case the Top-k BLEU oracle becomes a lower bound on the best possible score from system combination.

problem is of significant interest because while advances in genome sequencing techniques have produced large datasets of linear amino acid sequences, full understanding of the biological role of proteins require knowledge of their structure and function [10].

Protein structure prediction methods can generally be divided into two classes. Given a sequence with unknown protein structure, *template-based modeling* works by first identifying similar sequences in a pre-established protein database (PDB), and then the predicted structure is constructed from these possibly-similar "templates" (c.f. [39]). On the other hand, in *ab initio* modeling, the 3-D structure is constructed from scratch by physical simulation and sampling the conformal space [146]. In either case, a common component of the overall system involves ranking the set of templates or samples in order to select the most promising candidates [124].

In this work, we follow the framework set up by [124]. The particular task is to select among multiple protein structure candidates that are generated from various methods. The test set involves the protein structures submitted to the the annual CASP (Critical Assessment of Methods of Protein Structure Prediction) evaluation [116]. The CASP evaluation is a community-wide effort in protein structure prediction, where different research groups around the world submit their structure predictions on a common set of yet-to-be-known proteins. For each unknown protein sequence, we have a set of candidate structures from various sites. The goal of the ranker is to choose the best among the set.

The problem setup is therefore analogous to the machine translation task. Whereas MT rankers work with a N-best list of hypothesis translations, the protein prediction ranker works with a N-best list of candidate 3-D structures. In MT, the goal is to find the best translation that matches a human reference translation. In protein prediction, the goal is to find the best structure that matches the true structure as determined by X-ray crystallography or other methods (which are usually more expensive than the computational approach).

3.3.1 Protein Structure Prediction Dataset

We use the protein structure prediction dataset provided by [124].[2] The training data consists of the predicted structures submitted to CASP5 and CASP6 evaluations. The native protein structures

[2]I am immensely grateful to Jian Qiu and Bill Noble for their numerous help on preparing this dataset.

from PDB used in the training set of [124] is not included here since our training algorithms are based on ranking.[3] The test set consists of proteins to be predicted for the CASP7 challenge.

A brief summary of data statistics is shown in Table 3.4. As seen, there are 99 proteins to be predicted in the test set, each with around 211-267 candidate structures to choose from. The number of candidate structures per list is more for the test set because the test set includes the top five models submitted by site participants, whereas the training data only contains the top model. For details, refer to [124].

Table 3.4: Protein Prediction Data characteristics

	TRAINING SET	TEST SET
#lists, i.e. number of proteins to predict	73	99
#candidate structures per list	61-132	211-267
Total number of structures	7730	24128
#features (original)	25	25

There are 25 features that represent each protein structure. The features divide into two major types. One type measures the properties of the structure directly, such as pairwise atomic potential of the fold, the overall shape and packing, and the hydrogen bonding patterns. The other type of feature are termed “consensus features” and measure the similarity of a structure to other structures in the same list. The intuition is that correctly-folded structure is more likely to be similar to other predicted structures for the same target protein, and that incorrectly-folded structures would most likely be outliers. For a given structure x_i, its consensus feature is computed by $median(sim(x_i,x_j)).\forall j \neq i$. $sim(\cdot,\cdot)$ is a similarity function that measure the distance between two structures (such as the GDT-TS metric). The median, rather than the mean, is used to avoid being sensitive to skewed distributions.

[3]Qiu and Noble [124] trains by a regression objective, thereby allowing them to use these native protein structures as additional information. We cannot use these native structures in ranking because they are not lists, but single points.

3.3.2 Protein Structure Prediction Evaluation

Our evaluation metric consists of the GDT-TS [164] and the z-score, following [124] and CASP evaluations. The GDT-TS measures the structure's backbone quality when compared to the true structure. Similar to the MT evaluation, we also report the Top-k oracle GDT-TS scores (For K=5, this is equivalent to the GDT-TS5 metric used in [124]). The higher the GDT-TS score, the more similar the predicted structure is to the true structure. We report overall GDT-TS by averaging over GDT-TS of all test proteins.

In addition, since GDT-TS scores are not calibrated across different proteins, we additionally report z-scores. The z-scores are calculated by first taking the mean GDT-TS score of each list, then measuring how many standard deviations (z) the chosen structure is from the mean. The z-score calibrates for absolute differences of GDT-TS score between different protein targets. It can be positive (meaning that the chosen structure is better than average) or negative (meaning that ranking did worse than the average).

The state-of-the-art results, as reported in [124], can be summarized as:

- Top performing systems ([124]): GDT-TS = 0.589; z-score = 1.11 to 1.17
- Oracle (best structures are manually selected): GDT-TS = 0.636; z-score = 1.81
- The second-best system (SVR-noZhang in [124]) achieves GDT-TS = 0.576 and z-score=1.02. This difference of (0.589-0.576=0.013) is statistically significant from the top performing systems according to the Wilcoxon signed rank test.

Chapter Summary

We described the 3 applications (6 datasets) considered in this work. Characteristics of the datasets are summarized in Table 3.5. Importantly, we note that each application has different characteristics, which will be helpful in analyzing when an algorithm works and when it does not.

Table 3.5: Summary of all datasets used in this work.

	Information retrieval			Machine translation		
	TREC 2003	TREC 2004	OHSUMED	Arabic-English	Italian-English	Protein prediction
# lists	50	75	106	482	500	73
label type	discrete	discrete	discrete	continuous	continuous	continuous
avg # objects/list	983	988	152	260	362	61-132
# features	44	44	25	9	10	25
evaluation	MAP, NDCG			BLEU,PER		GDT-TS, z-score

Chapter 4
A LOCAL/TRANSDUCTIVE FRAMEWORK FOR RANKING

In this chapter, we propose a simple yet flexible local/transductive meta-algorithm for ranking. The key idea is to adapt the training procedure to each test list after observing the documents that need to be ranked. This framework allows us to explore various assumptions in unlabeled data.

The organization of the chapter is as follows: The proposed local/transductive meta-algorithm framework is presented in Section 4.1. Then we present a brief background review of RankBoost (Section 5.1.2), a supervised ranking algorithm that serves as a basic component in each of the three approaches we will discuss in subsequent chapters. Section 4.3 describes an extension of RankBoost that works with continuous-value labels, which will be necessary for the machine translation task. The goal of this chapter is to set up the basics, so that we can delve into actual algorithms in Chapters 5 to 7.

4.1 Description of Local/Transductive Framework

We use the following notation in this chapter. For concreteness, we will present the ideas using Information Retrieval as an example, so objects are documents, and lists are documents to be ranked. 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 4.1.

We now introduce our general framework for thinking about local/transductive ranking. We

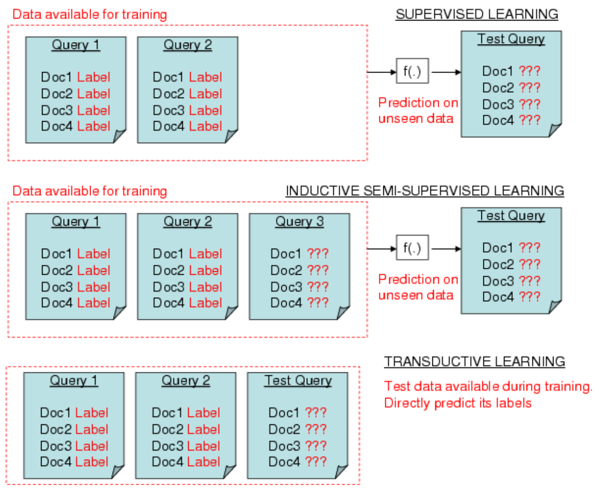

Data available for training
SUPERVISED LEARNING
Query 1
Query 2
Doc1 Label
Doc2 Label
Doc3 Label
Doc4 Label
f(.)
Prediction on unseen data
Test Query
Doc1 ???
Doc2 ???
Doc3 ???
Doc4 ???
INDUCTIVE SEMI-SUPERVISED LEARNING
Query 3
TRANSDUCTIVE LEARNING
Test data available during training.
Directly predict its labels

Figure 4.1: Supervised learning, inductive semi-supervised learning, and transductive learning: here we focus on the transductive setting, where test query is observed during training.

motivate it with the following question:

> 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 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. [1]

Algorithm 1 presents our general framework (meta-algorithm) for transductive ranking in pseudo-code. 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,

[1]The motivation here has some analogies to query classification (c.f. [59]), 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.

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 Local/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 transductive ranking technique. Pseudo-relevance feedback (c.f. [157], [110] 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 [22]. 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 [62] and by query-time association rules [150] have achieved promising results.

The fact that we focus on a single test list at a time has several advantages:

1. Letting our unlabeled data be the test data (transductive scenario) is arguably a simpler situation than inductive semi-supervised learning [149]. One main goal of the thesis is to explore how we can use unlabeled data, so the fact that the unlabeled data is the test data avoids additional challenges associated with learning generalization.

2. Focusing on a *single* test list at a time can be thought of as a form of local learning. Since ranking functions can be complex, local learning can be a convenient method to improve over state-of-the art baselines.

3. Importantly, combining local learning with transductive learning means that we only work with unlabeled objects from the *same* list at a time. This can help give a meaningful interpretation when we adopt assumptions from semi-supervised classification. For instance, the Low Density Assumption on pairwise instances extracted from a single list may reveal cluster structure, which gives a meaningful interpretation that objects with good and bad ranks exist on opposite sides of the feature space. However, if we had extracted pairwise instances from multiple lists, the cluster structure (if it exists), would be more difficult to interpret.

We investigate three instantiations of this general framework: The Feature Generation approach (Chapter 5) 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 (Chapter 6) 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. The Low Density Separation approach (Chapter 7) exploits the cluster assumption on pairs of objects extracted on the test list.

One note about terminology: We call the method "local" because it focused on a single test list at a time, and "transductive" because it does not train until test data is received. However, some may disagree with these terms. Especially, "transductive" sometimes imply that a set of test points are provided–in our case we only have one test list and therefore do not leverage any potential information that can be obtained across lists.

4.2 RankBoost: a supervised ranking algorithm

In successive chapters, we will use RankBoost or its variant as the component in Line 3 of Algorithm 1, therefore we will describe RankBoost in detail here. RankBoost [58] is an extension of the boosting approach [129] 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:

$$F(d^{(j)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(j)}), \tag{4.1}$$

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 2 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 $\theta_t$
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 $F(d^{(n)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(n)})$.
```

4.3 Modifications to RankBoost for continuous-level judgments

RankBoost, by design, works with discrete relevance judgments. As seen in the pseudocode in Algorithm 2, RankBoost extracts pairs of samples, where each pair represents two documents that have different ranks. Two documents with the same rank are not included as pairwise data because one does not need to learn whether one object is ranked higher than another.[2] RankBoost generally assumes a discrete set of relevance judgments, so it is clear which pairs of documents should be tied.

However, for Machine Translation (as well as Protein Structure Prediction), the "relevance judgment" comes in the form of real-valued scores. In MT, this may be the sentence-level BLEU score; for Protein Prediction, it may be the GDT-TS score of the structure. The question in this case is: If we have two translations, do we extract them as a pair for RankBoost if one has BLEU of .43 and the other has BLEU of .44? What if the difference is .43 vs. .45? Continuous value judgments pose two problems for RankBoost:

1. A difference in value (i.e. .44 vs .43) may not be significant enough to warrant an extracted pair. The values may be so close that the objects should practically be considered as tied.

2. If we were to extract pairs for all objects with absolute differences in their value judgments, there may be too many pairs and the memory requirements may become prohibitive. In gen-

[2]There is recent work, however, that seeks to exploit this kind of tie information. c.f. [169].

> eral, the number of pairs for continuous labels scale as $O(N^2)$ (where N is the number of documents in a list), but for discrete labels with small cardinality, the number of extracted pairs can be much smaller in practice.[3] For large N (e.g. N=1000 for 1000-best lists), the memory requirement for continuous labels may become prohibitive while that for discrete labels may still run efficiently.

A solution to the continuous-value label problem is to quantize the values into discrete levels. This approach was taken in [133], which applied RankSVM/Perceptron to MT Re-ranking by "splitting" the N-best list into positive and negative hypotheses. This essentially corresponds to a two-level quantization, where the midpoint is tuned. The results were mixed, however.

We use a different solution here. Rather than applying quantization, we extract pairs only if their score difference is above a threshold. This is clarified in the simple pseudocode in Algorithm 3. To compare this approach with quantization, consider the list shown in Figure 4.2. We see that quantization and Algorithm 3 extract very different pairs.

Algorithm 3 Pair extraction with threshold

```
Input: A training list $(q, \mathbf{d}, \mathbf{y})$, where $\mathbf{d} = \{d^{(j)}\}_{j=1..N}$
Input: User-set threshold $t \geq 0$
Output: A set $P$ of pairs $(i, j)$, $i, j \in (1..N)$ which represent hard preference relations

1: for $i = 1$ to $N$ do
2:     for $j = 1$ to $N$ do
3:         if $y^{(i)} > (y^{(j)} + t)$ then insert $(i, j)$ in $P$.
4:     end for
5: end for
```

The relative performance of these approaches depends on the data. Table 4.1 shows the dev set BLEU score (on the Arabic task) for a variety of quantization and pair extraction schemes. We chose the best one (threshold=30) in all our following experiments. In general, Algorithm 3 outperforms quantization on machine translation datasets.

[3]e.g. Suppose we have only two discrete labels, then the number of pairs range from $(N-1)*1 = N-1$ to $(N/2)*(N/2) = N^2/4$.

BLEU of individual hypothesis in a list
(BLEU=1.0) 1. please press yellow button
(BLEU=.6) 2. please press the yellow the button
(BLEU=.5) 3. please press the button yellow
(BLEU=.4) 4. please press button yellow
(BLEU=.3) 5. please press a yellow the button

Figure 4.2: Pair extraction example. The quantization approach may discretize all labels with BLEU$>$0.45 to 1 and all labels with BLEU $<$ 0.45 to 0, leading to the pairs (1,4), (1,5), (2,4), (2,5), (3,4). On the other hand, pair extraction with threshold (t=0.3) will extract entirely different pairs: (1,2),(1,3),(1,4),(1,5),(2,5).

Chapter Summary

In this chapter we introduced the local/transductive framework for ranking. The next three chapters will present practical algorithms that exploit different semi-supervised assumptions under this framework.

We also reviewed RankBoost and a modification thereof to work with continuous-level labels. This supervised ranking algorithm will be basic components in the local/transductive algorithms of the next chapters.

Table 4.1: Dev set BLEU of various pair extraction schemes

Pair Extraction Method	# Pairs Extracted	Dev BLEU
Algorithm 3, t=40	41k	29.0
Algorithm 3, t=30	251k	**29.2**
Algorithm 3, t=20	1480k	29.1
Quantize: Best hyp = 1, Others = 0	24k	26.6
Quantize: All hyp with BLEU >40 = 1, Others = 0	282k	27.2
Quantize: All hyp with BLEU > than that chosen by MERT = 1	225k	27.3
Quantize: All hyp in top 70 percentile = 1	525k	26.4
Quantize by k-means	701k	27.3

Chapter 5

INVESTIGATING THE CHANGE OF REPRESENTATION ASSUMPTION

The Change of Representation Assumption assumes that better feature representations are possible, and that unlabeled data can help discover these representations. In this chapter, we present a Feature Generation Approach (Section 5.1) which exploits this assumption under the Local/Transductive Framework (introduced in Chapter 4). Experimental evaluation of this approach in Information Retrieval, Machine Translation, and Protein Structure are presented in Sections 5.3 to 5.5. In addition, a novel feature extraction algorithm is described in Section 5.2–this method enhances the Feature Generation Approach.

5.1 Feature Generation Approach

The Feature Generation (FG) Approach, works by finding better features on the test data. It employs 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 a labeled training data with this new representation, which ideally is more pertinent to the test list in question.

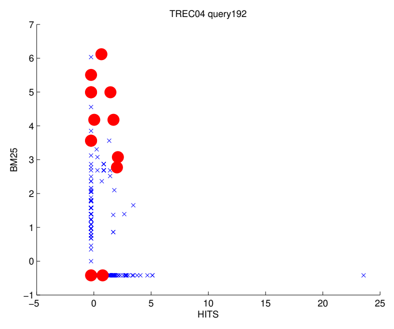
Algorithm 4 Feature Generation (FG) Approach to Transductive Ranking

