Transfer Learning for Information Retrieval

Pengfei Li

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Master of Computing (Honors), Australia National University

Master of Computing, Australia National University

Bachelor of Software Engineering, Soochow University

School of Science

College of Science, Engineering and Health

RMIT University

May, 2019

Declaration of Authorship

I, Pengfei Li, declare that this thesis titled "Transfer Learning For Information Retrieval", and the work presented in it is my own.

I certify that, except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Signed: Pengfei Li

Date: April, 2019

“Two things are infinite: the universe and human stupidity; and I’m not sure about the universe.”

Albert Einstein

Credits

Proportion of the work in this thesis has been accepted for previous publications:

- Pengfei Li. 2015. Transfer learning for information retrieval. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1061.
- Pengfei Li, Mark Sanderson, Mark Carman, and Falk Scholer. 2016. On the effectiveness of query weighting for adapting rank learners to new unlabelled collections. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, 1413 – 1422

At the time of writing, the work in Chapter 5 is under revision for *Information Sciences journal.*

Acknowledgements

I would like to express my special appreciation and thanks to my advisors, Professor Dr. Mark Sanderson, and Associate Professor, Dr. Falk Scholer: both of you have been tremendous mentors for me. A special thanks to my external advisor, Associate Professor Mark Carman, who has been a fantastic mentor and always inspires me to achieve better. I would like to thank you all for encouraging me to chase my research dream and shape my way of viewing the world. Your advice on both research and my career has been priceless.

Thanks to all my colleagues in the same research office: Ahmed Mourad, Kevin Ong, and Johanne Trippas. You have been great friends and keep me from feeling lonely during this journey.

And finally, a special thanks to my wife, Xianan. Words cannot express how grateful I am to my special other. It has not been easy to tolerate and support me throughout the process, especially through all the days when I was just in my own zone. Without her, I could not make this far.

RMIT UNIVERSITY

Abstract

College of Science, Engineering and Health
School of Science

Doctor of Philosophy

by Pengfei Li

The lack of relevance labels is increasingly challenging and presents a bottleneck in the training of reliable learning-to-rank (L2R) models. Obtaining relevance labels using human judgment is expensive and even impossible in some scenarios. Previous research has studied different approaches to solving the problem including generating relevance labels by crowdsourcing and active learning. Recent studies have started to find ways to reuse knowledge from a related collection to help the ranking in a new collection. However, the effectiveness of a ranking function trained in one collection may be degraded when used in another collection due to the generalization issues of machine learning.

Transfer learning involves a set of algorithms that are used to train or adapt a model for a target collection without sucient training labels by transferring knowledge from a related source collection with abundant labels. Transfer learning can also be applied to L2R to help train ranking functions for a new task by reusing data from a related collection while minimizing the generalization gap.

Some attempts have been made to apply transfer learning techniques on L2R tasks. This thesis investigates different approaches to transfer learning methods for L2R, which are called transfer ranking. However, most of the existing studies on transfer ranking have been focused on the scenario when there are a small but not sucient number of relevance labels. The field of transfer ranking with no target collection labels is still relatively undeveloped. Moreover, the main reason why a transfer ranking solution is needed is that a ranking function trained in the source collection cannot generalize to the target collection, due to the differences in the data distribution of the two collections. However, the effect of the data distribution differences on ranking model generalization has not been examined in detail. The focus of this study is the scenario when there are no relevance labels from the new collection (the target collection), but where a related collection (the target collection) has an abundant amount of training data and labels.

In this thesis, we first demonstrate the generalization gap of different L2R algorithms when the distribution of the source and target collections are different in multiple ways, and we then develop alternative solutions to tackling the problem, which includes instance weighting algorithms and self-labeling methods. Instance weighting algorithms estimate weights for each training query in the source collection according to the target query distribution and use the weighted objective function to optimize a ranking function for the target collection. The results on different test collections suggest that instance weighting methods, including existing approaches, are not reliable. The self-labeling methods use other approaches to generate imputed relevance labels for queries in the target collection, which look to transfer the ranking knowledge to the target collection by transferring the label knowledge. The algorithms were tested on various transferring scenarios and showed significant effectiveness and consistency. We thus demonstrate that the performance of self-labeling methods can be further improved with a minimal number of calibration labels from the target collection. The algorithms and knowledge developed in this thesis can help solve generic ranking knowledge transfer problems under different scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ranking has laid the foundations of many fields, for example, Information Retrieval (IR) and Recommender Systems, as well as Question Answering (QA). For IR applications like search engines, the ranking system looks to return a permutation of documents ordered by their relevance to an information request, expressed in queries, submitted to the system. However, the relevance of a document to an information need is not straightforwardly expressed in the document. Instead, various ranking models, which include the BM25 [1] and language models [2, 3], have been developed to predict the relevance via a set of signals extracted from both the document and the query. However, it has repeatedly been demonstrated that the ranking effectiveness of ranking models varies across different test collections [4–6]. The majority of existing ranking models have been developed based on empirical studies and require parameter tuning for specific corpus. Recent research on Learning to Ranking (L2R) [7] has made significant strides towards training ranking models via machine learning techniques. Note that L2R is not learning to optimise the parameters for existing models, but to train a ranking model that can achieve optimised ranking function for a specific task.

Most L2R algorithms are supervised, which means plenty of training examples are required. Relevance judgments for IR systems are expensive to generate, and quality control can be difficult. In many cases, relevance judgments from a related collection can help ranking function training for the new collection. For example, a hotel booking company based in the US wants to expand its market to Asia and South America. Although they may have an effective ranking function for their existing markets trained with labeled data, it may not generalize to the new market because of language or other differences. However, the relevance judgments or click logs in their existing markets could help them build a new ranking system for the new markets using transfer learning techniques. This thesis investigates the technique used to transfer ranking knowledge from an existing collection (the source collection) to a new collection (the target collection), which is named Transfer Ranking (TR).

1.1 Problem Statement

Like conventional machine learning problems such as regression and classification, the ranking model trained by L2R algorithms can only generalize well when the training and test data are drawn from the same distribution. The so-called "dataset shift" problem [8] arises when the assumption is violated. For example, the effectiveness of a ranking function trained on one document collection will show some decrease when it is applied to a new document collection. Transfer learning [9–11], including its subproblems, domain adaptation [12–14], and multi-task learning [15, 16], has been widely used in the machine learning community for solving dataset shift problems.

Potentially, transfer learning is also a solution to solve the dataset shift issues for L2R collections. However, conventional transfer learning techniques cannot be used for TR directly due to many reasons. One particular reason is that the training data for L2R is generated from a different process as it is from a conventional machine learning dataset. The training data for an L2R algorithm is initialized by retrieving documents from a collection for a set of queries. For efficiency, documents are pooled at a certain depth, however, this makes it harder to formalize the data generating process. As a consequence, the data distribution of L2R collection is governed by a number of factors: the query set, document collection, and pooling depth, as well as the retrieval model used to gather the pool of documents. All the factors have contributed to the challenge of implementing transfer ranking algorithms.

The training data is also used differently by L2R models. According to Liu and others [7], three different types of algorithms have constituted the mainstream of L2R algorithms: *pointwise*, *pairwise*, and *listwise* algorithms. Most of the state-of-the-art L2R algorithms are listwise, so that the objective functions seek to minimize a query-level loss. Furthermore, the evaluation of the effectiveness of a ranking function is based on the averaged query-level measurement metric scores over a set of queries for testing.

Most of the existing transfer learning algorithms have sought to develop new techniques to minimize the differences between the source and target collection. The complicated compromise of the L2R training data has made it a great challenge even to measure the dataset shift among two collections, which makes it harder to implement conventional transfer learning techniques for such collections.

In recent years, some attempts have been made to adapt some of the classical transfer learning algorithms so that they can work for L2R collections. For example, Chen et al. [17] and Gao et al. [18] have sought to develop *instance weighting* algorithms, for TR among L2R collections; weights are assigned to training instances in the source collection to change the data distribution to better resemble the distribution in the target. The authors have demonstrated some success with some small L2R test collections. The study has made important contributions to the study of TR problems, which has inspired many others to try new methods for the task. However, the algorithms have only focused on a particular transferring scenario where the difference between the source and target collection is only the data distribution in the input space. Moreover, the developed algorithms can only work for pairwise algorithms.

Apart from instance weighting, some other methods have also been attempted to address the problem, which include *sample selection* [19], *co-regularization* [20], *feature engineering* [17, 21] and other miscellaneous approaches. Most of the existing solutions can only work for a particular type of L2R algorithm. For example, pairwise L2R algorithms have been widely used for studying TR problems as it is easier to measure the distribution change for the preference data. However, since most of the advanced L2R algorithms are listwise and the evaluation of performance is conducted at the query level, TR for listwise algorithms at the query level is worth examining more closely.

