Abstract

Learning to Rank with Partially-Labeled Data
(Computational Biology Excerpt)

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 Representation, 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 positively that semi-supervised ranking is possible, and improvements over the supervised baseline can be achieved under different dataset-method combinations.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter 1:</th>
<th>Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 Problem Formulation</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.3 Dataset</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2:</th>
<th>A Local/Transductive Framework for Ranking</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1 Description of Local/Transductive Framework</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.2 Investigating the Change of Representation Assumption</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.3 Investigating the Covariate Shift Assumption</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.4 Investigating the Low Density Separation Assumption</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.5 Evaluation in Protein Structure Prediction</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3:</th>
<th>Kernels on Lists</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.1 Motivation</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>3.2 Related Work on Kernels</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.3 Formulation of a List Kernel</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.4 Graph-based Methods with List Kernels</td>
<td>18</td>
</tr>
</tbody>
</table>

| Chapter 4: | Conclusions | 22 |

| Bibliography | 24 |
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. 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?

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

3. What can we learn by comparing the same semi-supervised ranking algorithm on different kinds of real-world datasets?
Figure 1.1: 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.

1.2 Problem Formulation

There are generally two interpretations of “learning to rank with partially-labeled data.” See Figure 1.1 for a pictorial comparison with information retrieval as an example. In the scenario we consider in this work, the document lists in our dataset are either fully labeled or not labeled at all. There are many possible search queries in the world, and we cannot possibly have labels for all of them. 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 do not. This second problem can arise when, e.g. the document list retrieved by one query is too long and the annotator can only label a few documents. Correspondingly for Protein Structure Prediction, the first problem may arise because there are some unknown proteins which we have a list of candidates; the second problem may arise when we additionally have GDT-TS values for some (but not all) of the candidates in the same list. We are interested in the first problem, which has received little attention so far.
1.3 Dataset

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. In this work, we follow the framework set up by [10]. 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 annual CASP (Critical Assessment of Methods of Protein Structure Prediction) evaluation [8]. 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 training data consists of the predicted structures submitted to CASP5 and CASP6 evaluations. The native protein structures from PDB used in the training set of [10] is not included here since we our training algorithms are based on ranking. The test set consists of proteins to be predicted for the CASP7 challenge.

A brief summary of data statistics is shown in Table 1.1. 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 [10].

<table>
<thead>
<tr>
<th></th>
<th>TRAINING SET</th>
<th>TEST SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>#lists, i.e. number of proteins to predict</td>
<td>73</td>
<td>99</td>
</tr>
<tr>
<td>#candidate structures per list</td>
<td>61-132</td>
<td>211-267</td>
</tr>
<tr>
<td>Total number of structures</td>
<td>7730</td>
<td>24128</td>
</tr>
<tr>
<td>#features (original)</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

[10] 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.
Chapter 2

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 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 [14].

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

We investigate three instantiations of this general framework: The Feature Generation approach (Section 2.2) 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 (Section 2.3) 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 (Section 2.4) exploits the cluster assumption on pairs of objects extracted on the test list.

2.1 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, $d$ = list of retrieved documents, and $y$ = list of relevance judgments. Let $S = \{(q_l, d_l, y_l)\}_{l=1..L}$ be the training set consisting of $L$ tuples of query-document-labels. Documents within the set $d_l$ 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 $q_l$).
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 $d_u$ (line 2). Then, we use this additional information, together with the original labeled training data, to obtain a ranking function (line 3). After the ranking function $F_u(\cdot)$ re-sorts the test list $u$, it can be discarded (line 4). The loop (lines 1-5) need not be a sequential operation, but can be computed in parallel since the ranking functions are trained independently.