```
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
```

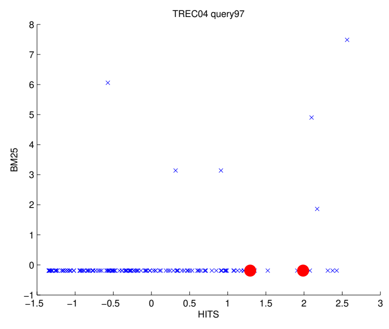
Algorithm 4 shows the pseudocode for this Feature Generation approach[1]. `DISCOVER()` is a generic unsupervised method that is applied to each test list $\mathbf{d}_u$ separately (line 2). `LEARN()` is a generic supervised method for learning rank functions. Since the feature-based representations of 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 5.1(a) and 5.1(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. For the first query (Fig. 5.1(a)), we see that the data varies mostly along the y-axis (BM25); for the second query (Fig 5.1(b)), the variation is on the x-axis (HITS). These two document lists would be better ranked by two different

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



(a)



(b)

Figure 5.1: 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.

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 thesis, we use kernel principal components analysis (Kernel PCA) [130] as the unsupervised method and RankBoost [58] as the supervised ranker. Kernel PCA is advantageous in its flexibility in generating many different types of features by the use of different kernels.[2] This is a good combination with RankBoost, which has been shown to be relatively robust to variations in tuning parameters and feature sets. These are described in detail in the following subsections.

5.1.1 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.

[2]Since we are using PCA transforms, we are performing a kind of *feature transformation* as opposed to *feature selection*.

Kernel PCA [130] 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.[3] 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 [132])

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 [130] for the derivation; here we only present the steps needed for this paper:

1. Define a kernel function $k(\cdot,\cdot) : (d^{(j)}, d^{(j')}) \to \mathbb{R}$ 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]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.

3. Let the $m \times m$ matrix K be the kernel values of all pairs of documents in the list. i.e. $K_{jj'} = k(d^{(j)}, d^{(j')})\ \forall j, j' \in \{1, 2, \dots, m\}$. This kernel matrix can be centered to ensure that the features Φ are zero-mean.

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 $d^{(n)}$, 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**: $k(d^{(j)}, d^{(j')}) = \exp(-\frac{||d^{(j)} - d^{(j')}||}{2\sigma})$. This is an isotropic kernel, with bandwidth σ adjusting for smoothness.

- **Diffusion kernel** [91]: 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 [139] is based on sparsity constraints and can extract patterns in $O(m)$.

5.1.2 *Supervised Ranking Algorithm: RankBoost*

We use RankBoost as the `LEARN()` component of the algorithm. 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 [112] 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.

5.2 RankLDA: Supervised feature transformation for Ranking

The Feature Generation Approach described above employs Kernel PCA as the method to discover useful features from the test list. However, this is not the only choice; in theory, any feature extraction/transformation method is possible.

In this section, we introduce a novel feature transformation method that complements Kernel PCA. Kernel PCA is a totally unsupervised method, and as such, it ignores information from the labeled part of the training data. While we rely on RankBoost to choose what Kernel PCA features are important for ranking, we could also directly generate features correlated with rank from the outset. We therefore introduce a *supervised* feature transformation method called RankLDA. The contributions with this method are that:

1. To the best of our knowledge, it is the first supervised feature transformation method that exploits rank information. Related methods such as Fisher's Linear Discriminant Analysis (LDA) was developed for classification, and therefore does not fit well into the ranking framework.

2. Features generated by the proposed RankLDA can be included with the feature set generated by Kernel PCA, thereby enlarging the model space for the Feature Generation Approach.

In the following, we first briefly review LDA, a feature transformation method for classification of which our method is based, before describing our supervised RankLDA method.

5.2.1 Feature Transformation by Linear Discriminant Analysis (LDA)

Recall the notation as follows: 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. Each document $\mathbf{d}_l^j$ is represented by a vector of K features, and the goal of feature transformation is to find a transformation matrix $\mathbf{A} \in \mathscr{R}^{K \times K'}, K' \leq K$ such that the projected document vectors $\hat{\mathbf{d}}_l^j = \mathbf{A}^T \mathbf{d}_l^j$, after input to a ranking algorithm, lead to better test results: i.e., a ranking algorithm trained on $\hat{S} = \{(q_l, \hat{\mathbf{d}}_l, \mathbf{y}_l)\}_{l=1..L}$ should outperform the same algorithm trained on $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}$. In supervised feature transformation, the labels $\mathbf{y}_l$ are used for computing $\mathbf{A}$.

Fisher's LDA is a classic method for computing the transformation matrix $\mathbf{A}$ for classification problems. The main idea is to find a transformation vector α that maximizes the covariance (scatter) among class means while minimizing the covariance within the same class. For a multi-class classification problem on dataset $(\{\mathbf{d}_l, y_l\}_{l=1..L})$, we find α by:

$$\arg\max_{\alpha} \frac{\alpha^T B \alpha}{\alpha^T W \alpha} \tag{5.1}$$

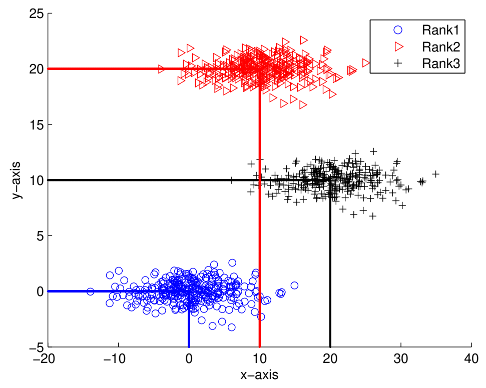


Figure 5.2: An example where LDA fails at ranking. Projecting on the y-axis will optimize Eq. 5.1 but doing so will reverse ranks 2 and 3. The x-axis is a better projection that respects the properties of linear ordering among ranks.

where $B = \sum_c (\mu_c - \mu)(\mu_c - \mu)^T$ is the between-class scatter and $W = \sum_c \sum_{l:y=c} (\mathbf{d}_l - \mu_c)(\mathbf{d}_l - \mu_c)^T$ is the within-class scatter. (Here, c indexes the class, μ_c is the class mean, μ is the overall mean).

LDA can be applied to a ranking problem by treating the relevance judgment as class labels. If discrete relevance judgment is provided for each document, then documents can be divided into classes in a straightforward way (e.g. "very relevant"=class 3, "relevant"=class 2, "irrelevant"=class 1). If relevance judgment is continuous-valued or is in the form of preferences between document pairs, then one should make some design decisions regarding how to map each document to a class label.[4]

[104] proved that ranking errors are upper-bounded by classification errors. One might therefore imagine that the LDA may work for ranking problems. However, one can construct artificial examples where achieving optimal class separation in the LDA sense actually reverses the ranking (Figure 5.2).

[4]The only additional modification we make in the ranking setting is to compute the scatter matrices independently for each document list. This avoids problems of uncalibration across different lists, and also allows for extraction of more features than the number of classes. For simplicity of presentation, in this paper, the B and W matrices are therefore sums of matrices from L lists.

5.2.2 Description of RankLDA

Our RankLDA method extends Eq. 5.1 for ranking. For ease of presentation, consider a 3-level ranking problem, where documents labeled 3 are preferred over those labeled 2, and 2 is preferred over 1.

First, we modify the idea of between-class scatter for pairwise relations. For the 3-level rank data, we would have 3 matrices: B_{12}, B_{13}, B_{23}, where B_{ij} is the between-class scatter between documents labeled i and j.[5] Second, we impose constraints on the projected variances $\alpha^T B_{ij} \alpha$ to respect the fact that variance of far-apart ranks $\alpha^T B_{13} \alpha$ should be larger than the variance of close-together ranks, such as $\alpha^T B_{12} \alpha$. This constraint enforces the algorithm to avoid reversing ranks at the expense of improving separability. Putting it together, we have:

$$\begin{aligned} & \arg\max_\alpha \frac{\alpha^T \tilde{B} \alpha}{\alpha^T W \alpha} \\ s.t. \quad & \alpha^T B_{13} \alpha > \alpha^T B_{12} \alpha \\ & \alpha^T B_{13} \alpha > \alpha^T B_{23} \alpha \end{aligned} \tag{5.2}$$

where $\tilde{B} = B_{13} + B_{12} + B_{23}$ is the combined between-class scatter. W is defined as in Equation 5.1. Note that there is no constraint of the form $\alpha^T B_{12} \alpha \sim \alpha^T B_{23} \alpha$, since we have no prior knowledge which of the scatter values should be larger.

Algorithm 5 presents the general RankLDA method. In each iteration, we compute the projection vector α in line 3, and the method of deflation is used to generate orthogonal projections iteratively. Note that slack variables ξ_q are added to allow for constraint violations and the parameter β determines the tradeoff. It is interesting to note that the objective in RankLDA has analogs to the formulation of RankSVM [82]. The main difference is that the concept of margin is here replaced with between-class scatter, and the constraints are now non-linear. We use a general interior-point algorithm (Matlab Optimization Toolbox) to optimize the objective.

[5] For N-level rank data, we have $N(N-1)/2$ matrices corresponding to $N(N-1)/2$ pairwise binary classification problems.

Algorithm 5 RankLDA

```
Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: $K'$ = number of features to be extracted
Input: $\beta$ = slack tolerance parameter
Output: Projection matrix $\mathbf{A}$
1: for $k = 1$ to $K'$ do
2:     Compute $B_{ij}$ for all $(i, j)$ pairs of distinct labels in $\mathbf{y}_l$. Compute $\tilde{B} = \sum B_{ij}$.
3:     Find projection vector $\alpha$:
           $\arg\max_\alpha \frac{\alpha^T \tilde{B} \alpha}{\alpha^T W \alpha} - \beta \sum \xi_q$
           $s.t. \quad \alpha^T B_{ij} \alpha + \xi_q > \alpha^T B_{rz} \alpha \quad , \xi_q \geq 0$
       for all preference relations $(i \leq r < z < j)$ or $(i < r < z \leq j)$
4:     $\mathbf{A}(:, k) = \alpha$
5:     Deflation: $\tilde{B} = \tilde{B} - (\alpha^T \tilde{B} \alpha) * \alpha\alpha^T)$
6: end for
```

While we do not attempt these here, RankLDA can be extended to include:

1. Non-linear formulation via kernels [132].

2. Semi-supervised extension by, e.g. adding a graph Laplacian objective [29].

5.3 Information Retrieval Experiments

We perform experiments on the LETOR dataset (version 2) [107], which contains three sets of document retrieval data: TREC'03, TREC'04, and OHSUMED. Our experiments compare the Feature Generation Approach (FG) with baseline systems under 5-fold cross validation, using MAP and NDCG as evaluation metrics. The Baseline is a supervised RankBoost, trained on the original training data. See Chapter 3 for details about the data.

For Feature Generation, we employed the following kernels for Kernel PCA and extracted five features for each, leading to an additional 25 features:

- Polynomial kernel, order 2
- Gaussian kernel, bandwidth = 1
- Diffusion kernel, time constant = 1
- Diffusion kernel, time constant = 10
- Linear kernel

Note that the hyperparameters of these features are chosen arbitrarily and have not been extensively tuned. They are only tested on a small subset of the development data to ensure that they are reasonable hyperparameters. The reason we chose not to tune the hyperparameters is that we work in the transductive scenario, so in practice there will not be time to tune a new set of parameters for a new set of dev/test data. We rely on RankBoost to choose the relevant features.

Table 5.1 shows the results (boldfaced MAP/NDCG numbers indicate a statistically significant improvement ($p < 0.05$) over Baseline.) We observe that nice results: Feature Generation (FG) improves over the Baseline in general. For example, for both TREC'03 and OHSUMED, FG achieves MAP values that are significantly better than the baseline (.3058 vs. .2482 for TREC'03; .4481 vs. .4424 for OHSUMED). Feature Generation also improves on all datasets for various NDCG values, though not all improvements are considered statistically significant.

For the OHSUMED dataset, which contains three levels of relevance judgments, we additionally applied RankLDA to create additional features for the Feature Generation Methods. The result row labeled (FG + RankLDA features) in Table 5.1 represents running RankBoost on top of the 25 original features, 25 Kernel PCA features, and 10 RankLDA features. We observe that FG+RankLDA gave a slight improvement over FG in MAP, a strong improvement over NDCG@1, and few changes for NDCG at lower positions. It is conceivable that the RankLDA features, which especially aim to separate ranks with large differences, have more impact on ranking top documents. In other words, whereas pairwise accuracy gives the same loss when a document with either relevance label 2 or 1 is ranked above a document with label 0, RankLDA would emphasize a larger separation for the case of label 2 than that of label 1. This may have the affect of pushing label 2 documents higher in the list, thereby improving NDCG@1.

Table 5.1: Main result for Feature Generation (FG). In general, FG provides improvements over baseline. Statistically significant improvements are bold-fonted.

	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**
`TREC'04`						
Baseline (supervised)	.3712	.4800	.4237	.4144	.4471	.4686
Feature Generation	.3760	.4800	.4514	.4415	.4665	**.4910**
`OHSUMED`						
Baseline (supervised)	.4424	.4906	.4543	.4501	.4230	.4218
Feature Generation (FG)	.4444	.5094	.4787	.4600	**.4469**	**.4377**
FG + RankLDA features	**.4481**	**.5252**	.4785	.4600	.4444	.4390

5.3.1 Information Retrieval: Detailed Analysis and Extensions

In order to understand the proposed framework better, we now describe an assortment of further experiments and analyses.

1. 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)? Kernel PCA may extract better features due to reductions in noise and discovery of useful non-linear combinations (due to the kernel). 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 5.2) 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 5.2: 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

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. Some insight into this question might be obtained by computing the correlation between the values of a Kernel PCA feature and an original feature. Table 5.3 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?

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 that no single combination of types is more prevalent than others. Figure 5.3 shows that the distribution of these (feature

Table 5.3: 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)

type) combinations. For example, seven of 25 rankers use a combination of original and diffusion kernel features; four of 25 rankers use a combination of original and polynomial kernel features. The results on the other two datasets show similar diversity. 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? Could similar gains be achieved if the Feature Generation approach were restricted to using 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.

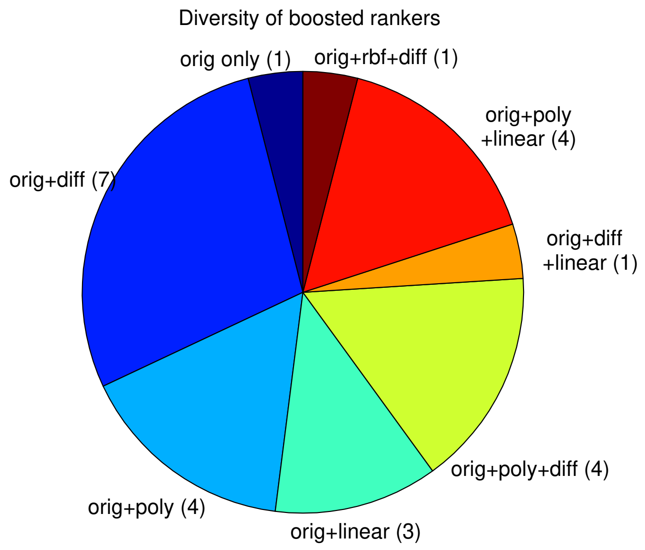
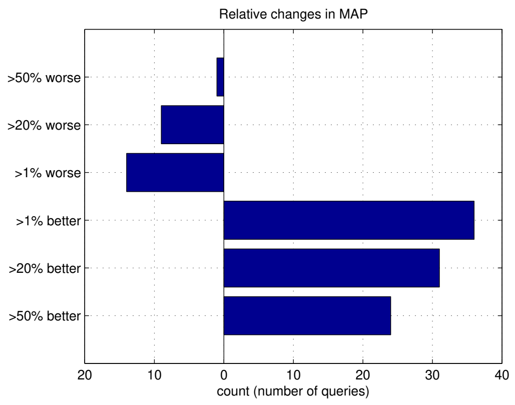
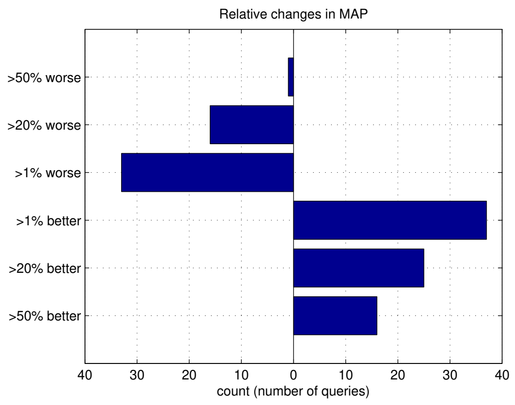


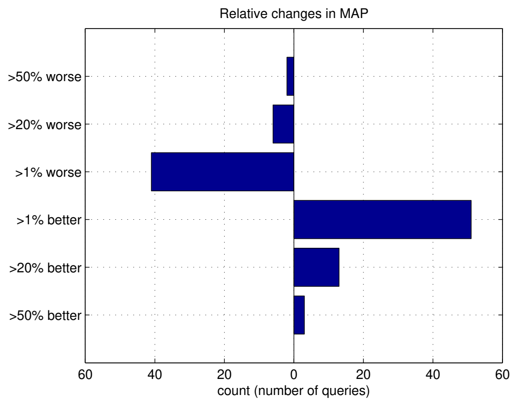
Figure 5.3: Pie chart showing the distribution of feature-type combinations for the 25 best rankers in the TREC'04 dataset. The number in the parenthesis indicates the count. For example, 3 of 25 rankers use a combination original and linear kernel features. The chart shows a diversity of feature combinations.



(a)



(b)



(c)

Figure 5.4: Query-level changes in MAP: We show the number of queries (in `Feature Generation`) 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.

5. How does performance vary across queries?

In Section 5.3, we present overall results averaged over all test queries. A more detailed analysis would include per-query MAP and NDCG. Figure 5.4 reports a histogram of queries that are improved vs. degraded by Feature Generation. For each plot, the bars on the right side indicate 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 FG approach does not give improvements across all queries. We are seeing gains in Section 5.3 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 under which the transductive approach is beneficial vs. harmful. On TREC'03, there is slight evidence showing that FG seems to benefit queries with poorer baselines (See Figure 5.5, 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. Viewed in another way, note that many of the our features (e.g. tfidf) attempt to quantify the match between the query and the document, so that high feature values usually correlate with better documents. However, while only the relative differences between feature values matter in ranking, RankBoost gives absolute value thresholds in its decision stumps. This implies that for some queries (possibly the difficult ones), most documents may pass or fail the feature value thresholds. By applying Kernel PCA to the documents of these queries, we can again focus on the features that have large relative range, which is expected to better discriminate among different documents in the set.

Nevertheless, it is difficult to test the above hypothesis. We 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. How do individual RankLDA features compare with LDA and original features?

How well do RankLDA and LDA perform, irrespective of the follow-up training algorithm for ranking? In order to evaluate this, we compared the first projection vector α extracted via RankLDA

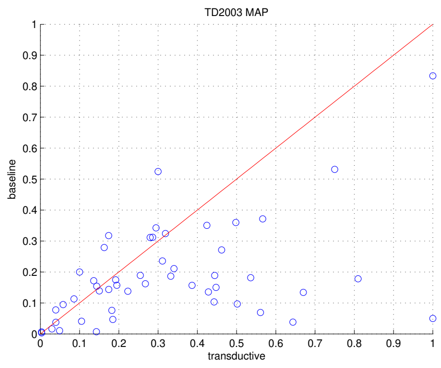


Figure 5.5: Scatterplot of TREC'03 MAP results for `Feature Generation` (x-axis) vs. `baseline` (y-axis).

or LDA. Specifically, we compute a single α on the training set, and rank the documents in the test set according to the projection values $\alpha^T \mathbf{d}$.

Table 5.4 compares RankLDA and LDA with the best and worst of the original 25 original features, where best/worst is determined on the test set (cheating experiment). For RankLDA, we tried different β parameters (0.1,1,10,50,100) and report the one with the best setting based on MAP/NDCG on the training set. We see that RankLDA consistently outperforms LDA, though it does not achieve the (cheating) results of the best possible original feature.

	RankLDA	**LDA**	**Original (Min)**	**Original (Max)**
MAP	0.4309	0.3005	0.3331	0.4488
NDCG@1	0.4346	0.1429	0.1851	0.5042
NDCG@7	0.4149	0.1764	0.2328	0.4354
NDCG@14	0.4109	0.1865	0.2480	0.4208

Table 5.4: Performance of single features. RankLDA and LDA are the rankings derived from the first projection vector α. The Original column presents the minimum and maximum test performance among the 25 original features.

Table 5.5: Arabic-English MT results

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	24.3	26.5	27.9	28.6	29.3	47.9
RankBoost (supervised) Baseline	23.7	26.4	28.0	28.9	29.6	47.9
Feature Generation (transductive)	23.4	25.7	27.0	27.9	28.6	48.3

5.4 Machine Translation Experiments

We apply the Feature Generation Approach on the two Machine Translation datasets, i.e. the Arabic-English and Italian-English IWSLT 2007 tasks. The experimental setup is identical to that described in the Information Retrieval experiments, except for the one modification to the RankBoost algorithm for continuous-value labels (see Section 4.3). For details regarding the data and task, refer to Section 3.

We compare the Feature Generation Approach with two baselines. Comparison with the supervised RankBoost baseline can give insight into the effect of unlabeled test data in training, since the Feature Generation approach is based on RankBoost. However, RankBoost is not a conventional algorithm used in MT, so we additionally compare with the MERT baseline. The MERT baseline is computed by running the minimum error rate training (MERT) algorithm [119], which generates linear weights.

The results for Arabic-English and Italian-English are presented in Tables 5.5 and 5.6. The general observations are:

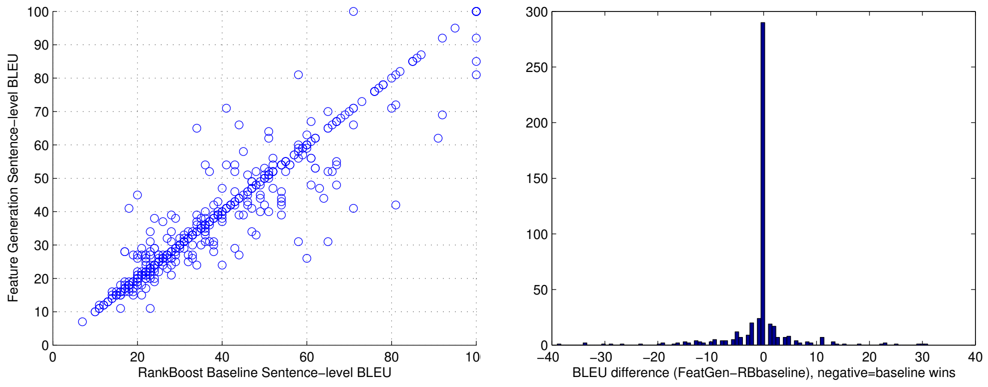
1. RankBoost is comparable to MERT in terms of BLEU results. For the Arabic task, RankBoost performs slightly worse than MERT for Top-1 BLEU but appears to improve at a slightly faster rate for Top-k with larger K.

2. However, Feature Generation gives degradation from the RankBoost baseline for both tasks. For Top-1 BLEU, we have FG (23.4) vs. Baseline (23.7) for Arabic translation, and FG (21.2) vs. Baseline (21.5) for Italian translation.

Table 5.6: Italian-English MT results

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	21.2	23.1	24.3	25.0	25.7	52.6
RankBoost (supervised) Baseline	21.9	23.6	24.7	25.4	26.0	51.7
Feature Generation (transductive)	21.5	23.4	24.4	24.9	25.3	52.5

Surprisingly Feature Generation did not work well in MT (whereas it worked well in IR). In order to analyze this, we first looked at the sentence-level BLEU score[6] of Feature Generation vs. RankBoost baseline results to see if there are consistent patterns of degradation. The visualization (for the Arabic data, results for Top-5 BLEU) is shown in Figure 5.6. In these graphs, we take the Top-5 result of FG (Overall BLEU=28.6) vs. RankBoost (Overall BLEU=29.6) and compared BLEU at the sentence level. Interestingly, we see that differences on the sentence-level do not appear to be large. The average difference of sentence-level BLEU scores between the two systems is only 0.5 points.



(a) (b)

Figure 5.6: Sentence-level BLEU analysis for Feature Generation vs. RankBoost Baseline. While the corpus-level BLEU result for RankBoost is 1 point better, there does not appear to be significant differences on the sentence level.

[6]Following [106], we compute a smoothed version of sentence-level BLEU with add-1 count for each n-gram precision to prevent arbitrary zeros. An alternative but more complex method to approximating sentence-level BLEU is described by [152].

We then looked deeper into the rankers that were trained via the Feature Generation method. On average, the pairwise training accuracy of FG rankers is 85.33%, compared to the RankBoost baseline, which is 82.42%. This means that the training algorithm in FG approach is optimizing the objective it was designed for better, due to the additional features. In fact, the percentage of weight that belongs to Kernel PCA features in the FG approach is around 0.4 (see Figure 5.7 for a histogram), which means that Kernel PCA features are consistently being used to a large extent. However, on the level of the individual ranker, the correlation coefficient between this percentage (i.e. how much weight is dedicated to Kernel PCA features, as opposed to original features) and the sentence-level BLEU, is $r = -0.0131$. This means that the amount of usage of Kernel PCA features has little correlation to the sentence-level BLEU score. Our summary of the observations:

1. In MT, Feature Generation performs worse than Baseline on overall (corpus-level) BLEU, but little difference is observed in sentence-level BLEU. Previous work has shown that sentence-level BLEU may not correspond well to corpus-level BLEU, but there is little choice for us since RankBoost (or any standard learning algorithm, with the exception of MERT) requires labels on the sentence level.

2. Feature Generation is indeed selecting Kernel PCA features during training, leading to better pairwise training accuracies, but rankers with more weight assigned to Kernel PCA do not necessarily achieve better sentence-level BLEU.

3. Therefore, the problem is likely that the pairwise accuracy optimized by RankBoost does not correlate very well with corpus-level BLEU.

In IR there is a strong relationship between the pairwise accuracy and the MAP and NDCG metrics, so enlarging the model space with additional features can give gains overall. For MT, there is a much larger chance of fitting a mismatched objective (since the link between pairwise accuracy and corpus-level BLEU is weaker), so more complex model spaces can lead to overfitting.

Another possibility is regarding the characteristics of the original features. There has been work [62] showing that cosine similarity with queries, for example, provide useful information for comparing lists. But in our setup throughout, we ignore the query (in IR) or source sentence (in

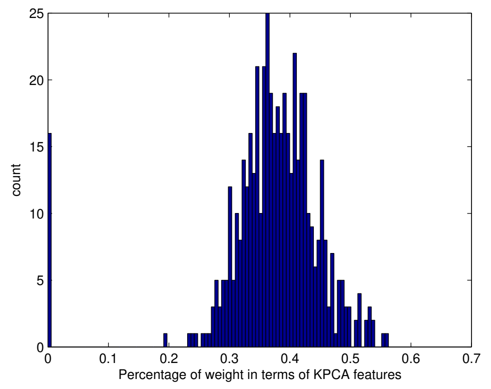


Figure 5.7: The percentage of total weight in RankBoost belonging to Kernel PCA features, in histogram (Arabic-English Translation Task)

MT); everything is done agnostically on the feature space.[7] Despite this, in IR, features such as *tf-idf* directly encode some information about the query (e.g. it counts the number of times the query term occurs in the document). However, in MT, features such as translation models only encode some information about the source sentences, since there there are many ways to translate a give phrase. This difference could potentially have an effect on the Feature Generation method.

5.5 Protein Structure Prediction Experiments

We applied the Feature Generation Approach to the Protein Structure Prediction dataset, using the same methods as IR. Although the Protein Prediction task also has continuous level labels like the MT task, we did not find the need to adopt a threshold in the pair extraction part of RankBoost. Empirically, we have observed that extracting all pairs without a threshold gave the best result on the training data.

The results for Feature Generation and the RankBoost and MERT baselines are shown in Tables 5.7 and 5.8. The MERT baseline of (GDT=.581, z-score=1.07) is comparable to the Support Vector Regression baseline reported in [124] (GDT=.589, z-score=1.07).

We observe that:

[7]The reason for ignore the query text information is that we want our methods to be application-dependent.

Table 5.7: Protein Prediction GDT-TS results

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	.581	.590	.597	.601	.604
RankBoost (supervised) Baseline	.579	.590	.595	.599	.604
Feature Generation (transductive)	.569	.586	.596	.601	.605

Table 5.8: Protein Prediction z-score results

Top-k z-score	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	1.07	1.17	1.26	1.31	1.34
RankBoost (supervised) Baseline	1.13	1.25	1.30	1.36	1.41
Feature Generation (transductive)	1.07	1.24	1.33	1.40	1.41

1. Feature Generation performs worse than the baseline for k=1, k=2, but is comparable for k>3 in Top-k GDT-TS.

2. The z-score results show similar trends. In general, we conclude that the Feature Generation method gives little gains or degradations for the Protein Prediction dataset.

3. None of the differences are statistically significant according to the Wilcoxon signed rank test.

To analyze the results in more detail, we look at the scatterplot (Figure 5.8) of Top-1 GDT-TS values for Feature Generation vs. the RankBoost baseline. The scatterplot shows that a majority of test lists do not exhibit GDT-TS differences for Feature Generation vs. Baseline. Only 19% of the lists are improved by 0.01 GDT-TS, and only 26% of the lists are degraded by 0.01.

Doing a similar analysis as done in Machine Translation, we looked at whether there are systematic differences between the 19% of FG rankers that improved vs. the 26% of FG rankers that degraded. Our goal is to determine whether the rankers that improved used a larger fraction of weight for the new Kernel PCA features in RankBoost. This would indicate whether the new features were useful; if the opposite trend were observed, this would indicate that the new features were

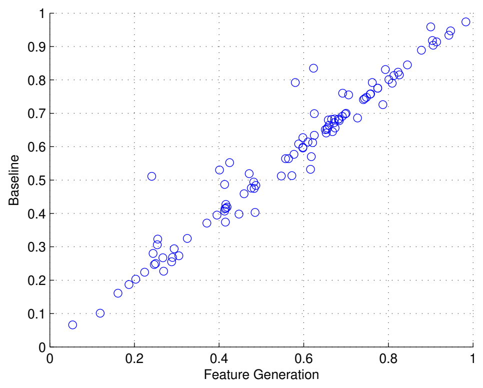


Figure 5.8: Scatterplot of GDT-TS values: Feature Generation (.569 average GDT-TS) vs. Baseline (.581 average GDT-TS). The majority of lists are not affected by Feature Generation; 19% of the lists are improved by 0.01, 26% of the lists are degraded. Correlation coefficient = .9717

actually detrimental. However, Figure 5.9 shows that there is actually little correlation between the amount of KPCA usage and the GDT-TS value. Furthermore, observing the histogram of KPCA usage (Figure 5.10), we see that there is on average only 17% usage of the new features, much less than that of Machine Translation (40%). Further, the Pairwise Training accuracies of the FG rankers (84.2% average accuracy, 83.9% minimum accuracy, 84.5% maximum accuracy) actually do not deviate much from the Baseline system (83.6%). This is in contrast to what we observed in Machine Translation–in that case, Pairwise Accuracy saw a statistically significant improvement of 3% when comparing FG to Baseline.

This together indicates that Feature Generation simply did not affect the Protein Prediction dataset very much. Unlike Information Retrieval, where new Kernel PCA features are useful, and unlike Machine Translation, where new Kernel PCA features were helpful for pairwise accuracy but harmed corpus-level BLEU, in the case of Protein Prediction, the effect of Kernel PCA was simply neutral and not used frequently by RankBoost.

Chapter Summary

We presented and evaluated the Feature Generation approach, whose main idea is to use Kernel PCA to discover better features from the test list before applying a supervised RankBoost learner.

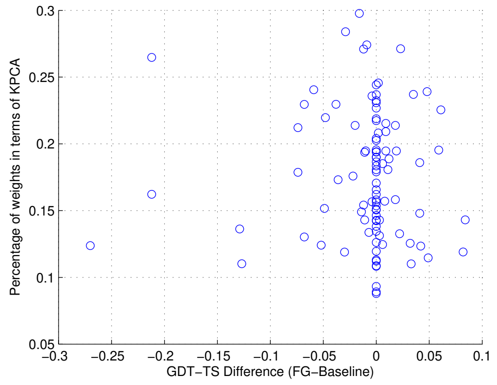


Figure 5.9: There is little correlation between the amount of Kernel PCA usage in Feature Generation vs. the GDT-TS score. (Protein Structure Prediction)

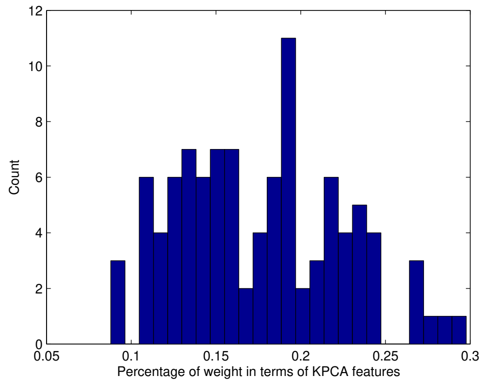


Figure 5.10: Percentage of total weight in RankBoost belonging to Kernel PCA features for Protein Prediction. Here, on average we have 17% of the weight represented by KPCA. Compare this to Machine Translation (Figure), which on average has 40% of weight dedicated to KPCA.

Results show that Feature Generation gave across-the-board improvements for all three Information Retrieval datasets. However, it gave degradations for Machine Translation datasets and little differences for the Protein Prediction dataset.

Our detailed analyses revealed several characteristics of the algorithm, such as:

- In Information Retrieval, it is not Kernel PCA per se, but its application on the test list that lead to improvements.
- Different test lists are best represented by different Kernel PCA feature representations, which reinforce the idea that test-specific ranking is a worthwhile research direction.
- For the machine translation datasets, the Feature Generation method is indeed choosing Kernel PCA features and improving the training pairwise accuracy. The fact that the method worked for Information Retrieval but not Machine Translation seems to indicate that pairwise accuracy is not as beneficial a surrogate loss function in MT as in IR.
- In Protein Structure Prediction, the new Kernel PCA features are simply not chosen often, which results in small changes in the pairwise training accuracy of RankBoost and overall little difference compared to the Baseline.

In addition to the above, we presented a novel feature extraction procedure for datasets containing more than two levels of discrete judgment labels. This RankLDA procedure could be used to enhance the Kernel PCA feature set in the Feature Generation Approach, and we indeed show improvements in the OHSUMED IR dataset.

Chapter 6
INVESTIGATING THE COVARIATE SHIFT ASSUMPTION

The idea of covariate shift (from domain adaptation literature) is that sample distribution differs between the training and the test set, but the functional relationship (mapping input to output, i.e. mapping a list to an ordering) remains the same. It assumes that observation of some unlabeled test samples is sufficient to allow us to "shift" the training distribution such that it better matches the test distribution.

In this chapter, we will propose a method that exploits the covariate shift assumption under the Local/Transductive Framework. The proposed Importance Weighting Approach is presented in Section 6.1 and evaluated in Sections 6.3 to 6.5. We also present a method to combine the Feature Generation Approach with the Importance Weighting Approach in Section 6.2.

6.1 Importance Weighting Approach

We now detail a way to exploit the Covariate Shift Assumption under the Local/Transductive Framework. The assumption is that each test list exists in a slightly different "domain" from the training data, thus Importance Weighting techniques (which looks at the locations of the unlabeled data in feature space) could correct the bias and improve results. A comparison of the standard covariate shift assumption vs. our local/transductive version is shown in Table 6.1.

The Importance Weighting (IW) Approach requires the two following components:

- An domain adaptation algorithm, `ADAPT()`, that generates importance weights specific to

Table 6.1: Comparison of Covariate Shift Assumption for Classification and Ranking

	Classification	Local/Transductive Ranking
Feature Space	Original vector of one object	Difference features from pairs of objects
Test Domain	All unlabeled vectors forms one domain	One domain per list

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 6 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 6 shows the pseudo-code for the Importance Weighting (IW) approach. In our instantiation, `WEIGHTED-LEARN()` is the AdaCost version of RankBoost [56] and `ADAPT()` is the Kullback-Liebler Importance Estimation Procedure (KLIEP) [141]. KLIEP is currently the state-of-the-art 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[1].

6.1.1 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 $\mathbf{d}_{u=1} \equiv \{d^{(j)}_{u=1}\}, j = 1..N_{u=1}$. 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 $(d^{(j)}_l, d^{(k)}_l)$ we derive the difference vector $x \equiv d^{(j)}_l - d^{(k)}_l$. 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 $\{x_u\}_{u=1..U_{pair}}$.[2]

4. Run the KLIEP importance weighting algorithm [141] 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 [141]. 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 $w(x) * p_{train}(x)$:

$$KL(p_{test}(x)//w(x) * p_{train}(x)) = \int p_{test}(x) \log \frac{p_{test}(x)}{w(x) * p_{train}(x)} dx \quad (6.1)$$

$$= \int p_{test}(x) \log \frac{p_{test}(x)}{p_{train}(x)} dx - \int p_{test}(x) \log w(x) dx \quad (6.2)$$

[1]We would define importance weights on the level of lists in Section 8.4

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

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

$$\mathscr{O}_{KLIEP} = \int p_{test}(x) \log w(x) dx \tag{6.3}$$

$$\approx \frac{1}{U_{pair}} \sum_{u=1}^{U_{pair}} \log w(x_u) \tag{6.4}$$

$$= \frac{1}{U_{pair}} \sum_{u=1}^{U_{pair}} \log \sum_{b=1}^{B} \beta_b \psi_b(x_u) \tag{6.5}$$

$$\tag{6.6}$$

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: $\psi_b(x) = \exp(-\frac{||x-x_{u=b}||}{2\sigma})$.[3] 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 occur in high density regions of the test pairs.

6.1.2 *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 7 shows the AdaCost modification. Note that the only change from traditional RankBoost (Algorithm 2 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:[4]

- 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)

[3]Note that the Gaussian kernel allows us to have infinite support for $p_test(x)$ and $p_train(x)$. Otherwise the integration in Equation 6.1 may not be valid.

[4]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 $[0, 1]$.

- 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.

Algorithm 7 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 $\theta_t$
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 $F(d^{(n)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(n)})$.
```