The performance of a ranking system also depends on the resources available during the transferring process, as well as the right choice of the corresponding algorithms. Different transferring scenarios, namely **Unsupervised TR** and **Supervised TR**, are listed and compared in Table 1.1. One commonality between the two transferring scenarios is that there exists a sufficient amount of labeled training examples in the source collection, which includes the document collection, queries submitted to the system, and

Table 1.1: Different transfer ranking scenarios

	D^s	Q^s	R^s	D^t	Q^t	R^t
Supervised TR	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	small amount
Unsupervised TR	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
Minimally Supervised TR	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	minimal amount

the corresponding relevance labels (or click-through logs) for query-document pairs. In some cases, the target collection only contains a document collection, a query set request through the system, and a smaller number of relevance labels available in the target collection. The aim of such a task, in supervised TR, is looking to leverage labeled training instances from a source collection to improve the training of the target ranking function. On the contrary, under the supervised TR scenario, no relevance judgments are assessed for the target collection. The task is more challenging as little information is known for the target collection. Later in the thesis, we also introduce a Minimally Supervised TR scenario, where is only a minimal amount of data required from the target collection to calibrate the transferring process.

Present understanding of TR algorithms is limited. The field of TR is still relatively undeveloped. The problem merits further investigation; for example, how to generalize different L2R models across different test collections; how to measure the relatedness of two L2R collections; and how to quantify the distribution change between L2R collections. Chapelle et al. [22] have also highlighted the need for further investigation of TR.

1.2 Aim and Scope

Most of the previous research on TR has been focused on supervised TR. Unsupervised TR scenarios require further investigation. This study seeks to address the unsupervised TR problem and aims to develop methods to improve the transfer effectiveness of the machine-learned source ranking functions on a target L2R collection.

Therefore the focus of this study is unsupervised TR; the investigation of supervised TR algorithms falls outside the scope of this study. A previous study [23] has attempted to address the unsupervised TR problem when multiple source collections are available. However, we have excluded such settings from our study due to the difficulty of accessing a valid testing environment and the fact that the setup cannot generalize to all real scenarios. Similarly, multi-task learning to rank is beyond the scope of this study.

In practice, there might exist some other information in the target collection which could help a transfer, for example: document information, collection information, and click logs. However, most of the public L2R test collections have purposely excluded such information from the collection. This study will focus on the unsupervised TR scenario with minimal information requirement, which is when the extracted feature vectors for query-document pairs are accessible.

One condition imposed by the study is that the source and target data should have the same feature space. Heterogeneous transfer learning, which looks to implement transfer learning when the feature spaces are different, will not be considered in this study.

To be able to develop better algorithms for unsupervised TR, several research questions are to be addressed.

Research question 1: ***What are the generalization abilities of various L2R algorithms across different L2R collections?***

Theoretical analysis of the generalization abilities of different L2R algorithms has been demonstrated by some early studies [24, 25]. However, the generalization ability of an L2R algorithm on a different collection has not been well studied. As mentioned in section 1.1, L2R collections can be distinct from one another in various ways, the impact of the differences on the cross-collection is not easy to determine and needs further investigation.

Furthermore, understanding of the cross-collection generalization will help understand how TR can contribute to improving ranking effectiveness on a target collection. An examination of generalization will lay the foundations for TR.

Research question 2: ***How can we implement unified unsupervised TR algorithms on L2R collections to maximize the transfer effectiveness on a target collection?***

The query-level nature of L2R algorithms poses some problems when implementing TR algorithms. The task of unsupervised TR is complicated further by unavailable relevance information. In this study, we look to develop new TR algorithms that maximize the ranking effectiveness on the target collection in an unsupervised environment.

Instead of developing algorithms that work for some specific L2R algorithms, the study seeks to develop unsupervised TR algorithms that fit most L2R algorithms.

Research question 3: ***How can we guarantee the performance of unsupervised TR techniques so that they can work in various environments?***

Finally, without any relevance judgment information from the target collection, it is hard to determine whether a successful transfer has been made. Further investigation is necessary to examine whether one could evaluate the performance of the TR algorithm with a minimal relevance judgment requirement.

1.3 Significance of the Study

One intended outcome of the study is to provide better insight into the generalization of the L2R algorithms. The further analysis of the data generating process - namely the probabilistic model controlling the data distribution - of L2R collection will broaden our understanding of L2R algorithms and can direct attention towards better strategies to establish L2R test collections. Moreover, the study will result in a better understanding of how L2R algorithms are training. The thorough examination of various L2R algorithms across different test collections may reveal better practices for training L2R models.

Transfer learning has attracted much attention in machine learning communities. Most of the existing solutions for transfer are focused on classification and regression problems, where the distribution change happens in either the input feature space or the conditional distribution. This study will expand the current knowledge of transfer learning for ranking data and learning for matching problems [26].

The robustness of ranking functions across various collections has always been a concern for the IR community. The study of TR problems will provide new insight into how machine learning trained ranking function responds to a particular change in the collection.

The study can contribute many practical applications in the industry. For example, the knowledge we gain from the study of unsupervised TR is related to the following applications:

- **Domain adaptation**: TR knowledge from an existing ranking system to a new domain, where no relevance judgments are available, for example, when an enterprise search service provider wants to adapt a ranking function trained for a medical company to a newspaper corporation.
- **Cross-language transfer**: Deploy ranking system to a market where the language of the collection is different from the original document collection, for example, when a search company looks to deploy their ranking system trained in the US market to other countries.
- **Document collection shift**: Update ranking systems when the original document collection has substantially changed over time, for example, when a newspaper press intends to update their news search engines after an extended period of time.
- **Ranking adaptation to unseen collection**: Adapt ranking function to a new collection, where the content of documents is not available due to privacy considerations. For example, when an email service company want to train a ranking function for their customers email search system, where only some of their staffs email is disclosed for making relevance judgments.

1.4 Overview of the Study

The thesis consists of seven further chapters to address the posed research questions. In Chapter 2, the background theory of learning to rank and TR is discussed. To give a clear image of the field of TR, related works on both transfer learning and TR are also presented.

The main content of the study addresses the first research question through empirical study of various L2R algorithms across different test collections in Chapter 3, which includes a thorough examination of L2R algorithms by training and testing in various test collections, and an experiment to analyze what makes a good ranking model.

One of the most common solutions for transfer learning is instance weighting, which looks to change the data distribution in a source collection to be closer to the data distribution in a target collection by assigning instance weighting to training samples in the target collection. Instance weighting has also been used for TR. However, existing methods lack consideration of query effects on rankings. In Chapter 4, we develop query-level instance weighting algorithms. The algorithms are tested and compared with other instance weighting methods across various collections and different transferring settings. Instance weighting could be a great challenge for TR due to the difficulty of density ratio estimation for queries. Alternatively, one could use label imputation-based methods to generate pseudo relevance labels for training examples in the target collection. This approach is discussed in detail in Chapter 5. A challenge for all unsupervised TR algorithms is that the performance may vary in different transferring settings, yet it is hard to measure the performance without relevance information. In Chapter 6, we have relaxed the condition (TR without any relevance labels) by allowing a few relevance judgments. Various techniques are studied to help train the system and improve the reliability of the algorithms.

Finally, conclusions of findings, the limitations and the future works of the studies are discussed in Chapter 7.

Chapter 2

Background and Related Work

This chapter introduces the field of transfer learning for IR. It begins by introducing conventional ranking models for IR, followed by some basic background on *learning-to-rank*. The concept of transfer learning, as well as TR will be introduced in the next sections. The last section of this chapter will discuss some related work.

2.1 A Brief History of Ranking Models for IR

The task for an IR system is to find relevant documents from a corpus of documents in order to satisfy a particular information need, which is usually expressed in the format of a user-defined search query. An effective IR system should discover as many relevant documents from the corpus as possible, and rank them in decreasing order of relevance to the information need.

Before learning-to-rank was introduced, many retrieval and ranking models were developed. Most of these models aim to compute similarity scores between the documents and the query. The similarity scores are then used to determine which documents are included in the result list and the ranking orders. In the following subsections, some of the classical retrieval models will be reviewed briefly.

2.1.1 Vector Space Model

The vector space model (VSM) [27] is one of the earliest ranking models. Under the VSM model, both documents and queries are represented by vector representations in word space. The similarity between the query and document is then computed using cosine similarity. The element in the vector could be a statistic of the words, which may be word counts or term frequencyinverse document frequency (tf-idf). The *tf-idf* score is a combination of two term statistics: term frequency and inverse document frequency, which aims to measure the importance of terms in the document and the whole corpus. There are different variants of term frequency and inverse document frequency. One of the simplest implementations of tf-idf is given as follows.