**Algorithm 1** Local/Transductive Meta-Algorithm

**Input:** Train set $S = \{(q_l, d_l, y_l)\}_{l=1..L}$

**Input:** Test set $E = \{(q_u, d_u)\}_{u=1..U}$

**Output:** Predicted rankings for test: $\{y_u\}_{u=1..U}$

1: for $u = 1$ to $U$ do
2: Observe the test documents $d_u = \{d_u^{(j)}\}_{j=1..J_u}$ 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: $y_u = F_u(d_u)$
5: end for

### 2.2 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. Our proposed Feature Generation 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 2 Feature Generation (FG) Approach to Transductive Ranking

Input: Train set $S = \{(q_l, d_l, y_l)\}_{l=1}^L$

Input: Test set $E = \{(q_u, d_u)\}_{u=1}^U$

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

Input: LEARN(), a supervised ranking algorithm

Output: Predicted rankings for test: $\{y_u\}_{u=1}^U$

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

Algorithm 2 shows the pseudocode for this Feature Generation approach. DISCOVER() is a generic unsupervised method that is applied to each test list $d_u$ separately (line 2). LEARN() is a generic supervised method for learning rank functions. Since the feature-based representations of the training documents $(\{d_l\}_{l=1}^L)$ are enriched with additional test-specific features (line 5), we learn a different ranking function $F_u(\cdot)$ for each test query (line 7). In this paper, we use kernel principal components analysis (Kernel PCA) [11] as the unsupervised method and RankBoost [4] as the supervised ranker. Kernel PCA is advantageous in its flexibility in generating many different types of features by the use of different kernels. This is a good combination with RankBoost, which has been shown to be relatively robust to variations in tuning parameters and feature sets.

1Here, line 2 corresponds to line 2 in the Algorithm 1, lines 3-7 correspond to line 3 in Algorithm 1.
2.3 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. Our Importance Weighting Approach assumes is that each test list exists in a slightly different “domain” from the training data. It requires the two following components:

- An domain adaptation algorithm, \textsc{Adapt()}, that generates importance weights specific to each test list.
- A supervised learning to rank algorithm, \textsc{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 3 Importance Weighting (IW) Approach to Transductive Ranking

\begin{verbatim}
Input: Train set \( S = \{(q_l, d_l, y_l)\}_{l=1..L} \)
Input: Test set \( E = \{(q_u, d_u)\}_{u=1..U} \)
Input: \textsc{Adapt}(), a domain adaptation algorithm
Input: \textsc{Weighted-Learn}(), a supervised ranking algorithm that handles weighted data
Output: Predicted rankings for test: \( \{y_u\}_{u=1..U} \)

1: for \( u = 1 \) to \( U \) do
2: \( W = \textsc{Adapt}(d_u, \{(q_l, d_l, y_l)\}_{l=1..L}) \) \# find weighting over training samples such that samples close to test have high weights
3: \( F_u(\cdot) = \textsc{Weighted-Learn}(W, \{(q_l, d_l, y_l)\}_{l=1..L}) \)
4: \( y_u = F_u(d_u) \) \# predict test ranking
5: end for
\end{verbatim}

Algorithm 3 shows the pseudo-code for the Importance Weighting (IW) approach. In our instantiation, \textsc{Weighted-Learn()} is the AdaCost version of RankBoost [3] and \textsc{Adapt()} is the
Kullback-Liebler Importance Estimation Procedure (KLIEP) [13]. 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 $\text{WEIGHTED-LEARN}()$. Since AdaCost-RankBoost is a pairwise ranking algorithm, our importance weights will be applied to samples consisting of document pairs. If $\text{WEIGHTED-LEARN}()$ 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.

2.4 Investigating the Low Density Separation Assumption

Here we propose a method to exploit the low density separation assumption under the Local/Transductive Framework. In 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” [6, 1, 2]. In classification, for an unlabeled sample $x_i$, the pseudo-margin is defined as $|F(x_i)|$. This is equivalent to assuming that the label is $\text{sign}(F(x_i))$, i.e. which means that we trust the prediction to be correct. For ranking, we would extend pseudo-margin on pairwise samples. Our Pseudo Margin approach (Algorithm 4) first extracts all pairwise samples from an unlabeled list. Then we minimize,

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

which leads to low density separation in the pairwise sample space.