6.2 Combining Feature Generation and Importance Weighting

It is possible to combine Feature Generation and Importance Weighting into one method, since they operate at different stages in the Local/Transductive Framework. Feature Generation modifies the feature set, whereas Importance Weighting changes the optimization in RankBoost.

In this section we show one straightforward method for combining the two methods. The procedure is as follows:

1. For each test list, run Kernel PCA to obtain new feature representation
2. Run KLIEP on the new feature representation, obtaining a importance weighting for the training data (which are represented with augmented Kernel PCA features).

3. Train AdaCost-RankBoost on the augmented features and importance weights.

This combined method allows us to effectively select (soft) subsets of the training data while training with test-specific features. Since we believe that different feature representations are optimal for different queries, it may be reasonable to also believe that we should train on different subsets of the projected training data, rather than the whole set.

Whether this method works depends on whether *both* of Change of Representation and Covariate Shift Assumptions are satisfied.

6.3 Information Retrieval Experiments

In the experiments, we used our own implementation of AdaCost-RankBoost; the KLIEP software is available from [141]. We compare three systems: supervised RankBoost baseline, Importance Weighting, and the combined Feature Generation / Importance Weighting ranker.

The results are shown in Table 6.2. We observe that

- Importance Weighting outperforms Baseline in all three datasets. For example for MAP, we have .2482 (Baseline) vs. .2932 (Importance Weighting) for TREC'03. The improvements are statistically significant for numerous metrics on all datasets.

- The Combined FG+IW method improves upon Importance Weighting and achieves the best result overall. Further, observing Figure 6.1 which compares FG, IW, and FG+IW, we note that on virtually all metrics and all datasets, the combined method performs better or equivalent to either of FG or IW individually.

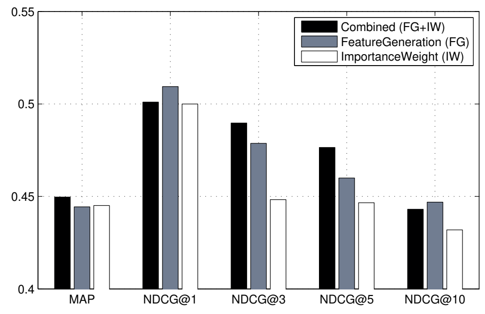
We now present a few more detailed analyses to better characterize the Importance Weighting method:

1. What is the effect of extracting all pairs of test documents

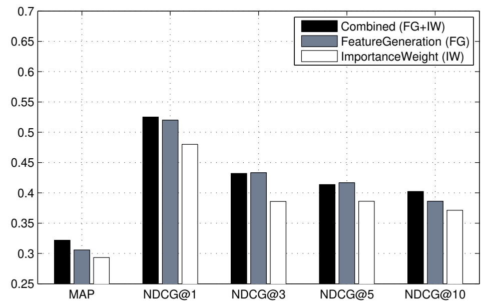
We demonstrated that the Importance Weighting approach consistently improves over the supervised Baseline. However, it is not clear whether our weights are optimal–there may be other sets of weights that achieve even better results. In particular, we are interested in the question of whether

Table 6.2: Importance Weighting Results on Information Retrieval. Importance Weighting (IW) outperforms the Baseline in various metrics. The combined Feature Generation (FG) and IW method gave further improvements.

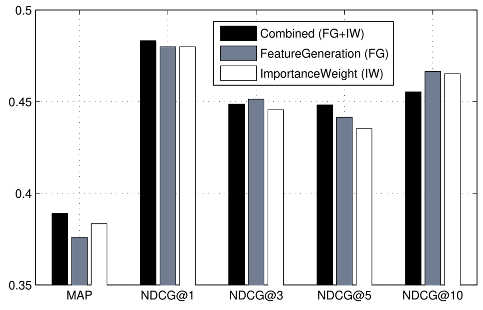
	MAP	N@1	N@3	N@5	N@10	N@14
`TREC'03`						
Baseline (supervised)	.2482	.3200	.3455	.3404	.3388	.3401
Importance Weighting	**.2932**	**.4800**	.3858	**.3862**	.3713	.3755
Combined FG+IW	**.3219**	**.5250**	.4321	**.4138**	**.4023**	**.3990**
`TREC'04`						
Baseline (supervised)	.3712	.4800	.4237	.4144	.4471	.4686
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
Importance Weighting	.4440	.5000	.4483	.4466	**.4319**	**.4280**
Combined FG+IW	**.4497**	.5010	**.4897**	.4765	**.4431**	**.4422**



(a)



(b)



(c)

Figure 6.1: Combining Feature Generation with Importance Weighting allows for soft selecting of the projected training data. Combined results improves MAP for all three datasets (a) OHSUMED, (b) TREC'03, and (c) TREC'04. Results are mixed for NDCG.

extracting all pairs of documents from the test list actually introduces a bias in the test distribution, since in actuality most of these pairs are not ranked pairs but ties.

Thus, we performed a cheating experiment: rather than extracting all pairs of documents in the test list, we extract only the pairs that have rank differences by observing the test labels. These "oracle" document pairs correspond to samples that contribute to the actual pairwise ranking test accuracies. On average, this corresponds to a 70%-80% reduction in the number of document pairs. This would therefore be a "more focused" target domain. The results in figure 6.2, as expected, show that the current importance weights can be improved, though the gap is not large for the case of OHSUMED. The test list has roughly 1000 documents for TREC'03 and 150 for OHSUMED. We believe there is more chance for improvement in TREC'03 since there are many more possible test document pairs in that dataset.