Term frequency (*tf*) is the frequency of a term t appearing in the document d, which can be computed as:

$$tf(t, d) = c(t, d) \tag{2.1}$$

where $c(t, d)$ denotes the count of term t in document d. There are also different variants of term frequency, which include document length normalized term frequency, logarithmically scaled term frequency and others [28, 29]. For example,logarithmically scaled term frequency is calculated as:

$$tf(t, d) = log(1 + c(t, d)) \tag{2.2}$$

The document frequency (*df*), on the other hand, measures the document frequency of a term in the collection. It measures how common a term is among all the documents in the collection. If a term in a document rarely appears in the collection, it provides an important signal for distinguishing the document from others. The inverse document frequency is commonly calculated as:

$$idf(t, C) = \log \frac{|C|}{|\{d \in C|t \in d\}|} \tag{2.3}$$

where C denotes a set of documents (the collection), and $|C|$ is the size. Similar to term frequency, there exist other variants of inverse document frequency [28, 29].

The VSM model calculates the cosine similarity of the document and the query using the inner product of the two vectors:

$$sim(d,q) = \frac{\vec{V}(d) \cdot \vec{V}(q)}{\left\|\vec{V}(d)\right\| \left\|\vec{V}(q)\right\|} \tag{2.4}$$

Here $\vec{V}(d)$ and $\vec{V}(q)$ are the vectors for the document and query respectively. $|\vec{V}(d)| = \sqrt{\sum_i \vec{V}(d)}$ is their Euclidean length. Given the similarity scores for every query-document pair, documents are then ranked in the descending order.

2.1.2 BM25

BM25 [1] is a probabilistic ranking model that has been widely used for ranking documents. Before BM25 was developed, several "BM" algorithms were proposed to heuristically approximate the 2-poisson probabilistic model of Robertson and Walker [30]. One of the most famous variants of BM25 is the ATIRE BM25 [6], which can be computed as:

$$BM25(d,q,C) = \sum_{t \in q}^{M} \frac{idf(t,C) \cdot c(t,d)(k_1+1)}{tf(t,d) + k_1(1 - b + b \cdot \frac{c(d)}{avgc(d)})} \tag{2.5}$$

where $c(d)$ is the document length (number of words in d), $avg\ c(d)$ is the average document length in the collection, and k_1 and b are two user-specified parameters. Different variants of BM25 have been developed; a complete comparison of different variants can be found in Trotman et al. [6]. BM25 and its variants have shown to be effective on many TREC collections, however, its performance may vary amaong different collections.

2.1.3 Language Model

Language models [2] are widely used probabilistic retrieval approaches for document retrieval. Statistical modelings of text documents have been widely used for solving natural language processing problems before they were introduced to IR by Ponte and Croft [2]. Language models for IR (LMIR) assume that queries are formulated by choosing terms from a relevant document. As a result, the relevance between the document and query can be approximated by estimating the "query likelihood" for a document, which is the likelihood of generating the query given the document. To be able to compute the query likelihood, the algorithm first establishes a probabilistic language model M_d that fits for each document d in the collection.

The language model for each document is a categorical distribution over the set of terms, which can be approximated as follows:

$$\hat{p}(t|M_d) = \frac{c(t,d)}{c(d)} \tag{2.6}$$

where $c(d)$ is the total number of terms in document d.

The query likelihood is calculated as $p(q|M_d)$. The probability that a query q been generated from the language model M_d [2] is then estimated by:

$$\hat{p}(q|M_d) = \prod_{t \in q} \hat{p}(t|M_d) \times \prod_{t \notin q} (1 - \hat{p}(t|M_d)) \tag{2.7}$$

Some terms appearing in the query may not be seen in the document, which means that $\hat{p}(t|M_d)$ could be zero. As a consequence, the query likelihood will be zero if some of the query terms do not exist in the document, which is problematic. To tackle this "smoothing" issue, when a term is absent from the document, Ponte and Croft [2] proposed to smooth the probability with the term's global probability in the collection:

$$\hat{p}(t|M_d) \approx \frac{c(t,C)}{cl} \tag{2.8}$$

where $c(t,C)$ denotes the counts of term t in collection C. $cl = \sum_{t'} c(t',C)$ is the total number of tokens in the collection. Several variants of language models have been investigated to improve the stability.

A complete review of the language models was discussed in Zhai [31]. As it has shown above, smoothing the maximum likelihood of $p(t|M_d)$ can affect the accuracy of the language model. Many solutions have been proposed to improve the accuracy of the estimation, for example, the Jelinek-Mercer method [32] is a mixture model using the linear interpolation of the maximum likelihood model of the document with the model with the collection collection:

$$p(t|M_d) = (1 - \lambda)p(t|M_d) + \lambda p(t|M_C) \tag{2.9}$$

where M_C is a language model fit for the whole collection, $p(t|M_C)$ is the likelihood of the term t in the collection language model, λ is a parameter controlling the importance of the document and collection language model. Like BM25 models, most language models have tuning parameters. In Zhai and Lafferty [32], the authors have shown that the performance of language models on IR collections are sensitive to the parameters.

2.1.4 Learning-to-Rank

Many retrieval and ranking models have been proposed to achieve better ranking effectiveness. However, the effectiveness of different ranking models can vary across different test collections, and many require parameter tuning. Data fusion methods [33, 34] have been investigated for combining different retrieval models in order to take advantage of multiple models. However, the data fusion approaches also require settings for proper combinations of models.

Learning-to-rank is closely related to data fusion. Learning-to-rank algorithms use machine learning techniques to combine various query-document-related features. The features could also be the scores produced by some retrieval models. However, as it has been pointed out by Macdonald et al. [35], the scenarios where data fusion and learning-to-rank can be used are different. More specifically, data fusion combines different retrieval results from different systems, which means the retrieved documents could be different.

The advent of learning-to-rank has brought significant improvements to the ranking effectiveness of modern IR systems. It uses machine learning techniques to learn discriminative models, which combine various query-document-related features, to build more complex and effective ranking functions. Previous studies [36] have demonstrated the power of learning-to-rank algorithms. The next section will provide a more detailed explanation of learning-to-rank algorithms.

2.2 Learning-to-Rank

Learning-to-rank (L2R) is a field of research that uses machine learning techniques to solve ranking problems. For IR systems, the results represented to users are obtained via a two-stage ranking process illustrated in Figure 2.1. A user submits a search query

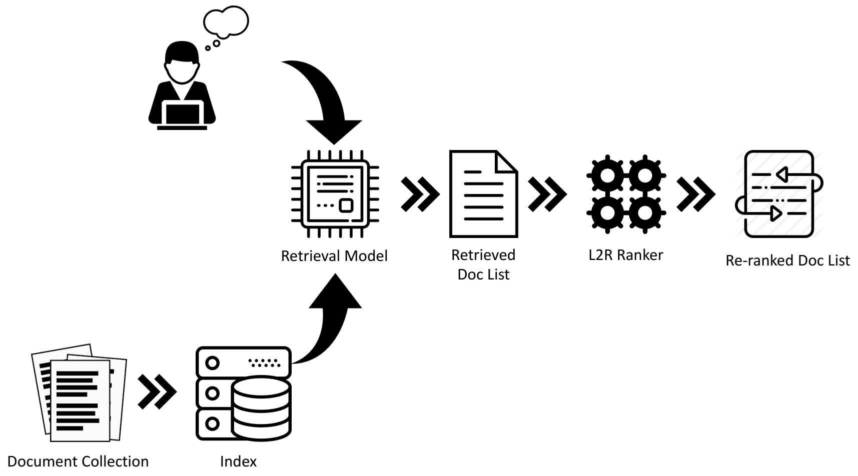


FIGURE 2.1: Two-stage ranking system

expressing his/her information need to the retrieval system. For the consideration of efficiency, a conventional retrieval model is used to retrieve potentially relevant documents from the collection. In the second stage, an L2R trained ranker is utilized to re-rank the set of documents [37] that have been retrieved during the first stage. One of the reason why L2R is only applied at the second stage is that some of the features used in L2R models may be too expensive to calculate, which can harm users' search experiences.