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 4 Pseudo-Margin (PM) Approach to Transductive Ranking

Input: Train set \( S = \{(q_l, d_l, y_l)\}_{l=1..L} \)

Input: Test set \( E = \{(q_u, d_u)\}_{u=1..U} \)

Input: PAIR-EXTRACT(), a procedure to extract pairs of samples

Input: SEMI-LEARN(), a semi-supervised ranking algorithm

Output: Predicted rankings for test: \( \{y_u\}_{u=1..U} \)

1: for \( u = 1 \) to \( U \) do
2: \( P = \text{PAIR-EXTRACT}(d_u) \)
3: \( F_u(\cdot) = \text{SEMI-LEARN}(W, \{(q_l, d_l, y_l)\}_{l=1..L}, P) \)
4: \( y_u = F_u(d_u) \) # predict test ranking
5: end for

2.5 Evaluation in Protein Structure Prediction

We compare the three approaches presented above with a supervised RankBoost baseline. The evaluation metric is GDT-TS [17] and the corresponding z-score. In Tables 2.1 and 2.2 we report results for Top-k items in the ranked list, i.e. Top-1 here corresponds to GDT-TS1, Top-5 corresponds to GDT-TS5 in [10].

In general, we observe little improvement with any of the proposed methods. Importance Weighting is slightly better, but the difference is not statistically significant. The SVR results reported in [10] are GDT=0.589 and z-score=1.17. Scatter plots that compare the result at the individual protein level are shown in Figure 2.1. We observe that Importance Weighting is a relatively more stable method in that the majority (68%) of proteins do not show differences of greater than 0.01 GDT-TS. In contrast, Pseudo Margin is a high-risk high reward approach where 37% of the proteins were improved, 33% were degraded, and 30% remained within 0.01 difference.

The result for Protein Prediction is different from those observed for experiments with Machine
Figure 2.1: Scatter plot of GDT-TS values for the proposed approaches vs. Baseline. Pseudo Margin shows most variability, while Importance Weighting shows the least.
Table 2.1: GDT-TS Results for the Local/Transductive algorithms

<table>
<thead>
<tr>
<th>Top-k GDT-TS</th>
<th>k=1</th>
<th>k=2</th>
<th>k=3</th>
<th>k=4</th>
<th>k=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised Baseline: RankBoost</td>
<td>.579</td>
<td>.590</td>
<td>.595</td>
<td>.599</td>
<td>.604</td>
</tr>
<tr>
<td>Feature Generation (FG)</td>
<td>.569</td>
<td>.586</td>
<td>.596</td>
<td>.601</td>
<td>.605</td>
</tr>
<tr>
<td>Importance Weighting (IW)</td>
<td>.583</td>
<td>.596</td>
<td>.603</td>
<td>.605</td>
<td>.608</td>
</tr>
<tr>
<td>Pseudo Margin</td>
<td>.574</td>
<td>.590</td>
<td>.599</td>
<td>.603</td>
<td>.608</td>
</tr>
</tbody>
</table>

Table 2.2: z-score Results for the Local/Transductive algorithms

<table>
<thead>
<tr>
<th>Top-k z-score</th>
<th>k=1</th>
<th>k=2</th>
<th>k=3</th>
<th>k=4</th>
<th>k=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised Baseline: RankBoost</td>
<td>1.13</td>
<td>1.25</td>
<td>1.30</td>
<td>1.36</td>
<td>1.41</td>
</tr>
<tr>
<td>Feature Generation (FG)</td>
<td>1.07</td>
<td>1.24</td>
<td>1.33</td>
<td>1.40</td>
<td>1.41</td>
</tr>
<tr>
<td>Importance Weighting (IW)</td>
<td>1.15</td>
<td>1.26</td>
<td>1.33</td>
<td>1.35</td>
<td>1.39</td>
</tr>
<tr>
<td>Pseudo Margin</td>
<td>1.03</td>
<td>1.24</td>
<td>1.34</td>
<td>1.40</td>
<td>1.47</td>
</tr>
</tbody>
</table>

Translation and Information Retrieval (not presented here). In Machine Translation, Pseudo Margin gave significant improvements; in Information Retrieval, both Feature Generation and Importance Weighting were very beneficial.
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 particular, we will present a graph-based method the exploits the manifold assumption among lists, where the distances are defined by the list kernel.

3.1 Motivation

Many popular classification 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 give computational speed-ups but also accuracy improvements. Graph-based methods is another class of techniques where kernels 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 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. $K(x,y): R^{d \times N_x} \times R^{d \times N_y} \rightarrow R^+$, 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 \( c \in \mathbb{R}^m \neq 0 \)).