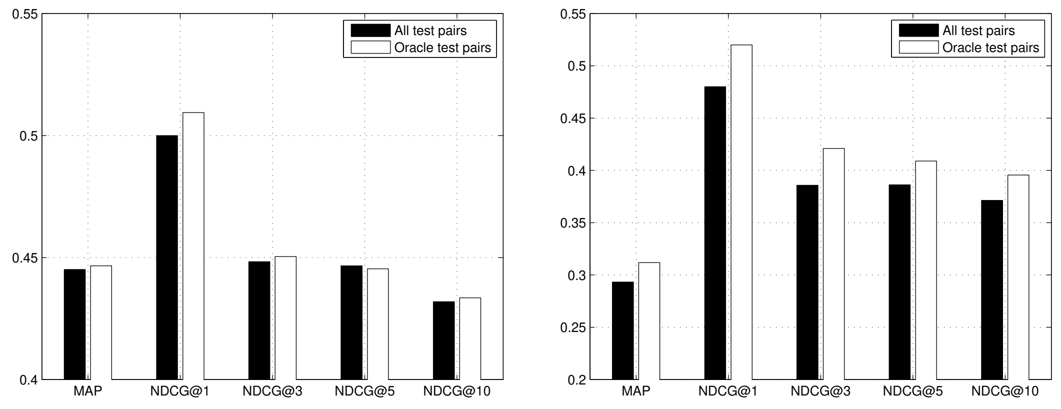
Note that this oracle result does not necessarily imply the absolute upper limit of the potential of Importance Weighting, since we are still using a particular implementation (KLIEP) to compute these weights. Though KLIEP is a state-of-the-art algorithm, there are other methods that may potentially achieve better weights.[5]

2. 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 KLIEP is indeed selecting subsets of the training data. Figure 6.3 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 6.3. 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

[5]For example, we have experimented with enhancing KLIEP with ranking-specific features derived from the initial list. Though the results are not statistically different (not reported here), it does show small improvements over the standard KLIEP algorithm. The point is that we think improvements to include ranking characteristics into a otherwise standard Importance Weighting algorithm could bring additional improvements.



(a) (b)

Figure 6.2: Comparison of importance weights extracted with all test pairs (current implementation) or oracle test pairs (cheating experiment). (a) OHSUMED shows a smaller gap, while (b) TREC'03 implies more chance for improvement can be achieved.

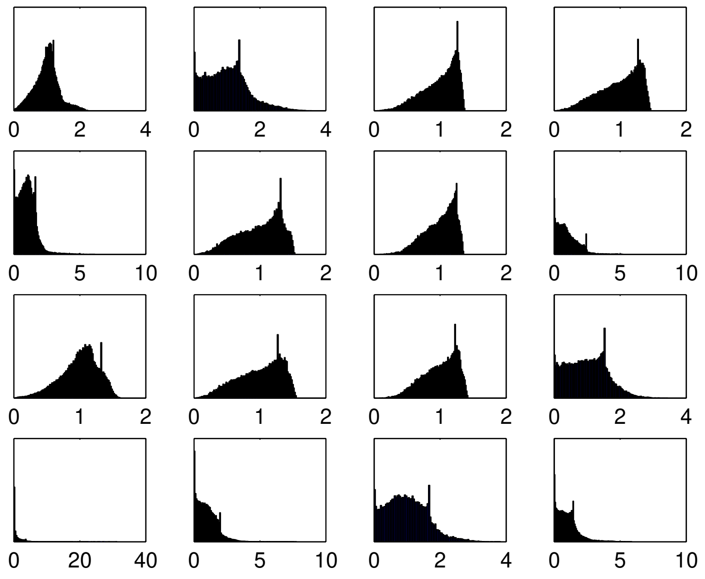


Figure 6.3: Importance weight histogram from some OHSUMED queries. The x-axis is the importance weight value; y-axis is the histogram count. The large variety in distribution implies that the target test statistics differ drastically.

Table 6.3: Importance weight statistics. Median represent the average median value of importance weights, across all test lists. Similarly, the 25Rh/75h quantile capture the value of the 25th and 75th portion of the weight's cumulative distribution function (CDF). Standard deviation and entropy show how much the importance weight distribution differs from the uniform distribution. 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	
	(all pair)	(oracle pair)	(all pair)	(oracle pair)	(all pair)	(oracle pair)
Median	0.9142	0.6588	0.5140	0.0370	0.0461	0.0539
75th Quantile	1.2808	1.1304	1.3642	0.2406	0.5137	0.0772
25th Quantile	0.6104	0.3789	0.1420	0.0143	0.0011	0.0533
Mode	0.9031	0.6511	0.0416	28.9366	0.3041	56.4709
Std Deviation	0.5712	1.4719	1.2951	18.7540	3.2136	35.3528
Entropy	1.8582	1.7784	1.9653	0.8520	1.2753	0.4803

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 broader than OHSUMED. In 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.

We also observe interesting statistics when comparing the (all test pairs) with (oracle test pairs) implementation (see Section 6.3). The (oracle test pairs) case results in a more focused target domain, so the importance weight distribution becomes more peaked: the entropy becomes lower, standard deviation becomes higher, and in general many more samples get low weights (leading to lower median and lower 25th quantile).

3. Both FG and IW gave improvements over the Baseline. How do they compare under data ablation experiments?

We performed data ablation experiments to see how Feature Generation and Importance Weighting 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 original datasplit is already randomized and independently sampled, so we do not expect any biases with this subsampling scheme. The results for MAP and NDCG@10 are shown in Figure 6.4. 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.

6.4 Machine Translation Experiments

We repeated the experimental setup for Machine Translation. The results for Arabic-English and Italian-English are presented in Tables 6.4 and 6.5. The general observations are:

1. Importance Weighting improves over the RankBoost baseline slightly for both tasks. In Arabic-English translation, Importance Weighting is 0.7 to 0.9 BLEU points higher than Baseline for Top-K (K = 1 to 5). In Italian-English, Top-1 BLEU is equivalent for both systems but for $K > 1$, Importance Weighting outperforms by 0.5 to 1.1 BLEU points. However, these improvements are not statistically significant according to the bootstrapping ttest procedure [167].[6]

2. The Combined FG+IW performs worse than IW individually, in general, and is often worse

[6]BLEU is a corpus-level metric, not an average of sentence-level metrics. Therefore, in order to compute confidence levels, bootstrapping is used. The idea is to randomly select (with repeats) sentences from the hypothesized outputs. Each set of random selection corresponds to a "sample" in the significance test. We perform 1000 samples, compute BLEU scores for each, and judge significance by the paired t-test, with $p < 0.05$.

Table 6.4: Arabic-English MT results

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	24.3	26.5	27.9	28.6	29.3	47.9
RankBoost (supervised) Baseline	23.7	26.4	28.0	28.9	29.6	47.9
Importance Weighting (transductive)	24.6	27.1	28.7	29.6	30.5	47.7
FG+IW	23.3	26.0	27.6	28.4	29.0	47.9

Table 6.5: Italian-English MT results

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	21.2	23.1	24.3	25.0	25.7	52.6
RankBoost (supervised) Baseline	21.9	23.6	24.7	25.4	26.0	51.7
Importance Weighting (transductive)	21.9	24.7	25.8	25.9	26.5	51.5
FG+IW	21.6	23.5	25.0	24.9	25.6	52.5

than the supervised Baseline as well.

It is interesting to observe that FG+IW performed worse than IW in Machine Translation but gave further improvements in Information Retrieval. We suspect that since the new Kernel PCA features do not correlate well with BLEU (and indeed FG did not perform well by itself in Machine Translation), it is detrimental to include them in calculating importance weights.

6.5 Protein Structure Prediction Experiments

The results and conclusions for Protein Structure Prediction is very similar to that of Machine Translation. Tables 6.6 and 6.7 compares IW and FG+IW with baselines and show that:

1. Importance Weighting slightly improves over supervised baselines for all k's in Top-k GDT-TS. However, the improvements are not statistically significant according to the Wilcoxon signed rank test.

Table 6.6: Protein Prediction GDT-TS results

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	.581	.590	.597	.601	.604
RankBoost (supervised) Baseline	.579	.590	.595	.599	.604
Importance Weighting (transductive)	.583	.596	.603	.605	.608
FG+IW	.568	.584	.593	.596	.601

Table 6.7: Protein Prediction z-score results

Top-k z-score	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	1.07	1.17	1.26	1.31	1.34
RankBoost (supervised) Baseline	1.13	1.25	1.30	1.36	1.41
Importance Weighting (transductive)	1.15	1.26	1.33	1.35	1.39
FG+IW	1.02	1.23	1.29	1.32	1.39

2. Using the Kernel PCA features to compute important weights appear to be detrimental, and the combined FG+IW performed worse than IW itself. This is observed in both GDT-TS scores and z-scores.

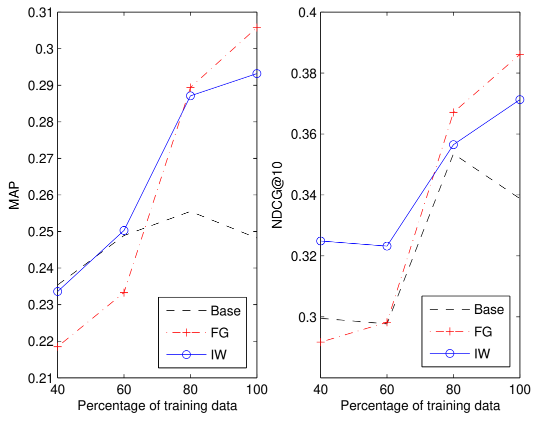
Further, we look at the scatterplot of GDT-TS values for Importance Weight vs. Baseline to see how results vary by individual lists. Figure 6.5 shows that Importance Weighting can be called a low-risk enhancement in the sense that the majority of lists (68%) are not affected much by the weighting, and that the degree of improvement/degradation is relatively mild.

Chapter Summary

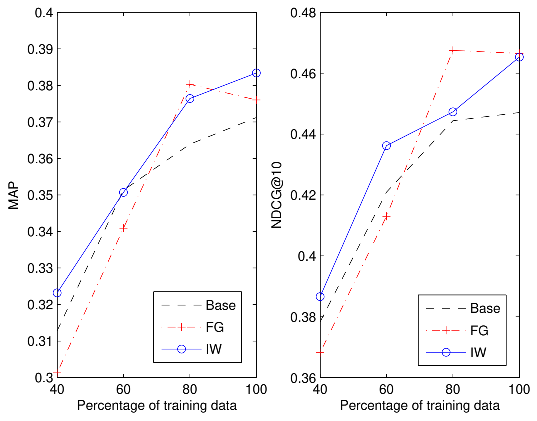
We presented the Importance Weighting Approach, which exploits the covariate shift assumption by adapting the training distribution to each test list. This is achieved by modifying the KLIEP importance weighting method for pairwise instances and using an AdaCost version of RankBoost to incorporate distributions.

Importance Weighting appears to be a relatively stable method, achieving gains in all datasets. In particular, the majority of gains in Information Retrieval are statistically significant. Data ablation experiments also show Importance Weighting to be a stable method that consistently outperforms the Baseline (in contrast to Feature Generation, which may underperform in low data scenarios).

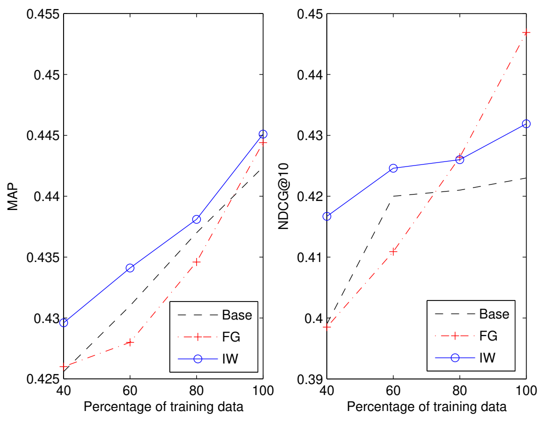
We also experimented with a Combined Feature Generation and Importance Weighting approach, which has the potential to adapt to test lists both in distribution and feature representation. This combined method gave further improvements in Information Retrieval, but actually degraded results in Machine Translation and Protein Prediction. This is most likely due to the effectiveness of Kernel PCA features in the first task and their corresponding ineffectiveness in the two latter tasks.



(a)



(b)



(c)

Figure 6.4: 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.

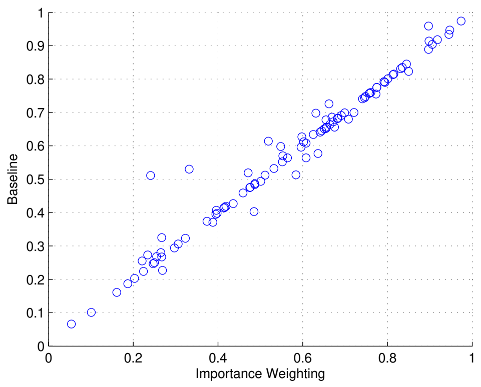


Figure 6.5: Scatterplot of GDT-TS values: Importance Weights (.583 average GDT-TS) vs. Baseline (.581 average GDT-TS). The majority of lists are not affected by Importance Weights; 12% of the lists are improved by 0.01, 20% of the lists are degraded. Correlation coefficient = .9827

Chapter 7

INVESTIGATING THE LOW DENSITY SEPARATION ASSUMPTION

7.1 Pseudo Margin Approach

Here we propose a method to exploit the low density separation assumption under the Local/Transductive Framework. Under the low density separation assumption, unlabeled data is used to define regions of low-density vs. high density. The (classification) function is assumed to cut through low-density regions, which corresponds to the intuition that objects that cluster together belong to the same category.

One common way to implement this assumption is using the concept of a “pseudo-margin” [82, 16, 51]. We will briefly portray this idea with boosting (classification), following [16]: For labeled data $\{(x_i, y_i)\}_i = 1..L$, where x are the features and y are the (classification) labels, AdaBoost minimizes the following objective function:

$$\sum_{i \in labeled} \exp(-y_i F(x_i)) \tag{7.1}$$

where $F(x_i)$ is the value of the classification function on sample x_i. The product $y_i F(x_i)$ is called the margin (following support vector machine literature), and it is positive if the (binary) classification is correct, negative otherwise. By minimizing the objective, we are basically maximizing the margin for labeled points.

For an unlabeled sample, the margin (or pseudo-margin, to make the distinction with the labeled case clear), the margin is defined as $|F(x_i)|$. This is equivalent to assuming that the label is $sign(F(x_i))$, i.e. which means that we trust the prediction to be correct. The objective with unlabeled points therefore becomes:

$$\min \sum_{i \in labeled} \exp(-y_i F(x_i)) + \gamma \sum_{i \in unlabeled} \exp(-|F(x_i)|) \tag{7.2}$$

The effect of the second term is low density separation. The hyperparameter γ adjusts for its importance in the overall objective. The function $F(\cdot)$ will veer away from any unlabeled sample x_i in order to achieve large pseudo-margin.

Previous work in classification has applied the pseudo-margin idea to boosting. The ASSEMBLE system of [16] achieved the best result of the NIPS 2001 Unlabeled Data Competition. Here, we will extend the pseudo-margin idea in [16] to RankBoost.

The main idea in our extension to ranking is to measure the pseudo-margin on pairwise samples. In (supervised) RankBoost, the objective function minimizes:

$$\sum_{(i,j)} \exp(-(F(x_i) - F(x_j)) \tag{7.3}$$

for all pairs (i,j) where item i ranks above item j. Here, $y * (F(x_i) - F(x_j)), y = 1$ can be thought of as the margin. To extend this to unlabeled samples, we simply define the pseudo-margin

$$|F(x_i) - F(x_j)| \text{for any pair (i,j) from unlabeled list} \tag{7.4}$$

Thus, our approach first extracts all pairwise samples from an unlabeled list. Then we minimize,

$$\sum_{(i,j)\in labeled} \exp(-(F(x_i) - F(x_j))) + \gamma \sum_{(i,j)\in unlabeled} \exp(-|F(x_i) - F(x_j)|) \tag{7.5}$$

which leads to low density separation in the pairwise sample space. The pseudo-code for the overall framework is shown in Algorithm 8, and the pseudo-code for the modified RankBoost is shown in Algorithm 9. Note that in contrast to previous methods, this method requires a semi-supervised ranking algorithm in the inner-loop of the general Local/Transductive Framework.

Importantly, we should note that using the pseudo-margin in binary classification automatically carries an implicit (and usually valid) assumption that an unlabeled point is either one class or the other. In a similar vein, using it in the ranking scenario automatically assumes that one item is definitely rank above another item (though we do not know which is which). This assumption in ranking actually leaves out one potential option: that is, the two items being of the same rank. Therefore, in cases where many ties are possible from the pairwise samples extracted from an unlabeled list, pseudo-margin as defined here may not be a suitable way to exploit unlabeled data. We will observe this effect in our experiments.

Algorithm 8 Pseudo-Margin (PM) 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: PAIR-EXTRACT(), a procedure to extract pairs of samples
Input: SEMI-LEARN(), a semi-supervised ranking algorithm
Output: Predicted rankings for test: $\{\mathbf{y}_u\}_{u=1..U}$
1: for $u = 1$ to $U$ do
2:     $P$ = PAIR-EXTRACT($\mathbf{d}_u$)
3:     $F_u(\cdot) =$ SEMI-LEARN($W, \{(q_l, \mathbf{d}_l, \mathbf{y}_l)\}_{l=1..L}, P$)
4:     $\mathbf{y}_u = F_u(\mathbf{d}_u)$ # predict test ranking
5: end for
```

Algorithm 9 RankBoost with Pseudo-Margins