2.2.1 Training Data for L2R

Like many other machine learning algorithms, L2R algorithms learn patterns from existing examples. The training examples for learning-to-rank algorithms are a set of ranked lists of documents corresponding to queries from a query set. As mentioned in the previous section, documents in the ranked lists are first selected using a classical retrieval model. For efficiency consideration, a subset (usually the top k items) of the returned documents list is used for training. The documents retrieved for a query are represented by feature vectors. The features for L2R models could be: i) **document features**, for example, signals measuring the quality of the document like PageRank; ii) **query features**, which indicate the statistics of the queries, for example, the length of the query; and iii) **matching features**, which reflect the relation of the query and document. For example, matching features could be conventional retrieval models like BM25. The ranking order of retrieved documents for each query is inferred from assessor-annotated relevance labels or click-through data [38].

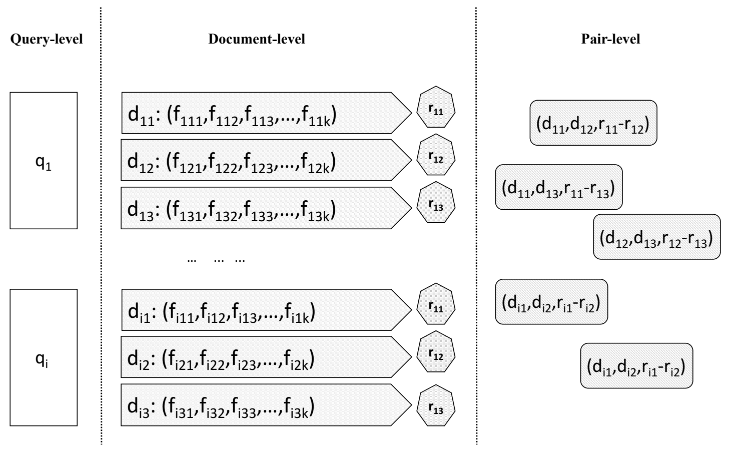

Query-level
Document-level
Pair-level
q1
qi
d11: (f111,f112,f113,...,f11k)
d12: (f121,f122,f123,...,f12k)
d13: (f131,f132,f133,...,f13k)
di1: (fi11,fi12,fi13,...,fi1k)
di2: (fi21,fi22,fi23,...,fi2k)
di3: (fi31,fi32,fi33,...,fi3k)
r11
r12
r13
(d11,d12,r11-r12)
(d11,d13,r11-r13)
(d12,d13,r12-r13)
(di1,di2,ri1-ri2)
(di1,di2,ri1-ri2)

FIGURE 2.2: Data formatting for L2R

The relevance labels for query-document pairs can be binary (relevant/irrelevant). For more complicated systems like web search, graded relevance is usually employed. For example, the relevance label of query-document relevance could be labeled at four scales: irrelevant (0), marginally relevant (1), fairly relevant (2), and highly relevant (3) [39]. An example of L2R training data is demonstrated in Figure 2.2. According to the usage of data during the training, the training data for L2R forms three levels: query level, document level, and document pair level, which determines the differences between different L2R algorithms.

2.2.2 Formal Definition of Learning-to-Rank

Following the notation in Cao et al. [40], let $\mathcal{Q} = \{q_1, q_2, \cdots, q_m\}$ be a set of queries; d_i=$(d_{i1}, d_{i2}, \cdots, d_{in})$ be the list of documents associated with query q_i from document space $\mathcal{D}$, where d_{ij} is the j^{th} document for query q_i. Furthermore, let $\mathbf{x}_{ij} = \Phi(q_i, d_{ij})$ be the feature vector generated from the query document pair. For simplicity, we will refer to **query document pairs** as **documents** throughout the remaining sections. To avoid ambiguity, we use $\vec{\mathbf{x}}_i$ to denote the set of document feature vectors corresponding to the query $\vec{\mathbf{x}}_i = \{\mathbf{x}_{ij}\}_{j=1}^{n}$ and let $\vec{r_i} = \{r_{ij}\}_{j=1}^{n}$ be the list of relevance scores, where r_{ij} denotes the score of the j^{th} document for q_i. Finally, we use π_i to denote a permutation of the documents for the i^{th} query.

A training example $t_{ij} = (\mathbf{x}_{ij}, r_{ij})$ consists of a feature vector $\mathbf{x}_{ij}$ and a relevance judgement r_{ij}. For ease of expression, we simplify the notation, denoting the set of training examples for each query, i.e., the ranked list, as: $l_i = (\vec{\mathbf{x}}_i, \vec{r}_i)$. A training dataset consisting of multiple queries with associated relevance judgments is then denoted by $\mathcal{L}$.

Given a training sample $\mathcal{L}$, consisting of a set of ranked lists, drawn from the query set $\mathcal{Q}$ and the document collection $\mathcal{D}$, the objective of learning-to-rank algorithms is to train a ranking function that can best predict the ranking order of retrieve documents for a query, given the feature vectors for each document:

$$f(\{\mathbf{x}_{i1}, \mathbf{x}_{i2}, \cdots, \mathbf{x}_{in}\}) = \{\pi(1), \pi(2), \cdots, \pi(n)\} \tag{2.10}$$

Since it is difficult to train a function that directly map a set of feature vectors to a ranking order, L2R algorithms usually train ranking functions that predict relevance labels or real-valued relevance scores (indicating the degree of relevance) for individual documents. Documents are then ranked in decreasing order of the labels/scores.

Some other L2R algorithms train functions to predict the relative orders (pairwise preferences) of pairs of retrieved documents for the same query. A ranked list can then be induced from the pairwise ranking preferences.

2.2.3 Evaluation for IR Models

Effective retrieval models and ranking models aim to return a ranked list of documents to users that can meet their information needs. The effectiveness of a retrieval system is usually measured using an evaluation metric that accounts for users' perceptions of ranking quality on queries, and is averaged across all the queries submitted to the system. As a result, the objective of L2R models is to train ranking functions that can maximize these metrics. Before going into the details of how L2R algorithms are implemented, some IR evaluation metrics will be introduced in the following section to reflect the difference between L2R and other machine learning tasks.

The two most straightforward metrics to measure the effectiveness of IR models are precision and recall. Precision for a query is defined as the fraction of relevant documents among all the documents retrieved for an information need:

$$precision = \frac{\#retrieved\ relevant\ docs}{\#retrieved\ docs} \tag{2.11}$$

Meanwhile, recall is the percentage of retrieved relevant documents among all relevant documents in the collection:

$$recall = \frac{\#retrieved\ relevant\ docs}{\#relevant\ docs} \tag{2.12}$$

However, past research [41] has shown that users are less likely to examine search results after the first or second pages. As a result, a criteria for a good IR system is that it ranks more relevant documents higher. Neither of the above metrics consider the rank positions of relevant documents in the returned list. If relevant documents are ranked on the bottom of the list, they will have the same precision and recall score with the list that put all relevant documents on top of the list. Precision and recall can only reflect how good the retrieval system is, but not its ranking effectiveness.

Average precision (AP) is a metric that measures the average of precision values at each recall level (when running down the ranked list). As a result, the metric favors ranking functions that put relevant documents at the top of the list. These measurements can be completed up to some maximum rank, e.g., P@20.

The average precision of a query is the average of the precision values of the ranked list at the rank position of each relevant doc:

$$Average\ Precision = \frac{1}{|D_r|} \sum\nolimits_{d_k \in D_r} P@k(l_i) \tag{2.13}$$

where D_r is the set of retrieved relevant documents, $|D_r|$ is the size of the retrieved documents, and k is the ranking position of the document.

The mean average precision (MAP) measures the mean AP over all queries in the evaluation set:

$$MAP = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n} \sum_{k=1}^{n} P@k(l_i) \tag{2.14}$$

where m is the total number of queries in the evaluation set.

One drawback of MAP is that it only considers binary relevance labels, meaning that documents are either considered relevant or irrelevant. However, as mentioned before, the degree of relevance of documents to queries can be different. As a result, graded relevance is sometimes applied for relevance judgments. Normalized Discounted Cumulative Gain (NDCG) [39], on the other hand, measures the ranking effectiveness with graded relevance. The cumulative gain (CG) aggregates gains in the number of relevant documents observed when iterating through the ranked list. For an ideal ranking, highly relevant documents should be ranked higher on the list, thus a rank-based discount function is introduced to the cumulative gain so that the metric places more emphasis on top-ranked documents:

$$DCG = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{2.15}$$

Here rel_i denotes the relevance judgement for the i^{th} document in the list and $2^{rel_i} - 1$ is an exponent gain formula used in Burges et al. [42]. The denominator $\frac{1}{\log_2(i+1)}$ is the discount function. There exist other gain and discount functions for DCG, with a comparison of different methods discussed in Kanoulas and Aslam [43]. In effect, a highly relevant document ranked higher in the list obtains more gain than a highly relevant document that ranked lower in the list. Since the length of the list as well as total members of relevant and irrelevant documents can vary across queries, a normalized DCG, NDCG, was proposed to normalize the metric with respect to the ideal ranking of the documents retrieved for each query:

$$NDCG = \frac{DCG}{IDCG} \tag{2.16}$$

where IDCG is the ideal DCG score for the returned documents. Similarly, the effectiveness of a ranking system is measured by averaging across queries. Sometimes a cut-off k is applied for the metrics to reflect the users' preferences on top-ranked documents - for example, NDCG@10 measures the NDCG score at rank 10.