### 3.2 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 have focused on kernels for vectorial data, combinatorial data, and structured data (c.f. [12]). 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 [7] 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 \( p(z) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2} (z - \mu)^T \Sigma^{-1} (z - \mu)\} \) 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 \( \mu \) and covariance \( \Sigma \)).
3. Compute the distance between distributions of two lists \( p(z) \) and \( p'(z) \) using the Bhattacharyya distance:

\[
K(p(z), p'(z)) = \int (\sqrt{p(z)} - \sqrt{p'(z)})\,dz
\]  

(3.1)

While different distances on distributions (e.g. the Kullback-Liebler Divergence) could be used, [7] 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. [16] and Wolf & Shashua [15] are based on geometric properties of the lists using the concept of principal component angles, due to Hotelling [5]. 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 \( U_x = [u_1^x u_2^x u_3^x \ldots] \), where \( <u_i^x, u_j^x> = 0 \) \( \forall \ i \neq j \). The principal angle between the two subspaces is defined as:

\[
\cos(\theta) = \max_{u \in U_x} \max_{v \in U_y} u^T v
\]  

(3.2)

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 information about the shape of the list. If two lists both vary widely in the same dimension but not the others, 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 of the list is most important, since it shows the relationship among objects in the same list. 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.
3.3 Formulation of a List Kernel

In this section we present our kernel, which we simply call the List Kernel. The formulation turns out to be similar to the principal angles kernel since we focus on capturing shape invariance as the notion of distance between lists, though there are significant differences. The main idea is to first use principal component analysis to characterize the subspace spanned by objects within a list, then use a maximum bipartite matching algorithm to find the distance between all basis vectors of this subspace.

The pseudo-code for List Kernel is shown in Algorithm 5. To illustrate the kernel, suppose we have computed the basis vectors for list $x$: $[u^1_x, u^2_x, u^3_x]$, as well as the basis vectors for list $y$: $[u^1_y, u^2_y, u^3_y]$. For ease of explanation, supposed we had only extracted the top three principal component axes. If the top eigenvectors (principal axes) $u^1_x$ and $u^1_y$ point in similar directions, then their dot product is high and the corresponding list kernel value will be high. On the other hand, if $u^1_x$ and $u^1_y$ are dissimilar, but $u^1_x$ and $u^2_y$ are similar, then the list kernel value should be medium-ranged (in effect weighted by $\lambda^1_x \times \lambda^2_y$). 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 5 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 5 Computing the List Kernel

**Input:** List \( x \) and list \( y \).

**Output:** Kernel value \( K(x,y) \).

1: \([\mathbf{U}_x, \mathbf{\Lambda}_x] = \text{PCA}(x)\) (Compute \( M \) principal component axes \( u^m_x \), \( m = 1..M \) and eigenvalues \( \lambda^m_x \), based on vectors in list \( x \).)

2: \([\mathbf{U}_y, \mathbf{\Lambda}_y] = \text{PCA}(y)\) (Similarly compute for list \( y \).)

3: Define a bipartite graph \( G \) with \( M^2 \) edges and \( 2M \) nodes. One side of the graph represent \( u^m_x \) and the other side represent \( u^m_y \).

4: \textbf{for} \( m = 1 \) to \( M \) \textbf{do}

5: \hspace{1em} \textbf{for} \( m' = 1 \) to \( M \) \textbf{do}

6: \hspace{2em} Compute the edge weight, defined as the dot product between principal axes, weighted by the corresponding eigenvalues \( \lambda^m_x \lambda^{m'}_y \cdot | < u^m_x, u^{m'}_y > | \).