```
Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Unlabeled Pairs $P$.
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 $\theta_t$
4:     Compute cost factor $c(i,j)$ depending on importance weight $w$
5:     Update weights $D(i,j) = D(i,j)\exp(c(i,j)\theta_t(h_t(d^{(i)}) - h_t(d^{(j)})))$ for (i,j) from labeled data.
6:     Update weights $D(i,j) = D(i,j)\exp(c(i,j)\theta_t(|h_t(d^{(i)}) - h_t(d^{(j)})|))$ for (i,j) from unlabeled
       data. Normalize.
7: end for
8: Output final ranking function $F(d^{(n)}) = \sum_{t=1}^{T} \theta_t h_t(d^{(n)})$.
```

7.2 Information Retrieval Experiments

In the experiments we evaluate whether low density separation of pairwise samples is a suitable objective for ranking. We will compare three systems:

1. Baseline: Supervised RankBoost

2. Pseudo Margin Approach, as described in the previous section

3. Pseudo Margin with Oracle Pairs: This is a semi-cheating experiment where only pairs of documents that are not tied in rank are extracted for computing pseudo margins. In other words, whereas the standard (and non-cheating) Pseudo Margin Approach extracts all possible pairs from the test list (i.e. $(N*N-1)/2$ pairs in total for N documents), the "Pseudo Margin with Oracle Pairs" system extracts only a subset of pairs which are known to have rank label differences.

It is important to note that the Oracle Pairs system, though provided with documents pairs that are not tied, is not given information as to which of the pair is ranked above the other. That is why we say it is a *semi*-cheating experiment. In a sense, this system more closely resembles the low density separation assumption in classification, where it is clear that the unlabeled sample (in this case, unlabeled pair) is one of two classes (and that their is no third option).

The Pseudo Margin Approach has one hyperparameter (γ) for trading off the effect of labeled versus unlabeled samples. In all our experiments here we set it to a default of $0.5*N_l/N_u$ (where N_u and N_l are the number of unlabeled and unlabeled pairs), meaning that the loss from the pseudo-margin is half of that of the real labeled margin. Preliminary experiments show that RankBoost is relatively insensitive to the adjustment of this hyperparameter in reasonable domains: we obtain similar results for values 0.1 to 1.0.

The results for Information Retrieval are shown in Table 7.1. We observe that:

- The Pseudo Margin Approach in general performs worse than or equal to the Baseline. The only exception where Pseudo Margin performed better (statistically-significant) is for the

Table 7.1: Pseudo Margin Results. The Pseudo Margin approach performed equal to or worse than the Baseline due to violation of the low density separation assumption. Most unlabeled document pairs are in practice tied in rank and should not be encouraged to have large margins. Once these tied pairs are removed, the Oracle Pairs result show dramatic improvements for all datasets.

	MAP	N@1	N@3	N@5	N@10	N@14
`TREC'03`						
Baseline (supervised)	.2482	.3200	.3455	.3404	.3388	.3401
Pseudo Margin	.2502	.3400	.3399	.3500	.3403	.3433
w/ Oracle Pairs	**.5158**	**.7000**	**.6704**	**.6392**	**.6143**	**.6170**
`TREC'04`						
Baseline (supervised)	.3712	.4800	.4237	.4144	.4471	.4686
Pseudo Margin	.3502	.4533	.4143	.4070	.4350	.4524
w/ Oracle Pairs	**.6858**	**.8400**	**.7999**	**.7598**	**.7470**	**.7501**
`OHSUMED`						
Baseline (supervised)	.4424	.4906	.4543	.4501	.4230	.4218
Pseudo Margin	**.4520**	.4560	.4342	.4334	.4208	.4196
w/ Oracle Pairs	**.5136**	**.5252**	**.5159**	**.5117**	**.4941**	**.4948**

MAP metric for OHSUMED.[1]

- Pseudo Margin with Oracle Pairs performed overwhelmingly above the baseline, giving the best results so far. This is a semi-cheating experiment, but it shows how tied pairs invalidates the low density separation assumption.

To investigate deeper the issue of tied pairs, consider the following statistic for TREC'03:

- There are roughly 983 documents per list. Thus the Pseudo Margin approach would extract $983 * 982/2 = 482,653$ pairs of documents.

[1]In this case, it appears that precision at low ends is benefiting from the Pseudo Margin, which improved MAP overall.

- There are however only 2 levels of relevance judgments in TREC (relevant vs. irrelevant), and on average there are only 1 relevant document in the list of 983 documents. This means that there are only $1 * 982 = 982$ pairs of documents with different ranks (i.e. the Oracle Pairs).

- Only 982 out of 482,653 pairs are not tied. This is 0.02%. In other words, the vast majority of extracted pairs in the Pseudo Margin approach are forced to have large margin, but in actuality, they should be tied and have little margin.[2]

The issue of tied ranks is interesting because it reveals a case where assumptions from semi-supervised classification do not apply to ranking.

It may be possible to enhance the Pseudo Margin Approach in tied-rank datasets, using for example a two-step approach. In the first step, a traditional ranker is used to order the test documents. Then, assuming this is a relatively accurate ordering, we can divide the documents into top-half and bottom-half, then extract document pairs between these two halves. This strategy relies on the Bootstrapping assumption in the first step, but we do not need to assume that the top-half is necessarily better than the bottom-half. The order could be reversed–as long as the tied pairs are eliminated, then the low density separation assumption used in the second step becomes suitable.

Another possibility is to reformulate the objective such that tied pairs are accounted for. We know that for non-tied pairs, we would like to minimize $\sum_{(i,j)\in unlabeled} \exp(-|F(x_i) - F(x_j)|)$ and for tied pairs, we would alternatively like to minimize $\sum_{(i,j)\in unlabeled} \exp(|F(x_i) - F(x_j)|)$. Further, from the labeled dataset, we can estimate the ratio of tied vs. non-tied pairs. This prior information could potentially be incorporated into the obective such that a fraction of the unlabeled pairs maximizes pseudo-margin while another portion minimizes it, and the optimization technique has the flexibility to choose which pair belongs to which category. This is analogous to transductive SVM solutions [82] where one can specify the ratio of positive vs. negative samples in the unlabeled sample.

[2]In fact, we performed an additional experiment where we *minimize* rather than maximize the pseudo-margin. This actually gave improvements in OHSUMED (.4800 MAP) and TREC'04 (.3523 MAP), but not for TREC'03 (.2503 MAP).

7.3 Machine Translation Experiments

The experimental setup for Information Retrieval is repeated for Machine Translation, with very different results.

Tables 7.3 and 7.4 show that the Pseudo Margin Approach gave significant improvements over the Baseline, and in fact performs better than the Oracle Pairs version. Since machine translation uses continuous-value labels, we define oracle pairs in the same way labeled pairs are defined: if the difference of sentence-level BLEU exceeds a certain threshold, then a pair is considered to be ranked. If the difference is below the threshold (or zero), then they are considered tied.

In Arabic-to-English translation, the Top-1 BLEU of 26.1 for Pseudo Margin outperforms the Baseline of 24.3. Similarly, the BLEU score of Oracle Pairs (25.0) also improves upon the Baseline. In Italian-to-English translation, Pseudo Margin (24.3) and Oracle Pairs (23.7) also outperform Baseline (21.2). Interestingly, for this dataset, using only Oracle Pairs actually does not perform as well as extracting all pairs. This is because there are actually relatively few ties in the Machine Translation N-best lists–these lists have been de-duplicated so that identical surface strings are removed, and so most pairs of hypotheses would likely benefit from the large pseudo-margin. In contrast to the 0.02% non-tied pairs in Information Retrieval, in the Arabic task for example there are up to 20% of pairs that are not tied in terms of sentence-level BLEU.

Finally, we compare the translation output of baseline vs. Pseudo Margin in more detail. Table 7.2 shows the breakdown comparison of BLEU computation. We observe that the gains in BLEU for Pseudo Margin comes from better n-gram matches (and not the brevity term). Figure 7.1 shows some example translations obtained by the different methods.

We conclude that the low density separation assumption works well for our machine translation data. It would be worth investigating in future work whether this generalizes to other machine translation datasets.

7.4 Protein Structure Prediction Experiments

The results for Protein Structure Prediction are shown in Tables 7.5 and 7.6. We observe slight degradations with the Pseudo Margin Approach. The Oracle Pairs system performs slightly better than the standard Pseudo Margin Approach, but do not outperform the Baseline as seen in Informa-

Table 7.2: Breakdown comparison of BLEU for Baseline (MERT) vs. Pseudo-Margin

	Baseline	Pseudo-Margin
1-gram precision	65.03	66.74
2-gram precision	33.23	35.29
3-gram precision	18.30	19.94
4-gram precision	9.83	11.07
overall precision	.2497	.2685
length ratio (brevity)	.9731	.9725
BLEU	24.3	26.1

Table 7.3: Arabic-English MT results. The Pseudo Margin Approach outperforms the Baseline in all metrics. Boldface represents statistically significant improvement via the bootstrapping approach [167]

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	24.3	26.5	27.9	28.6	29.3	47.9
RankBoost (supervised) Baseline	23.7	26.4	28.0	28.9	29.6	47.9
Pseudo Margin	**26.1**	**28.8**	**30.1**	**30.9**	**31.8**	**46.5**
w/ Oracle Pairs	25.0	28.0	29.6	30.6	31.2	46.7

Table 7.4: Italian-English MT results. The Pseudo Margin Approach outperforms the Baseline in all metrics. Boldface represents statistically significant improvement via the bootstrapping approach [167]

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	21.2	23.1	24.3	25.0	25.7	52.6
RankBoost (supervised) Baseline	21.9	23.6	24.7	25.4	26.0	51.7
Pseudo Margin	**24.3**	**26.1**	**27.0**	**27.8**	**28.4**	**48.6**
w/ Oracle Pairs	23.7	25.8	26.7	27.3	27.9	48.8

REF: the store is usually open from nine am to six pm

Baseline: open shop usually at nine SbAHAFAlY pm

Pseudo-Margin: open shop is usually at nine SbAHAFAlY six in the evening

---

REF: it's not salt-free we can change it to salt-free if you need

Baseline: it's not is it urgent but we can change for another seat is that man

Pseudo-Margin: it's not is from the salt and but we can change for the last is it

---

REF:sorry you cannot turn the tv on until the plane has taken off

Baseline: excuse me i you turn tv until the plane departs

Pseudo-Margin: excuse me not you turn set until the plane departs

Figure 7.1: Example translation outputs for Baseline vs. Pseudo-Margin.

Table 7.5: Protein Prediction GDT-TS results

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	.581	.590	.597	.601	.604
RankBoost (supervised) Baseline	.579	.590	.595	.599	.604
Pseudo Margin	.574	.590	.599	.603	.608
w/ Oracle Pairs	.578	.591	.599	.603	.608

tion Retrieval.

The scatterplot of Pseudo Margin vs. Baseline for individual test lists reveal an interesting result (Figure 7.2). Compared to similar scatterplots for Importance Weighting and Feature Generation (Figs 6.5 and 5.8), the results here are much more varied (and the correlation coefficient is smaller). In the Pseudo Margin Approach, 70% of test lists differ by more than 0.01 from the Baseline GDT-TS. (Among these, 37% of the lists are improved by 0.01, and 33% of the lists are degraded, but overall the average GDT-TS is still a slight degradation.) Thus, in comparison, Pseudo Margin is a riskier approach–there is much more potential to improve, as well as degrade the Baseline results.

Table 7.6: Protein Prediction z-score results

Top-k z-score	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	1.07	1.17	1.26	1.31	1.34
RankBoost (supervised) Baseline	1.13	1.25	1.30	1.36	1.41
Pseudo Margin	1.03	1.24	1.34	1.40	1.47
w/ Oracle Pairs	1.09	1.26	1.35	1.40	1.45

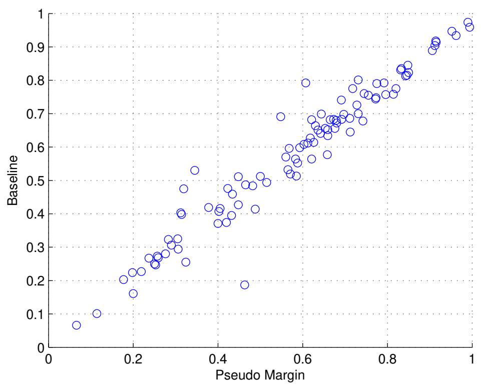


Figure 7.2: Scatterplot of GDT-TS values: Pseudo Margin (.574 average GDT-TS) vs. Baseline (.581 average GDT-TS). In contrast to Importance Weighting, here the majority of lists are affected by the Pseudo Margin Approach: 37% of the lists are improved by 0.01, 33% of the lists are degraded. Correlation coefficient = .9681

Chapter Summary

We apply the low density separation assumption to ranking by the definition of pseudo-margin over pairs of unlabeled documents extracted from the test list. The three different datasets exhibit drastically different results:

- In Information Retrieval, the abundance of tied-ranks led to inferior performance of the Pseudo Margin approach. However, once these ties are eliminated in a semi-cheating experiment, significant improvements can be seen. An avenue of future research is to use an initial ranking to identify possible tied pairs and eliminate them before the Pseudo Margin approach is applied.

- In Machine Translation, Pseudo Margin give significant improvements. The Oracle Pairs version also gave improvements, albeit less.

- We observe slight degradations with Pseudo Margin on the Protein Prediction dataset. These degradations are not statistically significant.

The Pseudo Margin Approach therefore appears to be high-risk and high-reward. If the low density assumption is satisfied (as in the case of Machine Translation or Oracle Pairs in Information Retrieval), then significant gains may be seen. On the other hand, issues such as tied ranks may violate the assumption and lead to degradations for this method.

Chapter 8
KERNELS ON LISTS

In this chapter we will introduce a kernel that operates on lists. The motivation for such a kernel is that it enables more classification techniques to be applied to ranking. In the following, Sections 8.1 and 8.2 will discuss the motivation and the related work in detail. The proposed List Kernel is presented in Section 8.3.

We will apply list kernels for semi-supervised ranking in two distinct ways: Section 8.4 explores using list kernels under the Covariate Shift Assumption, similar to the Importance Weighting method of Section 6 with the exception that kernels are applied to pairs of lists rather than pairs of vectors. Section 8.5 uses the distance information between lists in the framework of graph-based methods and proposes a variant of label propagation called Ranker Propagation.

8.1 Motivation

Many popular learning algorithms, be it supervised or semi-supervised, work with kernels that characterize the similarity between sample points. For example, support vector machines use kernel functions to measure the similarity of sample points in some implicitly-defined high-dimensional space. The use of kernels not only gives computational speed-ups but also accuracy improvements. Graph-based methods represent another class of techniques where kernels[1] or similarity information between points are central; in this case, the overall manifold of the dataset is captured by local distance functions. Many kernel methods or graph-based methods are modular with respect to the kernel they use, giving the designer the flexibility to plug-in different kernels (which may represent different quantifications of the designer's concept of invariance). It is however challenging to apply these methods directly to ranking if the kernels are defined over samples points (e.g. vectors), rather than lists (e.g. set of vectors).

We are therefore motivated to develop novel kernels that operate on *lists*. With *List Kernels*, we

[1]Kernels are actually not required for graph-based learning; it is sufficient to have a non-negative similarity function.

can conceivably adapt many kernel and graph-based methods for ranking problems. The criteria for a List Kernel $K(\cdot,\cdot)$ is as follows:

- The List Kernel $K(\cdot,\cdot)$ is a function that maps two lists to a non-negative scalar. i.e. $K(x,y)$: $R^{d\times N_x} \times R^{dxN_y} \rightarrow R^+$, where x and y are two lists.

- The number of vectors in x and y need not be equivalent, i.e. $N_x = N_y$ is not required.

- The kernel is computed only using information from the unlabeled portion of the lists (e.g. the features of each hypothesis in an N-best list), and no labeled information is utilized.

- Intuitively, the kernel should give large values for two lists that are similar, and small values for two lists that are dissimilar, where the concept of similarity is yet to be defined.

- $K(x,y)$ is symmetric (i.e. $K(x,y) = K(y,x)$) and positive semi-definite (i.e. $c^T * K(x,y) * c \geq 0$, for any x,y and any $c \in R^m \neq 0$)

8.2 Related Work on Kernels

While kernel design has been an active area of research, there has been considerably little related work for kernels on lists. Most previous work has focused on kernels for vectorial data, combinatorial data, and structured data (c.f. [132]). To the best of our knowledge, the only work involving kernels over lists of vectors are in the computer vision literature. The motivation in these cases is to deviate from the common practice of representing a two-dimensional image as a single vector (e.g. a 32 x 32 image would be represented as a vector of length 1024, where each entry is a pixel). Instead, the image would be represented by 32 vectors of length 32 each.

The works in computer vision (more specifically, image recognition) trace their ideas to mainly two different approaches. One is based on distance measures between probability distributions induced from the list of vectors, while the other is based purely on geometrical properties.

The work of Kondor & Jebara [90] is based on first generating a probability distribution that represents each list, then measuring the divergence between the distributions. Conceptually, the following steps are involved:

1. For list $x = \{z_n\}_{n=1..N_x}$, train a multivariate Gaussian $p(z) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-(z-\mu)^T \Sigma^{-1}(z-\mu)\}$ using the N_x vectors in the list.

2. Similarly, obtain the multivariate Gaussian by training on vectors in list y (i.e. compute the mean μ and covariance Σ).

3. Compute the distance between distributions of two lists ($p(z)$ and $p'(z)$) using the Bhattacharyya distance:
$$K(p(z), p'(z)) = \int (\sqrt{p(z)} - \sqrt{p'(z)}) dz \tag{8.1}$$

While different distances on distributions (e.g. the Kullback-Liebler Divergence) could be used, [90] showed that efficient computation of the above kernel can be achieved with the Bhattacharyya distance. The final kernel value can be computed in closed-form based on means and covariances, without requiring integration. For convenience, we will refer to this as the Bhattacharyya kernel.

The works of Yamaguchi et. al. [161] and Wolf & Shashua [154] are based on geometric properties of the lists using the concept of principal component angles, due to Hotelling [71]. Let U_x and U_y be the subspaces spanned by samples in list x and y, respectively. This subspace can be computed by principal components analysis, for example, leading to $U_x = [u_x^1 u_x^2 u_x^3 ..]$, where $< u_x^i, u_x^j >= 0 \; \forall \; i \neq j$. The principal angle between the two subspaces is defined as:

$$cos(\theta) = \max_{u \in U_x} \max_{v \in U_y} u^T v \tag{8.2}$$

In other words, the kernel value between two lists is the maximum dot product between two sets of basis functions computed by principal components analysis. For convenience, we will refer to this second approach as the principal angles kernel.

Note that both of the above kernels capture some information about the *orientation* of the list. If two lists differ from only slight rotation, they will receive high similarity. In addition, the principal angles kernel has a shift-invariant property, meaning that one could add a constant offset to all elements of a list without affecting the kernel value. It is also scale-invariant since a constant multiplier will not change the set of basis functions. On the other hand, the Bhattacharyya kernel is sensitive to shifts and scaling in feature space. For the application of ranking, we imagine that the *shape* and

Table 8.1: A summary of properties of kernels on sets of vectors. List Kernel is proposed in Section 8.3

	Bhattacharyya	Principal Angles	List Kernel
Shift-invariance	no	yes	yes
Scale-invariance	no	yes	yes
Rotation-invariance	no	no	no

orientation of the list is most important, since it shows the relationship among objects in the same list. In many cases, what the ranker does is to tradeoff the relative importance of different features, and this ratio manifests itself in the shape and orientation of the list. Shape, however, is not captured by the principal angles kernel since it only uses information about the PCA eigenvectors, and not the eigenvalues. Shift invariance and scale invariance may be desirable properties if we believe that the ranking function should not vary drastically at different parts of the feature space. They may not be desirable properties if we believe otherwise.

A summary of different properties of is shown in Table 8.1.

8.3 Formulation of a List Kernel

In this section we present our kernel, which we simply call the List Kernel. Part of the formulation turns out to be similar to the principal angles kernel since we focus on geometric properties of point clouds, though there are significant differences. We will try to characterize shape and rotation differences between two lists. The main idea is to first use principal component analysis to characterize the subspace spanned by objects within a list, then use a maximum weighted bipartite matching algorithm to find the distance between all basis vectors of this subspace. One can imagine trying to rotate one list such that it maximally aligns to the second list. The amount of work required to do this is the difference, and the inverse would be the similarity. The matching algorithm is required because there are many ways to match basis vectors from one list to basis vectors of the other list. We will enforce an one-to-one mapping between basis vectors because we are "rotating" the list as a rigid body in space and not doing any deformation. In sum, we compare two lists by shape and orientation similarity, and in the following we propose one specific algorithm to achieve it.

The pseudo-code for List Kernel is shown in Algorithm 10. To illustrate the kernel, suppose we have computed the basis vectors for list x: $[u_x^1\ u_x^2\ u_x^3]$, as well as the basis vectors for list y: $[u_y^1\ u_y^2\ u_y^3]$. For ease of explanation, supposed we had only extracted the top three principal component axes. If the top eigenvectors (principal axes) u_x^1 and u_y^1 point in similar directions, then their dot product is high and the corresponding list kernel value will be high. On the other hand, if u_x^1 and u_y^1 are dissimilar, but u_x^1 and u_y^2 are similar, then the list kernel value should be medium-ranged (in effect weighted by $\lambda_x^1 \times \lambda_y^2$). Finally, if none of the three eigenvectors of x match well with that of y, then the list kernel value will be small. The goal of the maximum bipartite matching step in Algorithm 10 is to find the best possible one-to-one correspondence between the two subspaces, so therefore the list kernel value is defined as the attained matching value.

Algorithm 10 Computing the List Kernel