2.2.4 Learning-to-Rank Approaches

Different learning-to-rank algorithms have been developed to maximize the ranking effectiveness of the trained ranking function on a target collection. As was noted in the previous section, the training data for learning-to-rank can be viewed on three different levels, and the objective can be achieved by minimizing losses at different levels. As a result, learning-to-rank algorithms can be classified as *pointwise*, *pairwise* and *listwise* algorithms. In this section, we will explain the differences between various learning-to-rank algorithms and how they are implemented.

Pointwise learning-to-rank algorithms train ranker functions at the document level. The algorithms are designed to minimize the expected loss over all the query document pairs used for training. Pointwise algorithms attempt to predict the relevance class of each document separately, and thus cast the ranking problem as a classification, regression or ordinal regression problem [7].

$$\vec{\theta}^{*} = \arg\underset{\vec{\theta}}{min}\ \mathbb{E}_{(\mathbf{x},r)\in\mathbf{T}}[\ell(f(\mathbf{x};\vec{\theta})), r] \tag{2.17}$$

where $\mathbf{x}$ is the feature vector for a query-document pair, r is the corresponding relevance label, $\mathbf{T}$ is the training set, f is the ranking function, $\vec{\theta}$ is the parameter of the ranking function to be learned.

All ranking functions learned with pointwise algorithms take a query-feature vector as input; the differences between the algorithms exist in the output space. The classification-based approaches predict a relevance label for the feature vector:

$$f(\mathbf{x};\vec{\theta}) \mapsto \hat{r} \tag{2.18}$$

where θ are the parameters for function f.

As a result, the loss function for classification-based approaches measures the gap between the ground-truth labels and their predicted labels, $\ell(f(x;\vec{\theta}), r)$. The training is then aiming to minimize the average loss over the observed training set:

$$\vec{\theta}^{*} \approx \arg\underset{\vec{\theta}}{min}\ \frac{1}{|\mathbf{T}|}\sum_{(\mathbf{x},r)\in\mathbf{T}}[\ell(f(\mathbf{x};\vec{\theta}), r)] \tag{2.19}$$

where $\mathbf{T}$ denotes the number of training query document pairs in the collection. As has been mentioned previously, the relevance labels for the query document pairs could be either binary or multi-graded. As a result, some algorithms have cast the problem as a classification task and used binary classifiers like Support Vector Machines (SVM) [44] to predict the relevance labels.

The ranking function trained with a SVM model is in the linear scoring form: $f(\mathbf{x};\vec{\theta}) = \vec{\theta}^{\top}\phi(\mathbf{x})$+b, where $\phi(.)$ is a kernel function that maps the feature vector to a kernel space, b is a constant. The relevance labels are mapped to binary labels: $y = 1$ when $r = 1$, $y = -1$ when $r = 0$. The objective function is as following:

$$\begin{aligned}
\vec{\theta}^{*} = &\arg\min_{\vec{\theta}} && \frac{1}{2}||\vec{\theta}||^2 + \lambda \sum_{i=1}^{|\mathbf{T}|} \xi_i \\
&\text{subject to} && \vec{\theta}^{\top}\phi(\mathbf{x}_i) \leqslant -1 + \xi_i, \text{if } r_i = 0, \forall i \\
& && \vec{\theta}^{\top}\phi(\mathbf{x}_i) \geqslant 1 - \xi_i, \text{if } r_i = 1. \forall i \\
& && \xi_i \geqslant 0.
\end{aligned} \tag{2.20}$$

where ξ denotes the hinge loss:

$$\xi = max(0, 1 - y_i \cdot f(\mathbf{x}; \vec{\theta})) \tag{2.21}$$

Other methods have also been investigated to deal with graded relevance, for example, Li et al. [45] have used boosted classification trees to minimize a surrogate loss [1] for the multi-class classification, which is then used to predict the relevance labels. Apart from the classification-based solutions, some research has looked to cast the problem as regression or ordinal regression problems [46–48] .

Pointwise algorithms minimize the differences between the predicted relevance labels of documents and the ground-truth labels. However, since the absolute relevance labels are less significant than the relative ordering of pairs of documents, the solution can sometimes be problematic due to the complexity of relevance judgements.

Pairwise algorithms are a series of algorithms that use document pairs as training data. Different from pointwise algorithms, the inputs for pairwise L2R algorithms are pairs of documents retrieved for the same query, $(\mathbf{x}_{ij}, \mathbf{x}_{ik})$, which are from the same ranked list. The output of pairwise L2R algorithms is a ranking function predicting whether one document is more relevant than another ($\Delta r_{ijk} = 1, \ \forall \ r_{ij} > r_{ik}$, and $\Delta r_{ijk} = -1, \ \forall \ r_{ij} < r_{ik}$). The ranking function is thus trained by minimizing the expected pairwise loss:

$$\vec{\theta}^{*} = arg \min_{\vec{\theta}} \mathbb{E}_{(\mathbf{x}^2, \Delta r) \in T}[\ell(f(\mathbf{x}^2; \vec{\theta}), \Delta r)] \tag{2.22}$$

[1]a surrogate loss replaces and approximates a designed loss function

where $\mathbf{x}^2 \in \{(\mathbf{x}_{ij}, \mathbf{x}_{ik})_{j \neq k}\}$. In practice, as it is difficult to compute the distribution of the document pairs, minimizing the mean pairwise loss is used:

$$\vec{\theta}^* = \underset{\vec{\theta}}{arg\ min}\ \frac{1}{N_{\mathbf{x}^2}} \sum_{(\mathbf{x}_{ij}, \mathbf{x}_{ik}, \Delta r) \in T} \ell(f(\mathbf{x}_{ij}, \mathbf{x}_{ik}; \vec{\theta}), \Delta r) \tag{2.23}$$

The loss function for the pairwise models looks to minimize the gap between the predicted pairwise preferences and the observed ranking preferences. RankingSVM[38] is an example of a pairwise L2R algorithm that uses the SVM learning algorithm to build a classifier that can predict the ranking preferences of a pair of documents. The input for the algorithm is the difference between the feature vectors of the document pairs, while the labels (Δr) are the difference between the corresponding relevance labels.

Listwise L2R are a set of algorithms that looks to directly optimize the ranking function at the query level. The objective of the listwise L2R algorithms is to minimize the expected loss over the ranking for each query:

$$\vec{\theta}^* = \arg \underset{\vec{\theta}}{min}\ \mathbb{E}_{(\mathbf{x}, \vec{r}) \in \mathcal{L}}[\ell(f(\mathbf{x}; \vec{\theta}), \vec{r})] \tag{2.24}$$

Similarly, the expected loss is estimated by the mean loss over the queries in the training set:

$$\vec{\theta}^* = \underset{\vec{\theta}}{arg\ min}\ \frac{1}{N_q} \sum_{i=1}^{N_q} [\ell(f(\mathbf{x}_i; \vec{\theta}), \vec{r}_i)] \tag{2.25}$$

The loss function for the listwise algorithm quantifies the difference between the ranking orders of the documents in the query and the ground-truth rankings. Similar to the pointwise algorithm, the ranking function trained by the listwise algorithm takes a document feature vector as the input and outputs a relevance score. The documents are then ranked according to the relevance scores. The ground-truth rankings of the documents are inferred from the ground-truth labels of the query document pairs.

The loss function for listwise algorithms that compares the ground-truth ranking with the predicted ranking is not always straightforward. Some algorithms like ListNet [40] have attempted to use the cross-entropy loss of the permutation probabilities between the ground-truth ranking and the predicted ranking:

$$\ell(f(\mathbf{x}_i;\vec{\theta}),\vec{r}_i) = -p(\pi_i|\vec{r}_i)\log p(\hat{\pi}_i|f(\mathbf{x}_i;\vec{\theta})) \tag{2.26}$$

where $p(\pi_i|\vec{r}_i)$ is the probability of observing a permutation π_i, provided with the relevance labels $\vec{r}_i$. The permutation probability is estimated with the Plackett-Luce model [49, 50]; $p(\hat{\pi}_i|f(\mathbf{x}_i;\vec{\theta}))$ is the probability of the permutation $\hat{\pi}_i$ given the documents $\mathbf{x}_i$ and the ranking function f controlled by the parameters $\vec{\theta}$.