7: \hspace{1em} \textbf{end for}

8: \textbf{end for}

9: Compute maximum bipartite matching on graph \( G \). The unnormalized kernel value \( \hat{K}(x,y) \) is defined as the maximum matching value, i.e. \( \hat{K}(x,y) = \sum_{m=1}^{M} \lambda^m_x \lambda^{a(m)}_y \cdot | < u^m_x, u^{a(m)}_y > | \), where \( a(\cdot) \) is a bijection \( a : 1..M \rightarrow 1..M \) that represents the bipartite matching.\(^1\)

10: \textbf{end for}

11: The output kernel value \( K(x,y) \) is normalized by the norm of eigenvalues:

\[
K(x,y) = \frac{\hat{K}(x,y)}{(\|\mathbf{\Lambda}_x\| \cdot \|\mathbf{\Lambda}_y\|)}
\]

This list kernel has advantages over the principal angles kernel because it considers the overall shape of the lists. The principal angles kernel, with its “max-max” operation only considers the correspondence of only one pair of eigenvectors. For example, if \( u^1_x \) matches well with \( u^2_y \), 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.

We use the Hungarian Method (also known as the Kuhn-Munkres algorithm) for bipartite match-
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.

We now show that Algorithm 5 generates a valid kernel.

**Proposition 3.3.1.** The function $K(x, y)$ in Algorithm 5 is symmetric, i.e. $K(x, y) = K(y, x)$.

**Proof.**

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

**Proposition 3.3.2.** The function $K(x, y)$ in Algorithm 5 satisfies the Cauchy-Schwartz Inequality, i.e. $K(x, y)^2 \leq K(x, x)K(y, y)$.

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

$$K(x, x) = \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_x^{a(m)} \cdot | < u_x^m, u_x^{a(m)} > |}{(||\lambda_x|| \cdot ||\lambda_x||)} = \frac{\sum_{m=1}^{M} \lambda_x^m \lambda_x^m \cdot | < u_x^m, u_x^m > |}{(||\lambda_x|| \cdot ||\lambda_x||)} = \frac{||\lambda_x||^2}{(||\lambda_x|| \cdot ||\lambda_x||)} = 1$$
The second step follows from the fact that maximum bipartite matching would achieve \( a(m) = m \forall m \) since \( <u^m_x, u^m_y> = 1 \) and \( <u^m_x, u^{m'}_y> = 0 \) for any \( m \neq m' \). The third step is a result of \( <u^m_x, u^m_y> = 1 \).

Next we show that \( K(x,y)^2 \) is bounded by 1. Note that \( <u^m_x, a^{(m)}_y> \leq 1 \), so that \( K(x,y) = \sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)} \cdot | <u^m_x, a^{(m)}_y> | \) is negative. However, by construction, we will only obtain non-negative eigenvalues \( \lambda_x \) from PCA. Further, the absolute value operation \( | <u^m_x, a^{(m)}_y> | \) ensures non-negativity. Thus, the statement that \( K(x,y) < 0 \) for some \( x,y \) is false.

**Theorem 3.3.3** (Mercer’s Theorem, c.f. [12]). Every positive (semi) definite, symmetric function is a kernel: i.e., there exists a feature mapping \( \phi \) 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 3.3.4.** The function \( K(x,y) \) in Algorithm 5 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 \( \sum_{m=1}^{M} \lambda_x^m \lambda_y^{a(m)} \cdot | <u^m_x, a^{(m)}_y> | \) is negative. However, by construction, we will only obtain non-negative eigenvalues \( \lambda_x \) from PCA. Further, the absolute value operation \( | <u^m_x, a^{(m)}_y> | \) ensures non-negativity. Thus, the statement that \( K(x,y) < 0 \) for some \( x,y \) is false.

### 3.4 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 implementing 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 onto 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 3.1 compares the traditional manifold assumption for classification, and the version we extend to for 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 6.

**Algorithm 6** Ranker Propagation

**Input:** Train set \( S = \{(q_l, d_l, y_l)\}_{l=1..L} \)

**Input:** Test set \( E = \{(q_u, d_u)\}_{u=1..U} \)

**Input:** An optimization method for training linear rankers (in this case we use MERT, a common Machine Translation method \[9\]; alternatively we could use linear SVM)

**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 \( D_{ll} = \sum_j K_{lj} \) is the degree matrix
3. for \( l=1..L \) do
4. Compute list-specific weights: \( w_l = MERT(d_l, y_l) \)
5. Normalize weights: \( w_l = \frac{w_l}{\|w_l\|} \)
6. end for
7. Let \( W_u = -L^{-1}_{uu} \times L_{ul} W_l \), 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 6 minimizes for

\[
\sum_{ij} K_{ij} |w_i - w_j|^2
\]  

(3.3)
in a manner similar to Label Propagation [18], which optimizes

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

(3.4)

In these equations, $K$ is a pairwise similarity measure, $w_i \in \mathbb{R}^t$ is a vector (linear ranker), and $y_i \in \mathbb{R}$ is a scalar (classification label). Note that our objective in Equation 3.3 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.