```
Input: List $x$ and list $y$.
Output: Kernel value $K(x,y)$.
 1: $[U_x;\Lambda_x] = PCA(x)$ (Compute $M$ principal component axes $u_x^m$, $m = 1..M$ and eigenvalues $\lambda_x^m$,
    based on vectors in list $x$.)
 2: $[U_y;\Lambda_y] = PCA(y)$ (Similarly compute for list $y$.)
 3: Define a bipartite graph $G$ with $M^2$ edges and $2M$ nodes. One side of the graph represent $u_x^m$
    and the other side represent $u_y^m$.
 4: for $m = 1$ to $M$ do
 5:   for $m' = 1$ to $M$ do
 6:     Compute the edge weight, defined as the dot product between principal axes, weighted by
        the corresponding eigenvalues $\lambda_x^m \lambda_y^{m'} \cdot | < u_x^m, u_y^{m'} > |$.
 7:   end for
 8: end for
 9: Compute maximum weighted bipartite matching on graph $G$. The unnormalized kernel value
    $\hat{K}(x,y)$ is defined as the maximum matching value, i.e. $\hat{K}(x,y) = \sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)} \cdot | < u_x^m, u_y^{a(m)} >$
    $|$. where $a(\cdot)$ is a bijection $a : 1..M \rightarrow 1..M$ that represents the bipartite matching.[2]
10: The output kernel value $K(x,y)$ is normalized by the norm of eigenvalues:
    $K(x,y) = \hat{K}(x,y)/(||\lambda_x|| \cdot ||\lambda_y||)$
```

This list kernel has advantages over the principal angles kernel because it considers the overall shape and orientation of the lists. The principal angles kernel, with its "max-max" operation only considers the correspondence of only one pair of eigenvectors. Principal angles kernel can be seen as an extension of cosine distance (for vectors) to subspaces. Since the focus is on "cosine distance" between subspaces, there is no measure of the shape, which is characterized by ratios of eigenvalues of difference principal component axes. For example, if u_x^1 matches well with u_y^2, the principal component kernel will achieve high value regardless of whether the remaining eigenvectors match well. Further, the lack of weighting of principal axes may lead to stronger sensitivity to the number of components extracted (M). It is important to note that while both methods employ principal components as a first step, the principal angles kernel is most concerned with measuring the angle between subspaces, while our proposed list kernel focuses on matching the overall shapes between point clouds in lists. The List Kernel is also shift-invariant. The example in Figure 8.1 illustrates how List Kernel works.

We use the Hungarian Method (also known as the Kuhn-Munkres algorithm) for weighted bipartite matching. The overall computation cost is:

- $O(d^3)$ for the principal components analysis (d is the dimension of the feature vectors, which range from 10-50 in our tasks)

- plus $O(M^2)$ for computing the edge weights in the bipartite graph G. (M is the number of principal components extracted, which can be a small integer (e.g. 5 or 10).)

- plus $O(M^3)$ for the bipartite matching. Since M is a small integer, the computations required for list kernel is not at all prohibitive in practice.

We now show that Algorithm 10 generates a valid kernel.

Proposition 8.3.1. *The function $K(x,y)$ in Algorithm 10 is symmetric, i.e. $K(x,y) = K(y,x)$.*

Proof.

$$K(x,y) \quad = \quad \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)} \cdot | < u_x^m, u_y^{a(m)} > |}{(||\lambda_x|| \cdot ||\lambda_y||)}$$

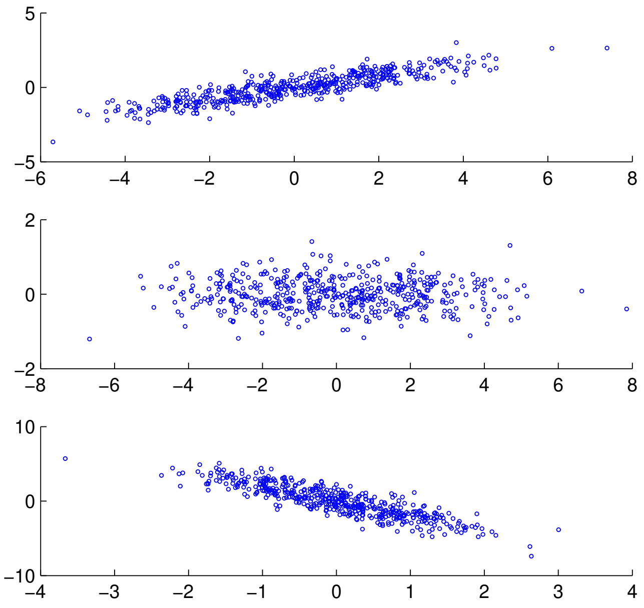


Figure 8.1: Illustration of list kernel. The top data is characterized a [.9 .3] vector as its first principal axes (large eigenvalue 5.2) and a [.3 -.9] vector as its second axes (small eigenvalue 0.1). The second and third datasets are rotations of the the first by 25 and 90 degrees, respectively. In the second dataset, the first principal axis is a [1 0] vector. In the third dataset, the first principal axis is a [.3 -.9] vector. The principal angles kernel would therefore find that the first and third data are close. However, the list kernel would successively discover via the maximum weighted bipartite matching procedure that the second dataset (which has less rotation) is closer to the first: it would match the axes that have both small cosine distance as well as large eigenvalues.

$$\begin{aligned} &= \frac{\sum_{m=1}^{M} \lambda_y^{a(m)} \lambda_x^m \cdot | < u_y^{a(m)}, u_x^m > |}{(||\lambda_y|| \cdot ||\lambda_x||)} \\ &= \frac{\sum_{m=1}^{M} \lambda_y^m \lambda_x^{a^{-1}(m)} \cdot | < u_y^m, u_x^{a^{-1}(m)} > |}{(||\lambda_y|| \cdot ||\lambda_x||)} \\ &= K(y,x) \end{aligned}$$

□

Proposition 8.3.2. *The function $K(x,y)$ in Algorithm 10 is satisfies the Cauchy-Schwartz Inequality, i.e. $K(x,y)^2 \leq K(x,x)K(y,y)$.*

Proof. First, we show that $K(x,x) = 1$:

$$\begin{aligned} K(x,x) &= \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_x^{a(m)} \cdot | < u_x^m, u_x^{a(m)} > |}{(||\lambda_x|| \cdot ||\lambda_x||)} \\ &= \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_x^m \cdot | < u_x^m, u_x^m > |}{(||\lambda_x|| \cdot ||\lambda_x||)} \\ &= \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_x^m}{(||\lambda_x|| \cdot ||\lambda_x||)} \\ &= \frac{||\lambda||^2}{(||\lambda_x|| \cdot ||\lambda_x||)} = 1 \end{aligned}$$

The second step follows from the fact that maximum bipartite matching would achieve $a(m) = m \, \forall m$ since $< u_x^m, u_x^m >= 1$ and $< u_x^m, u_x^{m'} >= 0$ for any $m \neq m'$. The third step is a result of $< u_x^m, u_x^m >= 1$.

Next we show that $K(x,y)^2$ is bounded by 1. Note that $< u_x^m, u_y^{a(m)} > \leq 1$, so that $K(x,y) \leq \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)}}{(||\lambda_x|| \cdot ||\lambda_y||)} \leq 1$ where the last inequality follows from applying Cauchy-Schwartz to the vectors of eigenvalues. □

Theorem 8.3.3 (Mercer's Theorem, c.f. [132]). *Every positive (semi) definite, symmetric function is a kernel: i.e., there exists a feature mapping ϕ such that it is possible to write: $K(x,y) =< \phi(x), \phi(y) >$.*

Mercer's Theorem is a powerful theorem which says that as long our function is positive semi-definite, we can be certain that there is an inherent (possibly high dimensional) feature representation whose dot product is the kernel function. We do not need to explicitly construct this feature space.

Proposition 8.3.4. *The function $K(x,y)$ in Algorithm 10 satisfies the Mercer Theorem.*

Proof. We have already proved that $K(x,y)$ is symmetric. To see that it is positive semi-definite, we just need to observe that $K(x,y) \geq 0$ for any x,y. We prove this by contradiction: Suppose $K(x,y) < 0$ for some x,y. This implies that $\sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)} \cdot | < u_x^m, u_y^{a(m)} > |$ is negative. However, by construction, we will only obtain non-negative eigenvalues λ_x from PCA. Further, the absolute value operation $| < u_x^m, u_y^{a(m)} > |$ ensures non-negativity. Thus, the statement that $K(x,y) < 0$ for some x,y is false. □

8.4 Importance Weighting with List Kernels

In this section, we explore a direct application of list kernels which is more similar to the standard classification scenario of covariate shift adaptation. The motivation is similar to the Importance Weighting Approach of the Local/Transductive Framework, as described in Section 6. The main difference is that we now focus on lists, rather than on pairs of objects within lists. The advantage with using lists as the atomic object is that list-based optimization methods, such as Minimum Error Rate Training (MERT), can be applied.

In Algorithm 11, we outline the procedure for using list kernels to implement the covariate shift assumption on lists. The idea is that if the test set contains lists all with a certain shape, then the training lists with similar shapes should be emphasized during training.

Algorithm 11 List-based Importance Weighting for Minimum Error Rate Training

```
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: Weight vector $w$, for testing on $E$.
1: For all pairs of lists in $S \cup E$, compute list kernel value $K(\cdot,\cdot)$
2: Apply the KLIEP algorithm on $L+U$ "samples", where each sample is a list, with the list kernel
   providing distance information. $L$ importance weights will be generated.
3: Train MERT with importance weights to obtain the adapted weight vector $w$ (Algorithm 12).
```

The traditional MERT algorithm is modified to allow importance weights as in Algorithm 12.

Algorithm 12 Minimum Error Rate Training with Importance Weights

```
Input: Train set $S = \{(q_l, \mathbf{d}_l, \mathbf{y}_l\}_{l=1..L}$
Input: Importance weights $v \in R^L$.
Output: Weight vector $w$, for testing on $E$.
1: Initialize $w$
2: for $i = 1..I$ (I is the number of features) do
3:     Perform line search along feature $i$ to find inflection points that cause hypothesis changes in $S$.
4:     For each inflection point, compute a weighted BLEU score, where the weights depend on the $v$.
5:     Choose $w_i$ that maximize the weighted BLEU score.
6: end for
7: Repeat for $J$ iterations
```

8.4.1 Evaluation in Machine Translation

In our evaluation, we compared four systems:

- MERT baseline (supervised)
- MERT with Importance Weights computed from Principal Angles Kernel
- MERT with Importance Weights computed from List Kernel
- Importance Weight with RankBoost in the Local/Transductive Framework (Section 6). There are two main differences between the Local/Transductive Framework and the techniques described in this section: (1) the Local/Transductive Framework computes new weights for each test list, whereas techniques in this section has one weight based on the entire set of test lists. (2) the KLIEP algorithm for Local/Transductive Framework is based on (difference) pairs of objects, whereas here it is based on lists.

For both Arabic and Italian datasets, we observe a small improvement over the MERT baseline with using List Kernel weights (e.g. 0.8 BLEU improvement). On the other hand, the Principal

Angles Kernel did not achieve significantly different results from the MERT baseline. Detailed examination of the importance weights reveal that they seldom differed from 1, which implies that there is little distributional difference between training and test lists from the perspective of the Principal Angles Kernel. The List Kernel, due to its more fine-grained characterization of overall shapes, could achieve better results by capturing slight differences between the distributions.

Furthermore, for the Arabic task, we observe an interesting result where the top-1 BLEU of List Kernel is better than that of the Importance Weighing Approach in the Local/Transductive Framework, but for Top-k, $k > 1$, the result is reversed. It is plausible that since the List Kernel allows us to use MERT, which directly optimizes top-1 BLEU, we could achieve the best top-1 result using this method. On the other hand, for higher k's, RankBoost may sometimes be superior to MERT. There are however too many differences between the two methods to firmly make this conclusion, yet we would just like to point out the possibility. Nevertheless, we observe that the List Kernel does indeed enable us to adapt new methods to ranking and give slight improvements. These improvements, however, are not statistically significant under the bootstrap test.

Table 8.2: Arabic-English MT results with Importance Weighting. Best results are underlined (no results were statistically significantly better).

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	24.3	26.5	27.9	28.6	29.3	47.9
Importance Weighting by Principal Angles Kernel	24.4	26.5	27.7	28.5	29.2	48.0
Importance Weighting by List Kernel	<u>25.1</u>	27.0	28.0	28.8	29.5	48.0
Importance Weight - Local/Transductive	24.6	<u>27.1</u>	<u>28.7</u>	<u>29.6</u>	<u>30.5</u>	<u>47.7</u>

8.4.2 Evaluation in Protein Structure Prediction

The evaluation for protein structure prediction is similar in setup as the machine translation tasks. In this case, there is a slight trend showing that List Kernel improves upon Principal Angles Kernel, but the difference between List Kernel and the Baseline are small.

Analysis of the importance weights from both Principal Angles Kernel and List Kernel reveal

Table 8.3: Italian-English MT results with Importance Weighting. Best results are underlined (no results were statistically significantly better).

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	21.2	23.1	24.3	25.0	25.7	52.6
Importance Weighting by Principal Angles Kernel	21.4	23.5	24.4	25.6	26.0	52.4
Importance Weighting by List Kernel	22.0	24.0	24.5	25.8	26.1	51.8
Importance Weight - Local/Transductive	21.9	24.7	25.8	25.9	26.5	51.5

Table 8.4: Protein Prediction GDT-TS results

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	.581	.590	.597	.601	.604
Importance Weighting by Principal Angles Kernel	.572	.584	.591	.596	.599
Importance Weighting by List Kernel	.581	.591	.597	.601	.602
Importance Weighting - Local/Transductive	.583	.596	.603	.605	.608

much sparsity, i.e. around 70%-90% of the weights were close to zero. This means that a significantly small set of training data was selected in the training process. Contrasting this with the machine translation results, we can say that:

- In Protein Prediction, there is an observable difference between the shapes of lists in the training set vs. the test set (thus leading to sparsity). In machine translation, this difference is too fined grained to be captured by the Principal Angles kernel–but it is captured by the List Kernel.

- In Protein Prediction, the List Kernel matches the Baseline result, which means that the small percentage of training data it selected correctly captures the characteristics of the dataset. However, it is insufficient to improve upon the Baseline. In contrast, in Machine Translation, there is a slight improvement obtained by List Kernels.

Table 8.5: Protein Prediction z-score results

Top-k z-score	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	1.07	1.17	1.26	1.31	1.34
Importance Weighting by Principal Angles Kernel	1.02	1.15	1.23	1.31	1.36
Importance Weighting by List Kernel	1.08	1.20	1.25	1.30	1.31
Importance Weighting - Local/Transductive	1.15	1.26	1.33	1.35	1.39

8.4.3 Evaluation in Information Retrieval

For information retrieval datasets, we evaluated the list kernel derived importance weights using the RankBoost algorithm (AdaCost version). RankBoost requires importance weights for pairs of documents, whereas now we have derived weights directly for each list. As a result, we simply tie each pair of documents to have the same weight as the list it comes from (i.e. all pairs in the same list receive the same weight; only pairs from different lists are weighted differently).

The result is shown in Table 8.6. In all datasets, we observe virtually no difference between the baseline RankBoost and the importance weight methods, either based on Principal Angles Kernel or List Kernel. Observation of the actual weight values reveal that almost all weights are valued at 1, meaning that very little data weighting is done in practice. This explains the lack of difference from Baseline results.

We conclude that importance weighting under the Local/Transductive framework produced positive results, though importance weighting under the current setup produced no discernable difference. We think this indicates that there is little distributional difference between the training set and the test set (which is possible under the data preparation conditions of LETOR, since queries were drawn randomly from the same set), though differences between the training set and *one* test list is large enough to be exploited. In addition, it is possible that the larger degree of freedom in pairs vs. list weighting could have an impact.

Table 8.6: Information Retrieval Results for List Kernel Importance Weighting. List Kernel and Principal Angles Kernel give virtually the same result as Baseline, due to the lack of deviation in the importance weights in practice.

	MAP	N@1	N@3	N@5	N@10	N@14
TREC'03						
Baseline (supervised RankBoost)	.2482	.3200	.3455	.3404	.3388	.3401
Importance Weight - Principal Angles	.2480	.3200	.3452	.3410	.3394	.3411
Importance Weight - List Kernel	.2490	.3200	.3455	.3414	.3378	.3411
Importance Weight - Local/Transductive	**.2932**	**.4800**	.3858	**.3862**	.3713	.3755
TREC'04						
Baseline (supervised RankBoost)	.3712	.4800	.4237	.4144	.4471	.4686
Importance Weight - Principal Angles	.3703	.4777	.4230	.4184	.4454	.4666
Importance Weight - List Kernel	.3700	.4790	.4242	.4153	.4433	.4690
Importance Weight - Local/Transductive	**.3834**	.4800	.4456	.4353	**.4653**	.4810
OHSUMED						
Baseline (supervised RankBoost)	.4424	.4906	.4543	.4501	.4230	.4218
Importance Weight - Principal Angles	.4420	.4901	.4534	.4512	.4224	.4218
Importance Weight - List Kernel	.4429	.4900	.4554	.4511	.4243	.4222
Importance Weight - Local/Transductive	.4440	.5000	.4483	.4466	**.4319**	**.4280**

Table 8.7: Comparison of Manifold Assumption for Classification and Ranking

	Manifold Assumption - Classification	Manifold Assumption - Ranking
Atomic object	A vector (sample)	A list (set of vectors)
Distance	Kernel defined between pairs of vectors	List Kernel
Smoothness	Labels vary slowly along manifold	Rankers vary slowly along manifold

8.5 Graph-based Methods with List Kernels

In this section, we explore another application of list kernels for semi-supervised ranking. In particular, we focus on utilizing the manifold assumption, common in graph-based methods. The manifold assumption (for classification) says that samples close together should receive similar labels. We will extend this manifold assumption to lists, to say that lists close to each other should be best ranked by similar rankers. In other words, the ranking function should vary smoothly over a manifold defined on lists. Table 8.7 compares the traditional manifold assumption for classification, and the version we extend to ranking.

We proposed a Ranker Propagation method for implementing the above Manifold Assumption. The idea is to train list-specific rankers for each list in the training set, and propagate the ranker parameters to the test lists using distance information (derived from list kernels). This is formalized in Algorithm 13.

Algorithm 13 Ranker Propagation