As discussed in the previous section, many metrics have been developed to measure the ranking effectiveness. Some listwise algorithms attempt to directly optimize the algorithm to maximize the metrics. These ranking effectiveness ranking metrics are, however, not smooth with respect to the relevance scores. Thus it is difficult for learning algorithms to optimize them directly. Different solutions have been proposed to achieve this goal by instead optimizing a surrogate objective function.

AdaRank[51] uses a boosting ensemble method called AdaBoost[52] to iteratively optimize the ranking function to achieve better ranking metric scores. Intuitively, AdaRank learns an ensemble of weak rankers that can achieve better ranking effectiveness:

$$f(\mathbf{x}) = \sum_{t=1}^{T}\alpha_t h_t(\mathbf{x}) \tag{2.27}$$

where $h_t(.)$ is the weak ranker trained at the t^{th} iteration of the algorithm and α_t are the weights for $h_t(.)$. In each iteration of AdaRank, the algorithm learns a new weak ranker to maximize the effectiveness metrics such as NDCG and then adds the weak ranker to the ensemble. The new added weak ranker will come with a weight which is computed based on its performance on the training set. Moreover, according to the performance of the current ensemble, the queries are assigned with weights to reflect their importance in the next iteration. The purpose of doing so is to make sure that the algorithm will focus on the poorly performed queries under current ensemble. Finally, an ensemble of weak rankers is trained to gain better effectiveness on the entire collection.

One of the state-of-the-art listwise algorithms is called LambdaMART [53], which trains boosted regression trees to maximize the ranking effectiveness. Note that some researchers classify LamdaMART as pairwise algorithms as the loss function was computed based on pairs of documents. However, the loss function of LambdaMART also involves the effectiveness metric of the ranking, thus the training of the algorithm will also be affected by the query distribution. As a result, LambdaMART is treated as listwise algorithm throughout the thesis. The output of the LambdaMART algorithms is an ensemble of regression trees, which is in the same format as Equation 2.27, but each hypothesis model $h_t(\mathbf{x})$ is a regression tree. LambdaMART was developed from a pairwise algorithm called RankNet [42]. RankNet models the ranking preference probabilities of document pairs, $\bar{P}(r_{ij} > r_{ik})$, which are inferred from the relevance labels. The ground-truth probability is modeled as:

$$\bar{P}(r_{ij} > r_{ik}) = \frac{1}{2}(1 + S_{ij}) \tag{2.28}$$

where $S_{ij} = 1$ if $r_{ij} > r_{ik}$, $S_{ij} = -1$ if $r_{ij} < r_{ik}$, and $S_{ij} = 0$ if $r_{ij} = r_{ik}$. The preference probability for a pair of documents is modeled using the sigmoid function applied to the difference between the relevance scores predicted by the model:

$$P(r_{ij} > r_{ik}) = \frac{1}{1 + e^{-\sigma(s_{ij} - s_{ik})}} \tag{2.29}$$

where σ controls the shape slope of the sigmoid function. To simplify, let $\bar{P}(r_{ij} > r_{ik})$ and $P(r_{ij} > r_{ik})$ be denoted as $\bar{P}_{ijk}$ and P_{ijk} respectively. RankNet uses the cross entropy loss to estimate the cost between $\bar{P}_{ijk}$ and P_{ijk}:

$$C_{ijk} = -\bar{P}_{ijk} \log P_{ijk} - (1 - \bar{P}_{ijk}) \log(1 - P_{ijk}) \tag{2.30}$$

The cost function of the model on the training set is then calculated as:

$$C = \sum_{q_i \in Q} \sum_{j,k} I(r_{ij} > r_{ik}) \log\left(1 + e^{-\sigma(s_{ij} - s_{ik})}\right) + I(r_{ij} < r_{ik}) \log\left(1 + e^{-\sigma(s_{ik} - s_{ij})}\right) \tag{2.31}$$

In order to optimize the model, the gradients of the cost with respect to the document scores, $\lambda_{ijk} = \frac{\partial C_{ijk}}{\partial s_{ij}}$ and $\lambda_{ikj} = \frac{\partial C_{ijk}}{\partial s_{ik}}$, will be calculated and the corresponding model weights will be updated accordingly.

A continuing study [42] shows that the effectiveness of the model can be further improved by incorporating weights to each document pairs for the cost function. The weights for document pairs, $|\Delta Z_{ijk}|$, are the differences in metrics. LambdaMART is a boosted regression tree implementation of LambdaRank [42]. The cost function is computed as:

$$C = \sum_{q_i \in Q} \sum_{j,k} |\Delta Z_{ijk}| (I(r_{ij} > r_{ik}) \log(1 + e^{-\sigma(s_{ij}-s_{ik})}) + I(r_{ij} < r_{ik}) \log(1 + e^{-\sigma(s_{ik}-s_{ij})})) \tag{2.32}$$

At each iteration of LambdaMART, the algorithm fits a regression tree whose predicting labels are the lambdas of current model. Given a document d_{ij} that is more relevant than another document d_{ik}, the gradient of the cost with respect to the relevance score for d_{ij}, $s_{ij} = f(\mathbf{x}_{ij}, \theta)$, is computed as:

$$\lambda_{ijk} = \frac{\partial C_{i,j,k}}{s_{ij}} = \frac{-\sigma}{1 + e^{\sigma(s_{ij}-s_{ik})}} |\Delta Z_{ijk}| \tag{2.33}$$

For an individual document in the training set, the gradient of the cost function with respect to its current model score is computed as:

$$\begin{aligned}\lambda_{ij} = \frac{\partial C_{ij}}{\partial s_{ij}} &= \sum_{k:k \neq j} |\Delta Z_{ijk}| (I(r_{ij} > r_{ik}) \frac{-\sigma}{1 + e^{\sigma(s_{ij}-s_{ik})}} + I(r_{ij} < r_{ik}) \frac{-\sigma}{1 + e^{\sigma(s_{ik}-s_{ij})}}) \\ &= \sum_{k:k \neq j} I(r_{ij} > r_{ik}) \lambda_{ijk} + I(r_{ij} < r_{ik}) \lambda_{ikj} \\ &= \sum_{k:k \neq j} I(r_{ij} > r_{ik}) \lambda_{ijk} - I(r_{ij} < r_{ik}) \lambda_{ijk}\end{aligned} \tag{2.34}$$

Note that ΔZ_{ijk} will be zero if the d_{ij} and d_{ik} have the same relevance labels. As a result, pairs of documents with equal relevance labels will not be affect λ_{ij}.

At each iteration of LambdaMART, the algorithm uses the λs as the training label for each document and the predicted value at each leaf node of the current tree is updated as:

$$\gamma_{km} = \frac{\sum_{d_{ij} \in R_{km}} \frac{\partial C_{ij}}{\partial s_{ij}}}{\sum_{d_{ij} \in R_{km}} \frac{\partial^2 C_{ij}}{\partial s_{ij}^2}} = \frac{\sum_{d_{ij} \in R_{km}} \lambda_{ij}}{\sum_{d_{ij} \in R_{km}} \frac{\partial \lambda_{ij}}{\partial s_{ij}}} \tag{2.35}$$

where R_{km} denotes the region of the m^{th} leaf of the k^{th} tree, γ_{km} is the value for R_{km},. The tree will be updated as: $f_k = f_{k-1}(\vec{x}) + \eta \sum_m \gamma_{km} I(x_i \in R_m)$, where η is the learning rate.

2.2.5 Learning-to-Rank Datasets

Different L2R test collections have been published to help the investigation of L2R algorithms. Table 2.1 lists some of the existing L2R test collections.

LETOR3.0 [37] is an L2R collection developed by the Microsoft Bing search team based on the TREC 2003 Web Track [54], the TREC 2004 Web Track [55], and the medical collection OSHUMED [56]. Both the TREC 2003 Web Track and the TREC 2004 Web Track have three different topic sets that target different types of information needs: topic distillation (TD), name-page finding (NP), and home-page finding (HP). The task of TD is to find relevant homepages for a broad query, while both NP and HP queries are navigational queries. Both navigational tasks look to find a particular webpage for each query. However, NP queries specify the name of the page, while HP queries do not have the name of the homepage. The document corpus used by both the two web tracks is the GOV collection, which is a corpus of web pages with the '.gov' domain, published by TREC[2]. The six-topic set from the Gov collection in LETOR3.0 has a small number of the queries, while nearly 1k documents were pooled per query. OSHUMED is a medical collection with 106 queries with each query containing 152.3 documents on average. The labels in LETOR3.0 are all binary, making each document as either relevant or irrelevant.