### 3.4.1 Evaluation in Protein Structure Prediction

We compare three systems:

1. Supervised Baseline - A linear ranker fit over the entire training set, using MERT optimization$^2$

2. A Random Baseline - First we fit a linear ranker for each training list. Then for each test list we randomly draw of these rankers to apply the predictions

3. Ranker Propagation - The described method, which propagates the linear rankers on the training lists to the test lists via List Kernel distances.

The evaluation metric is GDT-TS [17] and the corresponding z-score. In Tables 3.2 and 3.3 we report results for Top-k items in the ranked list, i.e. Top-1 here corresponds to GDT-TS1, Top-5 corresponds to GDT-TS5 in [10].

---

$^2$MERT is an optimization method commonly used in Machine Translation [9]. It assumes a linear ranker and optimizes the parameter iteratively by line search.
We observe that Ranker Propagation outperforms 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). All improvements in this case are statistically significant according to the Wilcoxon signed rank test. We conclude that the manifold assumption is effective for the Protein Structure Prediction task.

Table 3.2: 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).

<table>
<thead>
<tr>
<th>Top-k GDT-TS</th>
<th>k=1</th>
<th>k=2</th>
<th>k=3</th>
<th>k=4</th>
<th>k=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERT (supervised) Baseline</td>
<td>.581</td>
<td>.590</td>
<td>.597</td>
<td>.601</td>
<td>.604</td>
</tr>
<tr>
<td>Random Selection</td>
<td>.521</td>
<td>.549</td>
<td>.567</td>
<td>.582</td>
<td>.588</td>
</tr>
<tr>
<td>Ranker Propagation</td>
<td><strong>.591</strong></td>
<td><strong>.600</strong></td>
<td><strong>.605</strong></td>
<td><strong>.609</strong></td>
<td><strong>.612</strong></td>
</tr>
</tbody>
</table>

Table 3.3: Protein Prediction z-score results

<table>
<thead>
<tr>
<th>Top-k z-score</th>
<th>k=1</th>
<th>k=2</th>
<th>k=3</th>
<th>k=4</th>
<th>k=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERT (supervised) Baseline</td>
<td>1.07</td>
<td>1.17</td>
<td>1.26</td>
<td>1.31</td>
<td>1.34</td>
</tr>
<tr>
<td>Random Selection</td>
<td>0.51</td>
<td>0.81</td>
<td>0.96</td>
<td>1.11</td>
<td>1.16</td>
</tr>
<tr>
<td>Ranker Propagation</td>
<td><strong>1.20</strong></td>
<td><strong>1.31</strong></td>
<td><strong>1.37</strong></td>
<td><strong>1.41</strong></td>
<td><strong>1.44</strong></td>
</tr>
</tbody>
</table>

Regarding other tasks: The Ranker Propagation method also gave significant improvements for Machine Translation, but not for Information Retrieval.
Chapter 4

CONCLUSIONS

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.

- Low Density Separation Assumption: use unlabeled data to discover low density regions, which are to be avoided by the 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 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.

- 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 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.

A concise summary of results for all datasets is presented in Table 4.1.

Table 4.1: Summary of Results. + indicates improvement over baseline, - indicates degradation. = indicates similar results. ++ indicates the best method for a given dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Information Retrieval</th>
<th>Machine Translation</th>
<th>Protein Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Generation</td>
<td>+</td>
<td>-</td>
<td>=</td>
</tr>
<tr>
<td>Importance Weight (Local/Transductive)</td>
<td>++</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Pseudo-Margin</td>
<td>=</td>
<td>++</td>
<td>=</td>
</tr>
<tr>
<td>Ranker Propagation</td>
<td>=</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