```
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: Rankers $\{w_u\}_{u=1..U}$, one for each test list
1: For all pairs of lists in $S \cup E$, compute list kernel value $K(\cdot,\cdot)$. This forms the basis of the
   underlying graph/manifold.
2: Compute Laplacian $L = D - K$, where $D_{ii} = \sum_j K_{ij}$ is the degree matrix
3: for l=1..L do
4:     Compute list-specific weights: $w_l = MERT(\mathbf{d}_l, \mathbf{y}_l)$
5:     Normalize weights: $w_l = \frac{w_l}{||w_l||}$
6: end for
7: Let $W_u = -L_{uu}^{-1} \times L_{ul} W_l$, where $W_l$ is the stacking of $w_l$ and $W_u$ is the stacking of $w_u$.
```

We can show that Step 7 of Algorithm 13 minimizes for

$$\sum_{ij} K_{ij} ||w_i - w_j||^2 \tag{8.3}$$

in a manner similar to Label Propagation [172], which optimizes

$$\sum_{ij} K_{ij} (y_i - y_j)^2 \tag{8.4}$$

See Appendix A for derivations. In these equations, K is a pairwise similarity measure, $w_i \in R^t$ is a vector (linear ranker), and $y_i \in R$ is a scalar (classification label). Note that our objective in Equation 8.3 essentially states if two lists have high similarity (i.e. high K_{ij}, then the rankers w_i and w_j should be similar in the 2-norm. The 2-norm is intuitive if we assume a linear ranker parameterized by the scoring function $w^T x$, so that the difference between scores using different rankers is $||w_i^T x - w_j^T x|| = ||(w_i - w_j)^T x||$.

Finally, we note that after the weight vectors are trained for each test list, we can rank the results and produce the final ranking outputs. It is important to distinguish that we are propagating rankers rather than the ranks themselves. The ranks are computed after the the ranker for each test list is determined. An illustration summarizing the manifold assumption and the Ranker Propagation method is shown in Figure 8.2.

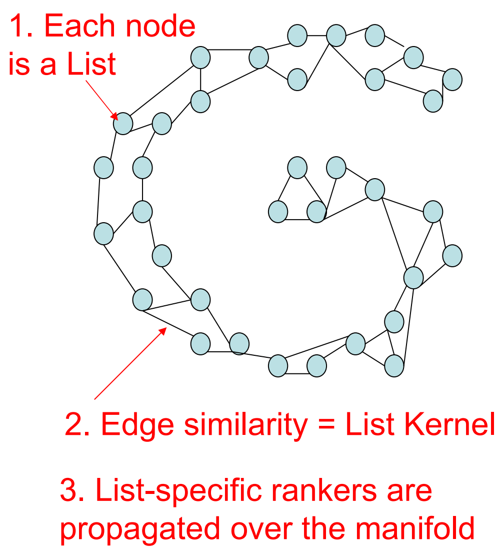

1. Each node is a List
2. Edge similarity = List Kernel
3. List-specific rankers are propagated over the manifold

Figure 8.2: Manifold Assumption and Ranker Propagation.

8.5.1 Evaluation in Machine Translation

In the following experiments, we compare the Ranker Propagation method with two baselines: supervised MERT and a random method where the ranker for a test list is randomly drawn from the set of training-list rankers.

We observe nice improvements with using Ranker Propagation. For example, on the Arabic-English MT task, BLEU improved by 1.3 points from 24.3 (baseline MERT) to 25.6 (Ranker Propagation). On the Italian-English MT task, BLEU improved by 1.1 points from 21.2 (baseline MERT) to 22.3 (Ranker Propagation). The Random Selection results were significantly below the MERT baseline.

We therefore conclude:

1. The List Kernel is effective in capturing distances between lists that lead to meaningful ranking functions.

2. In conjunction with the Ranker Propagation algorithm, the List Kernel achieves more than 1 point BLEU over the MERT baseline. The improvement of 25.6 over 24.3 (baseline) in Arabic translation is statistically significant (others are not).

3. The above improvement can be explained by the fact that Ranker Propagation fits more specific rankers individually to each list.

Table 8.8: Arabic-English MT results with Ranker Propagation. Statistically significant improvements are boldfaced; best but not statistically significant results are underlined.

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	24.3	26.5	27.9	28.6	29.3	47.9
Random Selection	22.7	25.2	26.4	27.4	28.0	50.0
Ranker Propagation	**25.6**	<u>27.3</u>	<u>28.4</u>	<u>29.4</u>	<u>30.0</u>	<u>47.5</u>

Table 8.9: Italian-English MT results with Ranker Propagation. Statistically significant improvements are boldfaced; best but not statistically significant results are underlined.

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
MERT (supervised) Baseline	21.2	23.1	24.3	25.0	25.7	52.6
Random Selection	19.3	22.4	23.6	24.1	24.3	54.1
Ranker Propagation	<u>22.3</u>	<u>23.9</u>	<u>25.4</u>	<u>26.0</u>	<u>26.2</u>	<u>50.9</u>

8.5.2 Evaluation in Protein Structure Prediction

The evaluation for Protein Structure Prediction is similar to Machine Translation. We again observe that Ranker Propagation with List Kernels outperform the baseline by a nice margin. For instance, the GDT-TS (k=1) of Ranker Propagation is .591, .010 point higher than the Baseline of .581. Correspondingly, the z-score improved from 1.07 (baseline) to 1.20 (Ranker Propagation). This is the best result thus far in Protein Structure Prediction in this work. All improvements in this case are statistically significant. We conclude that the manifold assumption is effective for the Protein Structure Prediction task.

Table 8.10: Protein Prediction GDT-TS results. Ranker Propagation gives statistically significant improvements over baseline supervised algorithm (Statistical significance is judged by the Wilcoxon signed rank test).

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	.581	.590	.597	.601	.604
Random Selection	.521	.549	.567	.582	.588
Ranker Propagation	**.591**	**.600**	**.605**	**.609**	**.612**

Table 8.11: Protein Prediction z-score results

Top-k z-score	k=1	k=2	k=3	k=4	k=5
MERT (supervised) Baseline	1.07	1.17	1.26	1.31	1.34
Random Selection	0.51	0.81	0.96	1.11	1.16
Ranker Propagation	**1.20**	**1.31**	**1.37**	**1.41**	**1.44**

8.5.3 Evaluation for Information Retrieval

In Information Retrieval, we use Rank SVM [82] as the base linear ranker. Each training list is optimized to obtain a linear weight vector, which is then propagated to the test lists via similarities defined by the List Kernel. Table 8.12 shows the results.

We observe that Ranker Propagation performed worse than the supervised Rank SVM baseline, possibly because the List Kernel did not give accurate similarities. In order to improve the reliability of the List Kernel, we therefore performed feature selection on the original features. In particular, we deleted features with weight values less than 0.1 according to the baseline Rank SVM. This corresponds to features that are less useful for ranking (they are about 10%-20% of the features).

List Kernel built on this reduced feature set gave improvements over the baseline. We therefore think that IR features may be noisy with respect to the ranking task, and these noisy features in particular may have high variance. These high variance but relatively useless features dominate the List Kernel computation, leading to degraded results. When these features are removed, List Kernel gave improvements. For example, in TREC'03, MAP improved from .2199 (baseline) to .2324; in

Table 8.12: Ranker Propagation for Information Retrieval. Ranker Propagation with Feature Selection outperforms both baseline and Ranker Prop with no feature selection. The Oracle result shows the accuracy if using Rank SVMs trained directly on the test lists.

	MAP	N@1	N@3	N@5	N@10	N@14
`TREC'03`						
Baseline (RankSVM)	.2199	.3600	.3358	.3236	.3134	.3132
RankerProp (NoSelection)	.1994	.3400	.3014	.3086	.2916	.2956
RankerProp (FeatureSelect)	**.2324**	.3700	.3480	.3273	.3198	.3186
Oracle	.7580	.8800	.8256	.8094	.7881	.7998
`TREC'04`						
Baseline (RankSVM)	.3618	.4467	.4167	.4062	.4164	.4077
RankerProp (NoSelection)	.3560	.4433	.4087	.3929	.4090	.4147
RankerProp (FeatureSelect)	.3683	**.4600**	.4237	**.4216**	.4190	**.4313**
Oracle	.8972	.9333	.9219	.9093	.9035	.9094
`OHSUMED`						
Baseline (RankSVM)	.4401	.4989	.4456	.4426	.4284	.4152
RankerProp (NoSelection)	.4149	.4048	.3768	.3619	.3589	.3627
RankerProp (FeatureSelect)	**.4453**	**.5191**	**.4789**	.4483	.4338	.4133
Oracle	.6636	.7170	.6825	.6588	.6505	.6512

OHSUMED, MAP improved from .4401 to .4453 (both statistically significant).

The Oracle results in Table 8.12 shows the potential improvement one may acquire if the test-specific weights were perfect, i.e. trained directly on the test list itself. The fact that there is large potential for improvement (even over the nice results of the current List Kernels) shows that improving upon the List Kernel may be a promising area of future work.

Chapter 9
OVERALL COMPARISONS AND CONCLUSIONS

This chapter summarizes the main conclusions and observations of this investigation and suggests future work.

9.1 Cross-Method Comparisons

In this section we will seek to compare all the previously proposed method. First, to summarize our methods:

- Under the Local/Transductive Framework, we proposed three main methods: Feature Generation (FG), Importance Weighting (IW), and Pseudo Margin (PM). In addition, we have a combination method of Feature Generation and Importance Weighting.

- Using List Kernels, we explored two methods: One is also based on the Importance Weighting Assumption. The other, Ranker Propagation, is based on the Manifold Assumption.

The overall results for Information Retrieval are presented in Tables 9.1 and 9.2. We observe that FG, IW, and FG+IW perform well. Ranker Propagation gave significant improvements over the Rank SVM baseline, but not necessarily the RankBoost baseline.

Tables 9.3 and 9.4 summarize the results for machine translation. We note that Ranker Propagation and Pseudo Margin gave the strongest improvements. This is followed by Importance Weighting (both the List Kernel and the Local/Transductive versions), which sometimes gave improvements (but is in general not statistically significant).

Tables 9.5 and 9.6 give the overall results for Protein Structure Prediction. We observe that Ranker Propagation is the only method that give statistically significant improvements. Importance Weighting gave slight improvements but it is not statistically significant. Other methods degraded the Baseline results (also not statistically significant, however).

A concise summary of results for all datasets is presented in Table 9.7

Table 9.1: Overall results for TREC. FG and IW approaches generally improved for all datasets. RankerProp outperformed the RankSVM baseline of which it is based (see Table 8.12) but does not always outperform the RankBoost baseline.

	MAP	N@1	N@3	N@5	N@10	N@14
`TREC'03`						
Supervised Baseline: RankBoost	.2482	.3200	.3455	.3404	.3388	.3401
Supervised Baseline: RankSVM	.2199	.3600	.3358	.3236	.3134	.3132
Feature Generation (FG)	**.3058**	**.5200**	.4332	**.4168**	.3861	**.3994**
Importance Weighting (IW)	**.2932**	**.4800**	.3858	**.3862**	.3713	.3755
Combined FG+IW	**.3219**	**.5250**	.4321	**.4138**	**.4023**	**.3990**
Pseudo Margin	.2502	.3400	.3399	.3500	.3403	.3433
Importance Weighting by List Kernel	.2490	.3200	.3455	.3414	.3378	.3411
RankerProp (FeatureSelect)	.2324	.3700	.3480	.3273	.3198	.3186
`TREC'04`						
Supervised Baseline: RankBoost	.3712	.4800	.4237	.4144	.4471	.4686
Supervised Baseline: RankSVM	.3618	.4467	.4167	.4062	.4164	.4077
Feature Generation (FG)	.3760	.4800	.4514	.4415	.4665	**.4910**
Importance Weighting (IW)	**.3834**	.4800	.4456	.4353	**.4653**	.4810
Combined FG+IW	**.3891**	**.4833**	.4487	**.4483**	.4554	.4873
Pseudo Margin	.3502	.4533	.4143	.4070	.4350	.4524
Importance Weighting by List Kernel	.3700	.4790	.4242	.4153	.4433	.4690
RankerProp (FeatureSelect)	.3683	.4600	.4237	.4216	.4190	.4313

Table 9.2: Overall results for OHSUMED.

	MAP	N@1	N@3	N@5	N@10	N@14
OHSUMED						
Supervised Baseline: RankBoost	.4424	.4906	.4543	.4501	.4230	.4218
Supervised Baseline: RankSVM	.4401	.4989	.4456	.4426	.4284	.4152
Feature Generation (FG)	.4444	.5094	.4787	.4600	**.4469**	**.4377**
FG + RankLDA features	**.4481**	**.5252**	.4785	.4600	.4444	.4390
Importance Weighting (IW)	.4440	.5000	.4483	.4466	**.4319**	**.4280**
Combined FG+IW	**.4497**	.5010	**.4897**	.4765	**.4431**	**.4422**
Pseudo Margin	**.4520**	.4560	.4342	.4334	.4208	.4196
Importance Weighting by List Kernel	.4429	.4900	.4554	.4511	.4243	.4222
RankerProp (FeatureSelect)	**.4453**	**.5191**	**.4789**	.4483	.4338	.4133

Table 9.3: Overall Arabic-English MT results.

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
Supervised Baseline: MERT	24.3	26.5	27.9	28.6	29.3	47.9
Supervised Baseline: RankBoost	23.7	26.4	28.0	28.9	29.6	47.9
Feature Generation (FG)	23.4	25.7	27.0	27.9	28.6	48.3
Importance Weighting (IW)	24.6	27.1	28.7	29.6	30.5	47.7
FG+IW	23.3	26.0	27.6	28.4	29.0	47.9
Pseudo Margin	**26.1**	**28.8**	**30.1**	**30.9**	**31.8**	**46.5**
Importance Weighting by List Kernel	25.1	27.0	28.0	28.8	29.5	48.0
Ranker Propagation	**25.6**	27.3	28.4	29.4	30.0	47.5

Table 9.4: Overall Italian-English MT results.

Top-k BLEU	k=1	k=2	k=3	k=4	k=5	PER
Supervised Baseline: MERT	21.2	23.1	24.3	25.0	25.7	52.6
Supervised Baseline: RankBoost	21.9	23.6	24.7	25.4	26.0	51.7
Feature Generation (FG)	21.5	23.4	24.4	24.9	25.3	52.5
Importance Weighting (IW)	21.9	24.7	25.8	25.9	26.5	51.5
FG+IW	21.6	23.5	25.0	24.9	25.6	52.5
Pseudo Margin	**24.3**	**26.1**	**27.0**	**27.8**	**28.4**	**48.6**
Importance Weighting by List Kernel	22.0	24.0	24.5	25.8	26.1	51.8
Ranker Propagation	22.3	23.9	25.4	26.0	26.2	50.9

Table 9.5: Overall GDT-TS Results for Protein Prediction

Top-k GDT-TS	k=1	k=2	k=3	k=4	k=5
Supervised Baseline: MERT	.581	.590	.597	.601	.604
Supervised Baseline: RankBoost	.579	.590	.595	.599	.604
Feature Generation (FG)	.569	.586	.596	.601	.605
Importance Weighting (IW)	.583	.596	.603	.605	.608
FG+IW	.568	.584	.593	.596	.601
Pseudo Margin	.574	.590	.599	.603	.608
Importance Weighting by List Kernel	.581	.591	.597	.601	.602
Ranker Propagation	**.591**	**.600**	**.605**	**.609**	**.612**

Table 9.6: Overall z-score Results for Protein Prediction

Top-k z-score	k=1	k=2	k=3	k=4	k=5
Supervised Baseline: MERT	1.07	1.17	1.26	1.31	1.34
Supervised Baseline: RankBoost	1.13	1.25	1.30	1.36	1.41
Feature Generation (FG)	1.07	1.24	1.33	1.40	1.41
Importance Weighting (IW)	1.15	1.26	1.33	1.35	1.39
FG+IW	1.02	1.23	1.29	1.32	1.39
Pseudo Margin	1.03	1.24	1.34	1.40	1.47
Importance Weighting by List Kernel	1.08	1.20	1.25	1.30	1.31
Ranker Propagation	**1.20**	**1.31**	**1.37**	**1.41**	**1.44**

9.2 Summary of Contributions

We present one of the first studies that investigate ranking problems in the context of semi-supervised learning. Drawing inspirations from related work in semi-supervised classification and domain adaptation, we investigated several assumptions for which unlabeled data may be helpful:

- Change of Representation Assumption: use unlabeled data to generate more salient features
- Covariate Shift Assumption: use unlabeled (test) data to discover the training samples that are similar in distribution to the test samples and place corresponding weights on the training data to correct for the bias.
- Low Density Separation Assumption: use unlabeled data to discover low density regions, which are to be avoided by the ranking function.
- Manifold Assumption: use unlabeled data to discover local similarities and manifold structure, which can be exploited for smoothness regularization.

Two main algorithmic contributions are introduced. First, the Local/Transductive Meta-algorithm allows us to implement the first three assumptions, leading respectively to the Feature Generation

Table 9.7: Summary of Results. + indicates improvement over baseline, - indicates degradation. = indicates similar results. ++ indicates the best method for a given dataset.

	Information Retrieval	Machine Translation	Protein Prediction
Feature Generation	+	-	=
Importance Weight (Local/Transductive)	+	=	=
FG+IW	++	-	=
Pseudo-Margin	=	++	=
Importance Weight (List Kernel)	=	=	=
Ranker Propagation	=	+	++

method, the Importance Weighting method, and the Pseudo-margin method. Second, a novel List Kernel was developed, which enabled examination of the Manifold Assumption.

We performed experiments on a total of six different real-world datasets, which come from Information Retrieval, Machine Translation, and Computational Biology (Protein Structure Prediction). We observe that different methods perform well for different datasets. Though it is difficult to judge in advance which method works best for which kind of dataset, our analysis of the results give the following guidelines:

- How well does Pairwise Accuracy correlate with the final evaluation metric (e.g. MAP, BLEU, GDT-TS)? If the correlation is relatively strong, Feature Generation methods may benefit because there are more features to optimize with. Otherwise, Feature Generation may overfit due to the larger feature space. Also important for Feature Generation is the question of whether the assumption made by PCA (that high variance features are important) is valid with the ranking task.

- How often do tie ranks occur in lists? If ties occur often, then Pseudo Margin is not a valid approach because the low density separation assumption is violated.

- Importance Weighting (especially the Local/Transductive version) is a relatively risk-free method. It either performs better or equal to the baseline and rarely rarely degrades results. For any dataset where we believe there might be slight differences within each list, Importance Weighting is a recommended approach.

- Ranker Propagation appears to be a method that improves results for all datasets. In particular, datasets with smaller feature sets (less than 25) seem to benefit more.

9.3 Future Work

9.3.1 Computational Speedup for the Local/Transductive Framework

The Local/Transductive Framework requires training during test time. Therefore, computational speed is an issue if semi-supervised ranking were to be deployed in practical systems. For example, in our Feature Generation experiments which are run on an 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.

To achieve near real-time computation, a combination of better code optimization, distributed computation, and algorithmic improvements are needed. We will highlight one algorithmic idea here based on caching: In the context of local learning in ranking, [62] has shown that offline solutions empirically approximate the accuracies of online k nearest neighbors. Therefore one algorithmic solution is to precompute a set of RankBoost rankers and select the “best” one during test time. For example, for Importance Weighting, multiple rankers can be pre-computed from various random weighting of the training set; these are then matched to test lists during query time by a fast distribution matching procedure (c.f. [65]). For Feature Generation, we would precompute a variety of RankBoost using Kernel PCA features generated from different subsets of the training or development set. During test time, we would then match the basis vector of the test list with those in the training/development set using, e.g., the List Kernel as distance measure.

The performance of these solutions will be characterized by a speed vs. MAP/NDCG trade-off that depends on the granularity of pre-computed components. If there are more pre-computed rankers, then the matching operation at query time will become more expensive, but the results can be more suitable for the test list.

9.3.2 Nonlinear extensions to Ranker Propagation

The current Ranker Propagation method assumes that linear weights (which represent a linear ranker) are smoothly varying across the manifold. This linear assumption allows for a straightforward and computationally efficient propagation algorithm. However, in cases where linear rankers are not sufficient[1], we may desire an algorithm that propagates nonlinear or kernelized rankers (e.g. RankBoost or RankSVM with nonlinear kernels).

The first challenge would be the characterization of "smoothness" for non-linear rankers. Smoothness is easy to quantify in the linear case using L2 norm, because L2 differences in linear weights has a correspondence to changes in ranking on a list. In other words, imagine two identical linear rankers: as we gradually adjust one of the weights of one ranker, the corresponding ordering of items will change in a monotonic way. On the other hand, in order to quantify the difference in orderings achieved by two non-linear rankers, we would likely need to compute the ordering (rather than directly comparing ranker parameters, as done in the linear case). In other words, we would like to minimize an objective like the following:

$$\sum_{ij} K_{ij}(D(\sigma(F_i(x_i)), \sigma(F_j(x_i))) + D(\sigma(F_i(x_j)), \sigma(F_j(x_j))) \tag{9.1}$$

where $\sigma(F_j(x_i))$ represents the permutation of objects in x_i under the ranker F_j and $D(\cdot)$ is some divergence between two permuations (or simply a rank-based evaluation metric such as MAP). In other words, we apply two neighboring rankers to the two respective lists and want to ensure that the final permutations will be similar. Depending on the form of F and D, this may present both interesting computational and algorithmic challenges.

[1]In our datasets, linear rankers appear to be sufficiently expressive. Note that we are fitting a separate linear ranker to each list, as opposed to one linear ranker for the entire dataset.

9.3.3 Different formulations of the List Kernel

Our List Kernel attempts to quantify the shape/orientation similarity between lists and is based on geometric properties, e.g. principal component axes. An underlying assumption with using principal component analysis is that of Gaussianity. Therefore in this respect, our List Kernel also has similarities to the Bhattacharyya kernel of [90] with assumes Gaussianity.

In general, it would be interesting to explore different types of list kernels under the application of ranking. We described several properties, such as shift-invariant, scale-invariant, and rotation-invariant, and it would be worthwhile to investigate which ones would be desirable in certain applications.

An additional avenue of future research is kernel learning (c.f. [96]). One could first define a list kernel that is parameterized. Then, we could learn the best parameters based on our (labeled) data which indicates which two lists should be similar. For example, we could train rankers on individual labeled lists, measure how different the rankers are, and use the resulting distance to learn the kernel. This kernel can then be applied to any unlabeled data.

9.3.4 Inductive semi-supervised ranking algorithms

All methods presented in this work are transductive in the sense that the test data is required. In the Local/Transductive Framework, we depend on observing one test list. In Ranker Propagation and Importance Weighting with List Kernels, we need the entire test set. It would be worthwhile to develop algorithms that exploit *any* unlabeled data, in particular those that are not the test set.

We note that not all methods should be extended to inductive algorithms, however. For example, the entire motivation of Importance Weighting rests on the assumption that there is a distribution difference between training and test, so it is natural to operate transductively. However, for the Feature Generation and Pseudo Margin approaches, it is possible to imagine unlabeled non-test dataset giving exploitable information. These inductive methods would need to be developed outside of the Local/Transductive Framework.

For the Ranker Propagation Method, there are ways to convert it to an inductive method and extend to out-of-sample lists, either by a k-nearest neighbor search (using List Kernel distances) when a new test list arrives, or a warping of the function space to match that of the manifold [136].

A brief sketch of the k-nearest neighbor search solution for Ranker Propagation is as follows:

1. Construct graph on labeled and unlabeled (non-test) lists.

2. Apply Ranker Propagation to obtain rankers on unlabeled (non-test) lists

3. When a test list arrives, perform k-nearest neighbor search on all labeled and unlabeled lists.

4. Each ranker within the neighborhood provides a ranking to the test list.

5. The ranking are aggregated using consensus ranking techniques or ideas from social choice theory. See, e.g. [114, 88].

9.3.5 Theory for the proposed methods