LETOR4.0[3] was built using the million query tracks [57, 58] from TREC 2007 and TREC 2008, which corresponds to query sets in LETOR4.0: MQ2007 and MQ2008. The GOV2 collection was used as the corpus for LETOR4.0. The average number of documents pooled for each query in MQ2007 is 41.1, while it is 19.4 in MQ2008. It is worth mentioning that the pooling methods used for LETOR4.0 are different from LETOR3.0. LETOR3.0 used BM25 as the base retrieval function and keeps the first 1k documents for ranking/judging, whereas LETOR4.0 used two different methods for pooling: Minimal Test Collections (MTC) and statAP [57]. Both are random sampling processes aimed at maximizing the information in the test collection to allow for better evaluation. The details of the two methods can be found in Allan et al. [58]. The relevance labels for query document pairs are judged at three levels, from 0 to 2.

The Microsoft learning-to-rank dataset (MSLR)[4] is a large L2R test collection developed

[2]http://ir.dcs.gla.ac.uk/test_collections/

[3]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/

[4]https://www.microsoft.com/en-us/research/project/mslr/

Table 2.1: L2R test collections

Dataset	Document Corpus	Query Set	Query Size	#Features	Avg #Docs	Relevance
LETOR3.0	GOV	TD2003	50	64	991.0	Binary
		TD2004	75		984.5	
		HP2003	150		984.0	
		HP2004	75		992.1	
		NP2003	150		981.2	
		NP2004	75		988.6	
	OHSUMED	-	106	45	152.3	Binary
LETOR4.0	GOV2	MQ2007	1,692	46	41.1	3-level
		MQ2008	784	46	19.4	
MSLR	Bing search engine	MSLR-10K	10k	136	120.0	5-level (0-4)
		MSLR-30K	30k		119.6	
Yahoo! L2R	Yahoo! search engine	Set 1 (US)	29,919	519	23.7	5-level (0-4)
		Set 2 (An Asian country)	6,330	596 (Rank normalized)	27.3	
Yandex2009	Yandex search engine	-	9,124	245	10.6	5-level (0-4)
Istella	TISCALI ITALIA search engine	Istella LETOR	33,018	220	316	5-level (0-4)
		Istella-S LETOR	33,018		103	

based on Bing's retired collections. MSLR contains two collections, MSLR-30K and MSLR-10K. MSLR-10K is composed of 30k queries, whereas MSLR-10K is a small sample of MSLR-30K, which contains 10k queries. The average pooling depth is 120 documents for queries in MSLR. The documents pooled for queries are judged at 5-levels, from irrelevant (0) to perfectly relevant (4).

The Yahoo! learning-to-rank (Yahoo!L2R) [59] is an L2R collection published by Yahoo!. Yahoo!L2R consists of two collections: Set 1 and Set 2. Set 1 and Set 2 are built to facilitate research on TR. Set 1 was built based on the US web search market while Set 2 was built on an Asian web search market. Set 1 has many more queries than Set 2. The relevance of the documents was also judged at five levels. Yahoo!L2R has a rather shallow pooling depth, with only 23.9 documents judged per query. The number of features is different for the two collections. There are 519 and 596 anonymous[5] features respectively in the two collections, with some overlap. All the features are rank-normalized as:

$$\tilde{\mathbf{x}}_i := \frac{1}{n-1}|\{j, \mathbf{x}_j < \mathbf{x}_i\}| \tag{2.36}$$

The total number of distinct features is 700, and the values for missing features are set as 0.

Yandex Internet Mathematics 2009 (Yandex2009) is the test collection by the Russian search engine company Yandex. It contains 9124 queries with 10.6 sampled documents per query on average. The relevance labels are also valued from 0 to 4, while the features are anonymous.

Istella [60, 61] is a test collection from an Italian search engine that is used to study the efficiency issues of L2R. It contains two datasets, an Istella LETOR dataset, and its subset Istella-s LETOR dataset. Istella LETOR datasets contain 33,018 queries with 316 documents on average per query. The subset collection Istella-s is randomly sampled from the document pool. As a result, Istella-s contains the same number of queries, while the average document size is 103.

[5]By 'anonymous' here we mean that the functions used to compute the feature values are unknown.

Although there are six sets of L2R collections, none of the existing collections are designed for evaluating TR algorithms, except for the Yahoo!L2R datasets. In the Yahoo!L2R collection, Set 1 and Set 2 are built for cross-domain TR. Information describing the features in terms of the function used to compute them or even the one-to-one correspondence between the features for the two sets is not provided, which makes it hard to conduct further analysis. Moreover, the number of queries in Set 2 is very small and does not include any additional unlabeled data. As a result, Set 2 might be a biased sample from the second market, which makes it harder for knowledge transfer to be transformed as well as for performance evaluation to be addressed. In other collections, no explicit scenarios exist where we can apply TR, due to the limitation of the unknown/anonymous features, or the small scales of the datasets. Later in this thesis, we have manually developed some environments for evaluating TR algorithms. Furthermore, the fact that no query and document information in the dataset is available for those large collections makes it difficult for us to explore other context-based methodologies for TR. As a result, in this thesis, we have limited our study of TR to feature distribution-based approaches.

2.3 Transfer Learning

The assumption made by almost all supervised learning algorithms is that the set of data used for training is a representative sample drawn from the same population as the test set. It is hoped that the predictive model trained on the training set will generalize to the whole population. As a result, conventionally, a new learning model will be trained to accomplish a new task. The training process will require massive training data. However, obtaining labels for training can be very expensive. For example, in IR, assessors will need to be recruited to annotate relevance labels for hundreds of thousands of query document pairs. Transfer learning [9, 62] looks to train or improve a learning model for a new/target task by transferring knowledge from an existing/source task, which provides a solution to the lack-of-training-labels problem.

As mentioned before, machine learning algorithms assume that the training and test set come from the same distribution. The performance of the trained model will be degraded if it is applied to a new task with different data distribution. The issue is commonly referred to as the dataset shift problem [8], where the distribution of the training and test data are different. Typically the data used for classification and regression problems form a joint distribution $P(X, Y)$, where X is the input feature space, y is the label to be predicted, and $P(X)$ is the marginal probability distribution for the feature space. Overfitting is the problem where the model closely fits the training sample, but does not generalise to the population. Overfitting can not be measured when moving to a new population (distribution), since the training data was not sampled from it. If a model generalises well to a population, it might still behave poorly when moving to a new population. As the joint distribution $P(X, Y)$ is controlled by both the marginal probability $P(X)$ and the conditional probability $P(Y|X)$, the changes to the data distribution can manifest in various ways. Solutions to the transfer learning problem are varied as they may tackle different aspects of the distribution change.

Many studies have focused on solving the problem where the source and target collection has the same conditional distribution but different marginal distribution $P^{ta}(X) \neq P^{so}(X), P^{ta}(Y|X) = P^{so}(Y|X)$[6]. This problem is commonly referred to as *Covariate Shift* [63] in the literature. For conventional machine learning algorithms, covariate shift can be measured and reduced by density ratio estimation [63, 64]. However, it is not possible to measure the covariate shift for learning to rank datasets as we will show later, due to the complexity of the data generating process for learning to rank datasets. Different from Covariate Shift, there might be other causes for the distribution change, for example, *Domain Adaptation* [12], where the conditional probability of the source and target are different, while the marginal distribution remains the same, $P^{so}(Y|X) \neq P^{ta}(Y|X), P^{so}(X) = P^{ta}(X)$. Both Covariate Shift and Domain Adaptation can happen at the same time, $P^{so}(Y|X) \neq P^{ta}(Y|X), P^{so}(X) \neq P^{ta}(X)$, which can be challenging to tackle [65]. Both of the cases will be explained in detail in the next chapter. In some cases, transfer learning looks to solve the situation when the feature spaces of the two collections are different, which is also known as *Heterogeneous Transfer Learning* [66].

[6]Note that throughout this thesis, we will use the superscript *ta* to denote parameters or elements from the target collection and use *so* to denote those from the source collection.

2.4 Transfer Ranking

Transfer Ranking (TR) is an application of transfer learning to L2R algorithms, which aims to help train a new ranking function for a new collection by incorporating knowledge from a related ranking task. In this section, we will provide a formal definition of the TR problem and then review previous related works.