We have empirically examined the proposed methods under several datasets and provided some intuitions of "what method works for what kind of dataset" in Section 9.2. In addition to these empirical observations, it would be worthwhile to investigate theoretically how each method can help, using techniques from statistical learning theory (c.f. [97, 149]).

While general theories and bounds for semi-supervised learning (e.g. [11]) may be difficult to derive and challenging to fit to real-world scenarios, algorithm-specific theories may be possible. In other words, bounds tailored to specific methods such as Feature Generation or Importance Weighting may be practically useful and insightful. For example, one promising direction is the work on analysis of representations by [14]. This theory aims to explain the tradeoff in designing feature representations for domain adaptation problems. Briefly, the generalization bounds indicate that a good feature representation should *simultaneously* achieve low error on the source (training) domain while minimizing the "distance" between the induced marginal distributions of the two domains[2]. Our Feature Generation Approach may be analyzed under this theory or variant thereof: for each test list, we can compute whether the the new Kernel PCA features reduce training error while minimizing the distance between the test distribution and the training distribution. We have already

[2]One of the main contributions of this work is a definition of a distance over distributions that is both practically computable and meaningful for representing changes in representation.

observed that training error is minimized in both Information Retrieval and Machine Translation experiments. If we also observe that the distribution distances decreased for Information Retrieval, but increased for Machine Translation, then we would have a theoretical confirmation for why the same method worked for one dataset but not another.

In summary, this dissertation presents one of the first comprehensive evaluation of several approaches for semi-supervised ranking. We have observed that different methods work well for different datasets, and have provided empirical analyses of the reasons. A significant step forward in future work would involve algorithm-specific bounds that seek to explain the results in a theoretically-motivated way.

BIBLIOGRAPHY

[1] S. Abney. Understanding the Yarowsky algorithm. *Computational Linguistics*, 30(3), 2004.

[2] S. Agarwal. Ranking on graph data. In *ICML*, 2006.

[3] N. Ailon. Reconciling real scores with binary comparisons: A unified logistic model for ranking,. In *NIPS*, 2008.

[4] A. Alexandrescu and K. Kirchhoff. Data-driven graph construction for semi-supervised graph-based learning in NLP. In *Proc. of NAACL Human Language Technologies*, 2007.

[5] Y. Altun, D. McAllester, and M. Belkin. Maximum margin semi-supervised learning for structured variables. In *Proceedings of Neural Information Proccessing Systems*, 2005.

[6] 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.

[7] R. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR*, 2005.

[8] A. Argyriou, M. Herbster, and M. Pontil. Combining graph laplacians for semi-supervised learning. In *NIPS*, 2005.

[9] K. Arrow. *Social Choice and Individual Values*. Yale University Press, 2nd ed., 1970.

[10] D. Baker. Protein structure prediction and structural genomics. In *Science*, volume 294, pages 93–96. American Association for the Advancement of Science, 2001.

[11] M. Balcan and A. Blum. A pac-style model for learning from labeled and unlabeled data. In *Proceedings of Computational Learning Theory*, 2005.

[12] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: a geometric framework for learning from examples. Technical report, University of Chicago, 2004.

[13] M. Belkin, P. Niyogi, and V. Sindhwani. On manifold regularization. In *AISTAT*, 2005.

[14] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *NIPS*, 2006.

[15] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *NIPS 12*, pages 368–374, 1998.

[16] K. P. Bennett, A. Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. In *Proc. of SIGKDD-2002*, Edmonton, Alberta, 2002.

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

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

[19] 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.

[20] 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.

[21] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of Computational Learning Theory*, 1998.

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

[23] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, 2004.

[24] S. Brin and L. Page. The anatomy of large scale hypertextual web search engine. In *Proc. of the 7th International World Wide Web Conference*, 1998.

[25] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[26] C. Burges, R. Ragno, and Q. Le. Learning to rank with non-smooth cost functions. In *NIPS*, 2006.

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

[28] L. Busse, P. Orbanz, and J. Buhmann. Cluster analysis of heterogenous rank data. In *ICML*, 2007.

[29] D. Cai, X. He, and J. Han. Semi-supervised discriminant analysis. In *ICCV*, 2007.

[30] C. Callison-Burch, D. Talbot, and M. Osborne. Statistical machine translation with word- and sentence-aligned parallel corpora. In *Proc. ACL 2004*, 2004.

[31] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y.-L. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *SIGIR*, 2006.

[32] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise to listwise approach. In *SIGIR*, 2007.

[33] V. Castelli and T. M. Cover. On the exponential value of labeled samples. *Pattern Recogn. Lett.*, 16(1):105–111, 1995.

[34] 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*, 1996.

[35] O. Chapelle, V. Sindhwani, and S. Keerthi. Branch and bound for semi-supervised support vector machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

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

[37] O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proc. of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[38] 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.

[39] D. Chivian and D. Baker. Homology modeling using parametric alignment ensemble generation with consensus and energy-based model selection. *Nucleic Acids Research*, 34(17), 2006.

[40] W. Chu and Z. Ghahramani. Extension of gaussian processes for ranking. In *NIPS Workshop on Learning to Rank*, 2005.

[41] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 1999.

[42] M. Collins. Discriminative reranking for natural language processing. In *ICML*, 2000.

[43] M. Collins and T. Koo. Discriminative reranking for natural langauge parsing. *Computational Linguistics*, 31(1), 2005.

[44] R. Collobert, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proc. of the International Conference on Machine Learning (ICML)*, 2006.

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

[46] A. Corduneanu and T. Jaakkola. Stable mixing of complete and incomplete information. Technical report, CSAIL, MIT, 2001.

[47] A. Corduneanu and T. Jaakkola. On information regularization. In *Proceedings of Uncertainty in Artificial Intelligence*, 2003.

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

[49] D. Cossock and T. Zhang. Subset ranking using regression. In *COLT*, 2006.

[50] N. Craswell and D. Hawking. Overview of the trec 2003 web track. In E. M. Voorhees and L. P. Buckland, editors, *NIST Special Publication 500-255:The Twelfth Text REtrieval Conference (TREC 2003)*. NIST, 2003.

[51] F. d'Alche Buc, Y. Grandvalet, and C. Ambroise. Semi-supervised marginboost. In *NIPS*, 2002.

[52] O. Dekel, C. Manning, and Y. Singer. Log-linear models for label ranking. In *NIPS*, 2004.

[53] A. P. Dempster, N. M. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1(39):1–38, 1977.

[54] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR*, 2008.

[55] D. Eramian, M. Shen, D. Devos, F. Melo, A. Sali, and M. Marti-Renom. A composite score for predicting errors in protein structure models. *Protein science*, 2006.

[56] 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*, pages 97–105. Morgan Kaufmann, 1999.

[57] M. Fligner and J. Verducci. *Probability Models and Statistical Analyses for Ranking Data*. Springer, 1993.

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

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

[60] A. Fujino, N. Ueda, and K. Saito. A hybrid generative/discriminative approach to semi-supervised classifier design. In *Proc. of AAAI*, 2005.

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

[62] 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.

[63] 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 *Proc. 6th conference on Natural language learning (CoNLL)*, pages 1–7, 2002.

[64] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Proceedings of Neural Information Proccessing Systems*, 2004.

[65] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *SODA*, 2006.

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

[67] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN*, 1999.

[68] W. R. Hersh, C. Buckley, T. J. Leone, and D. H. Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual ACM SIGIR Conference*, 1994.

[69] T. Hertz, A. Bar-Hillel, and D. Weinshall. Learning a kernel function for classification with small training samples. In *ICML*, 2006.

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

[71] H. Hotelling. Relationships betwen two sets of variates. *Biometrika*, 28:321–372, 1936.

[72] J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *JMLR*, 2009.

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

[74] Z. Huang, M. Harper, and W. Wang. Mandarin part-of-speech tagging and discriminative reranking. In *EMNLP*, 2007.

[75] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *NIPS*, 1998.

[76] T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in Neural Information Processing Systems*, 1999.

[77] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[78] H. Ji, C. Rudin, and R. Grishman. Re-ranking algorithms for name tagging. In *NAACL Workshop on Computationally Hard Problems and Joint Inference*, 2006.

[79] J. Jiang. A literature survey on domain adaptation of statistical classifiers; url: http://sifaka.cs.uiuc.edu/jiang4/domainadaptation/survey/, 2008.

[80] F. Jiao, S. Wang, C.-H. Lee, R. Greiner, and D. Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *COLING/ACL*, pages 209–216, Sydney, Australia, July 2006. Association for Computational Linguistics.

[81] T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning*, 1999.

[82] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

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

[84] T. Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

[85] A. Kapoor, Y. Qi, H. Ahn, and R. Picard. Hyperparameter and kernel learning for graph-based semi-supervised classification. In *NIPS*, 2005.

[86] K. Kirchhoff and M. Yang. The university of washington machine translation system for the iwslt 2007 competition. In *IWSLT*, 2007.

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

[88] A. Klementiev, D. Roth, K. Small, and I. Titov. Unsupervised rank aggregation with domain specific expertise. In *IJCAI*, 2009.

[89] P. Koehn et al. Moses: open source toolkit for statistical machine translation. In *ACL*, 2007.

[90] R. Kondor and T. Jebara. A kernel between sets of vectors. In *ICML*, 2003.

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

[92] T. Kudo, J. Suzuki, and H. Isozaki. Boosting-based parse reranking with subtree features. In *ACL*, 2005.

[93] J. Lafferty and L. Wasserman. Challenges in statistical machine learning. *Statistica Sinica*, 16(2), 2006.

[94] J. Lafferty and L. Wasserman. Statistical analysis of semi-supervised regression. In *NIPS*, 2007.

[95] J. Lafferty, X. Zhu, and Y. Liu. Kernel conditional random fields: Representation and clique selection. In *Proceedings of International Conference on Machine Learning*, 2004.

[96] G. Lanckriet, N. Cristianini, P. Bartlett, L. Ghaoui, and M. Jordan. Learning the kernel matrix with semi-definite programming. *Journal of Machine Learning Research*, 5, 2004.

[97] J. Langford. Tutorial on practical prediction theory for classification. *JMLR*, 2005.

[98] A. Lavie and A. Agarwal. METEOR: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Workshop on Statistical Machine Translation*, 2007.

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

[100] Q. Le and A. Smola. Direct optimization of ranking measures. Technical report, NICTA, 2007.

[101] G. Lebanon and J. Lafferty. Cranking: combining rankings using conditional probability models on permuations. In *ICML*, 2002.

[102] C.-H. Lee, S. Wang, F. Jiao, D. Schuurmans, and R. Greiner. Learning to model spatial dependency: semi-supervised discriminative random fields. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[103] B. Leskes. The value of agreement: A new boosting algorithm. In *Proceedings of Computational Learning Theory*, 2005.

[104] P. Li, C. Burges, and Q. Wu. Learning to rank using classification and gradient boosting. Technical Report MSR-TR-2007-74, Microsoft Research, 2007.

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

[106] P. Liang, A. Bouchard-Cote, D. Klein, and B. Taskar. An end-to-end discriminative approach to machine translation. In *ACL*, 2006.

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

[108] A. Lopez. Statistical machine translation. *ACM Computing Surveys*, 40(3), 2008.

[109] D. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover, 1989.

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

[111] J. I. Marden. *Analyzing and Modeling Rank Data*. Chapman & Hall/CRC, 1996.

[112] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting as gradient descent. In *NIPS*, 2000.

[113] P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman and Hall, London, 1989.

[114] M. Meila, K. Phadnis, A. Patterson, and J. Bilmes. Consensus ranking under the exponential model. Technical report, UW Statistics, 2008.

[115] D. Metzler. Direct maximization of rank-based metrics. Technical report, Univ. of Massachusetts, Amherst, CIIR, 2006.

[116] J. Moult, F. Kryzysztof, B. Rost, T. Hubbard, and A. Tramontano. Critical assessment of methods of protein structure prediction (casp) - round 6. *Proteins*, 61(S7):3–7, 2005.

[117] K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *CIKM*, 2000.

[118] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 30(3), 2000.

[119] F. Och. Minimum error rate training in statistical machine translation. In *ACL*, 2003.

[120] F. Och and H. Ney. Discriminative and maximum entropy models for statistical machine translation. In *ACL*, 2002.

[121] 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.

[122] K. Papineni, S. Roukos, ToddWard, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *ACL*, 2002.

[123] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W. Y. Ma. A study of relevance propagation for web search. In *SIGIR*, 2005.

[124] J. Qiu, W. Sheffler, D. Baker, and W. Noble. Ranking protein structures with support vector regression. *Proteins: Structure, Function, and Bioinformatics*, 2007.

[125] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data,. In *ICML*, 2007.

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

[127] A.-V. I. Rosti, N. F. Ayan, B. Xiang, S. Matsoukas, R. M. Schwartz, and B. J. Dorr. Combining outputs from multiple machine translation systems. In *NAACL-HLT*, 2007.

[128] C. Rudin. Ranking with a p-norm push. In *COLT*, 2006.

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

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

[131] A. Shashua and A. Levin. Taxonomy of large margin principle algorithms for ordinal regression. In *NIPS*, 2002.

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

[133] L. Shen, A. Sarkar, and F. Och. Discriminative reranking for machine translation. In *HLT-NAACL*, 2004.

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

[135] V. Sindhwani, S. Keerthi, and O. Chapelle. Deterministic annealing for semi-supervised kernel machines. In *Proc. of the International Conference on Machine Learning (ICML)*, 2006.

[136] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: from transductive to semi-supervised learning. In *ICML*, 2005.

[137] A. Singh, R. Nowak, and X. Zhu. Unlabeled data: Now it helps, now it doesn't. In *NIPS*, 2008.

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

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

[140] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. A study of translation edit rate with targeted human annotation. In *AMTA*, 2006.

[141] 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.

[142] M. Szummer and T. Jaakkola. Information regularization with partially labelled data. In *Proceedings of Neural Information Proccessing Systems*, 2002.

[143] T. Takezawa, E. Sumita, F. Sugaya, H. Yamamoto, and S. Yamamoto. Toward a broad-coverage bilingual corpus for speech translation of travel conversation in the real world. In *LREC*, 2002.

[144] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of International Conference on Machine Learning*, 2005.

[145] 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.

[146] J. Tsai, R. Bonneau, A. V. Morozov, B. Kuhlman, C. A. Rohl, and D. Baker. An improved protein decoy set for testing energy functions for protein structure prediction. *Proteins*, 53(76), 2003.

[147] 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.

[148] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of International Conference on Machine Learning*, 2004.

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

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

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

[152] T. Watanabe, J. Suzuki, H. Tsukada, and H. Isozaki. Online large-margin training for statistical machine translation. In *EMNLP-CoNLL*, 2007.

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

[154] L. Wolf and A. Shahsua. Learning over sets using kernel principal component angles. *JMLR*, 2003.

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

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

[157] 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.

[158] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. In *SIGIR*, 2007.

[159] J. Xu, L. Yu, and M. Li. Consensus fold recognition by predicted model quality. In *Proc. of the 3rd Asia-Pacific Bioinformatics Conference*, 2005.

[160] L. Xu and D. Schuurmans. Unsupervised and semi-supervised multi-class support vector machines. In *AAAI*, 2005.

[161] O. Yamaguchi, K. Fukui, and K. Maeda. Face recognition using temporal image sequence. In *IEEE International Conference on Automatic Face & Gesture Recognition*, 1998.

[162] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*, 1995.

[163] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimization average precision. In *SIGIR*, 2007.

[164] A. Zemla. LGA: a method for finding 3d similarities in protein structures. *Nucleic Acids Research*, 31:3370–3374, 2003.

[165] 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.

[166] T. Zhang and F. J. Oles. A probability analysis on the value of unlabeled data for classification problems. In *Proc. 17th International Conference on Machine Learning (ICML-2000)*, San Franscisco, CA, 2000. Morgan Kaufmann.

[167] Y. Zhang, S. Vogel, and A. Waibel. Interpreting bleu/nist scores: How much improvement do we need to have a better system? In *LREC*, 2004.

[168] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *NIPS*, 2004.

[169] K. Zhou, G.-R. Xue, H. Zha, and Y. Yu. Learning to rank with ties. In *SIGIR*, 2008.

[170] 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.

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

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

[173] X. Zhu, Z. Ghahramani, and J. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *NIPS*, 2005.

Appendix A
RANKER PROPAGATION OBJECTIVE FUNCTION

Algorithm 13 in Section 8.5 describes an method for propagating rankers across a graph defined on lists. Here we show that the algorithm minimizes the following objective:

$$\sum_{ij} K_{ij}||w_i - w_j||^2 \tag{A.1}$$

where K is a pairwise similarity measure and $w_i \in R^t$ is a vector representing a ranker situated on list i. The sum ij is over all pairs of lists. The objective essentially states that if two lists are close by K_{ij}, then we want the difference $||w_i - w_j||^2$ between their rankers to be small in the L2 sense.

To begin, we rewrite the objective in order to optimize each dimension separately:

$$\begin{aligned}
\sum_{ij} K_{ij}||w_i - w_j||^2 &= \sum_{ij} K_{ij}||w_i - w_j||^2 \\
&= \sum_{ij} K_{ij} \sum_t (w(t)_i - w(t)_j)^2 \\
&= \sum_t \sum_{ij} K_{ij}(w(t)_i - w(t)_j)^2
\end{aligned}$$

where t indexes a feature within the ranker vector w_i (i.e. $w(t)_i$ is the t-th parameter of the ranker for list i). Next, we proceed with a derivation similar to Label Propagation [172]:

$$\begin{aligned}
\sum_t \sum_{ij} K_{ij}(w(t)_i - w(t)_j)^2 &= \sum_t \sum_{ij} (K_{ij} w(t)_i^2 + w(t)_j^2 - 2w(t)_i w(t)_j) \\
&= \sum_t (\sum_{ij} K_{ij} w(t)_i^2 + \sum_{ij} K_{ij} w(t)_j^2 - 2\sum_{ij} K_{ij} w(t)_i w(t)_j) \\
&= \sum_t (\sum_i (w(t)_i^2 \sum_j K_{ij}) + \sum_j (w(t)_j^2 \sum_i K_{ij}) - 2\sum_{ij} K_{ij} w(t)_i w(t)_j) \\
&= \sum_t (2\sum_i (w(t)_i^2 \sum_j K_{ij}) - 2\sum_{ij} K_{ij} w(t)_i w(t)_j) \\
&= 2\sum_t (\sum_i (w(t)_i^2 D_{ii}) - \sum_{ij} K_{ij} w(t)_i w(t)_j)
\end{aligned} \tag{A.2}$$

$$
\begin{aligned}
&= 2\sum_t (W(t)^T DW(t) - W(t)^T KW(t)) && \text{(A.3)} \\
&= 2\sum_t (W(t)^T (D-K)W(t)) && \\
&= 2\sum_t (W(t)^T LW(t)) && \text{(A.4)}
\end{aligned}
$$

Step A.2 above follows from the definition $D_i i = \sum_j K_{ij}$. Step A.3 is a simple rewrite in terms of matrices, i.e., D is a diagonal matrix with $D_i i$ on the diagonal, and $W(t)$ is a matrix containing a stacking of $w(t)_i$ for all i. In Step A.4 we see the graph Laplacian $L = D - K$. Step A.4 is a concave function with respect to $W(t)$.

In order to minimize A.4 with respect to $W(t)$ for all t, we take the first derivative and set it to zero:

$$
\begin{aligned}
\frac{\partial 2\sum_t (W(t)^T LW(t))}{\partial W(t)} &= 4LW(t) \\
LW(t) &= 0
\end{aligned}
$$

In order to optimize only on weights on unlabeled lists, while keeping weights on labeled lists fixed, we write the Laplacian and weight matrices to separate the labeled and unlabeled parts:

$$
\begin{pmatrix} L_{ll} & L_{lu} \\ L_{ul} & L_{uu} \end{pmatrix} \begin{pmatrix} W(t)_l \\ W(t)_u \end{pmatrix} = 0
$$

Finally we obtain the equation:

$$
\begin{aligned}
L_{ul}W(t)_l + L_{uu}W(t)_u &= 0 \\
W(t)_u &= -inv(L_{uu}) * L_{ul}W(t)_l && \text{(A.5)}
\end{aligned}
$$

For each dimension of the ranker, we calcuate Equation A.5 to obtain the weight values for that dimension for all unlabeled lists. Note that since we do not impose constraints on the weight vectors for unlabeled lists, we can calculate the weights for each dimension separately, all in closed-form. If we were to impose constraints (e.g. forcing the weights on unlabeled list to normalize), then we would alternatively optimize the objective using, e.g., a projected subgradient method [23].

VITA

Kevin K. Duh received his Bachelor of Science in Electrical and Computer Engineering from Rice University in 2003. Thereafter he worked briefly at NTT Labs in Japan, and in 2004 he joined the Signals, Speech, and Language Interpretation (SSLI) Lab at the University of Washington. He received his Master of Science and Doctor of Philosophy degrees in Electrical Engineering from the University of Washington in 2006 and 2009, respectively. While a graduate student, he served as co-chair for the Student Research Workshop of the Association for Computational Linguistics (ACL) in 2006 and the Semi-supervised Learning for Natural Language Processing Workshop of the North American Association for Computational Linguistics (NAACL) in 2009. He was awarded the National Science Foundation Graduate Research Fellowship from 2005 to 2008.