2.4.1 Problem Definition

Given a target collection $\mathcal{L}^{ta}$, which consists of a small set of labeled query document pairs, $\mathcal{L}_l^{ta} = (X_l^{ta}, R_l^{ta})$[7] and a large set of unlabeled query document pairs, $\mathcal{L}_u^{ta} = (X_u^{ta}, R_u^{ta})$, where $R_u^{ta} = \emptyset$, the aim of the TR is to train or adapt a new ranking function for the target collection by incorporating knowledge from a related source collection $\mathbf{L}^{so}$, which is made up of a large number of labeled query document pairs (X^{so}, R^{so}). TR aims at tackling the scenario where the labeled target set is much smaller than the unlabeled target set, $|\mathcal{L}_u^{ta}| \ll |\mathcal{L}_l^{ta}|$. *Unsupervised TR* is the scenario when there are no labels in the target collection, $\mathcal{L}_l^{ta} = \emptyset$, and *Supervised TR* is the case when there is a small amount of labeled query-document pairs, $\mathcal{L}_l^{ta} \neq \emptyset$.

2.4.2 Relatedness of L2R datasets

TR aims to train a new ranking function for a new collection, by transferring knowledge from another "related" collection. However, the definition of "relatedness" is not clearly defined in most TR literature. When an unrelated collection is used as the source collection, *"negative transfer"* [67] could occur, where noise is introduced to the source dataset, degrading the performance of source ranker on the target dataset. Geng et al. [20] proposed a concept called "adaptability", which measures the benefit that a source collection can potentially bring to a target collection. They proposed to estimate the ranking adaptability by averaging the correlation between the ranked lists produced by the source model and the ground-truth lists, i.e., those based on human judgments. However, measuring this requires relevance judgments from the target collection and depends on the labeled queries. Thus comprehensive studies on the relatedness of L2R datasets are still needed to increase our understanding of what is possible in this scenario.

[7]We use subscript l to denote labeled data and u to denote unlabeled instances.

2.4.3 Categories of Transfer Ranking

The objectives of all TR algorithms are the same, namely, to improve the ranking effectiveness on a new collection by making use of labeled data from another collection. However, the transfer settings of TR vary from case to case. There is no general classification of TR settings. As mentioned above, the TR can be classified as supervised and unsupervised TR according to the availability of relevance labels in the target collection. TR problems can also be classified into homogeneous and heterogeneous TR based on the similarity in feature representation. Homogeneous TR is the case where the feature space of the source and target collections are the same, while in heterogeneous TR, the feature spaces of the datasets are different. The latter case necessitates a feature engineering process to connect the different feature spaces. In some cases, there may exist multiple source collections, or the source and target collections may require training simultaneously.

2.4.4 Challenges of Transfer Ranking

Although transfer learning has been widely studied and applied in other contexts such as natural language processing and image recognition, its application to IR requires further investigation. The study of TR has been limited for many reasons which we discuss below.

The training data for L2R can be required from three levels, namely the document level, pair level, and query level, which differentiates the problem from conventional machine learning (regression) tasks. As a result, many existing solutions from transfer learning techniques cannot be directly used, as these algorithms treat each data point as a single training instance. The objective of an L2R algorithm is to maximize the effectiveness of a ranking function on a set of queries. However, since most ranking effectiveness metrics are not smooth, we cannot calculate the derivative of the metrics with respect to the model scores, which we need in order to update the parameters of the ranking function. Most algorithms thus make use of some approximated objective function, which also contributes to the difficulty of TR problems. Moreover, due to efficiency considerations, the documents returned for a particular query are first retrieved using a base retrieval model like BM25 and then pooled at a certain depth. Therefore, the fact that the ranked instances are a subset of the remaining examples makes it even harder to analyze and quantify the differences in the data distribution between the source and target.

TABLE 2.2: Difference of L2R algorithms and conventional machine learning algorithms

Type	Conventional Machine Learning	Learning to Rank
Input Data	Single data instance	Document-level: Query-document feature vector Pair-level: Pair of query-document feature vectors Query-level: A list of query-document feature vectors
Objective Function	Loss on individual samples	Surrogates of the ranking metric
Data Generating Process	Usually controlled by a few parameters	Controlled by too many factors, impossible to model

The differences of the L2R algorithms and conventional machine learning algorithms are summarised in Table 2.2.

The dataset shift for L2R datasets is different from conventional machine learning datasets since it can be affected by many factors, including the documents, the queries, the mapping between documents and relevance labels and the domain parameters. Measuring the impact of these factors on the generalization of L2R algorithms is difficult as there are so few such benchmark collection for experiments.

2.5 Literature Review

TR is a rather new area that has not been extensively studied. In this section, we review previous related works on TR and discuss the knowledge gained from this work, as well as its limitations.

Before the emergence of TR, there were some attempts to develop semi-supervised L2R algorithms ([68–71]) that try to improve ranking effectiveness by leveraging unlabeled data. Semi-supervised L2R algorithms assume that the labeled training set is not sufficient for training reliable models. Semi-supervised L2R algorithms tackle this problem by leveraging (large quantities of) unlabeled training instances in the collection. TR, on the other hand, tries to incorporate knowledge from a related labeled collection. Ideally, TR should perform better than semi-supervised L2R since it also transfers knowledge about relevance labels from the other collection. In some cases, there are no relevance labels in the target collection, in which case semi-supervised L2R techniques cannot be used. If some other related collection is present, TR can be applied to help train a more general ranking function.

Solutions to the TR problem vary on different levels. However, the core idea of TR is to try to migrate or eliminate the differences between two different datasets (the source and the target). The most common approaches to resolving TR are that by Chen et al. [17], Gao et al. [18], Ren et al. [72], Cai et al. [73], sample selection [19, 74, 75], and feature engineering [17, 74, 76], but other miscellaneous methods also exist [23, 77, 78].

All the TR literature has focused on knowledge transfer between different ranking tasks, however, the resources that are available for transferring differ from case to case. Many of the previous studies [11, 19, 74, 77–83] attempt to solve the supervised TR problem, where the task is to transfer the ranking function from a source collection to a target collection that contains only a small number of relevance judgements. Among the studies, some [11, 80, 82, 83] focused on the heterogeneous TR scenario, where the feature spaces of the source and target collections are different. Differently, Gao et al. [18],Geurts and Louppe [84], Gao and Yang [76] and Macdonald et al. [85] attempted to solve the problem under when there are no relevance labels in the target collection, which is also known as the *unsupervised TR* problem. Apart from these mainstream problems, Goswami et al. [23] have looked at transferring from multiple sources, and Cai et al. [75] used transfer learning as a tool for active learning in learning to rank. There are also some works [15, 86, 87] that have attempted to transfer knowledge across different ranking task, namely multi-task learning to rank. All the existing studies on TR that we are aware of are given in Table 2.3.

In the following sections, we will review different algorithms in the order of types in Table 2.3.

2.5.1 Supervised Transfer Ranking

Most of the existing work on TR has belonged to the category of supervised TR. The reason why supervised TR attracted more attention than unsupervised TR is that the existing relevance labels in the target collection can calibrate the direction for adapting the source ranker. The feature space of the source and target collection can either be homogeneous or heterogeneous, which contributes to the difficulty of implementing generic algorithms in solving all situations.

Table 2.3: Literature classification

Problem	Feature Space	Solution	Literature
Supervised TR	Homogeneous	Instance Weighting	Chen et al. [17]
		Sample Selection	Chen et al. [74] Duh and Fujino [19]
		Feature Engineering	Chen et al. [74] Chen et al. [17] Geurts and Louppe [84] Zhou et al. [81] Macdonald et al. [85] Bahadori et al. [83]
		Model Adaptation	Gao et al. [79] Chen et al. [77] Wu et al. [78] Bai et al. [88] Wang et al. [89]
		Co-regularization	Geng et al. [20]
	Heterogeneous	Feature Engineering	Wang et al. [80] Geng et al. [20] Long et al. [82]
Unsupervised TR	Homogeneous	Instance Weighting	Gao et al. [18] Ren et al. [72] Cai et al. [73]
		Weak Supervision	Gao and Yang [76]
Multi-source Unsupervised TR	Homogeneous	Weak Supervision	Goswami et al. [23]
Multi-task L2R	Homogeneous		Bai et al. [86] Chapelle et al. [15] Tang and Hall [90] Chapelle et al. [87]
Transfer Active L2R	Homogeneous		Cai et al. [75]

2.5.1.1 Homogeneous Supervised TR

As mentioned before, TR is trying to solve the dataset shift problem, where there is a mismatch between the source and target data distribution, $P^{so}(X,Y) \neq P^{ta}(X,Y)$. Changes in the joint distribution across collections can result from both covariate shift ($P^{so}(X) \neq P^{ta}(X)$) and from changes in the class mapping function ($P^{so}(Y|X) \neq P^{ta}(Y|X)$). Different solutions have made different assumptions on the causes of dataset shift.

Instance Weighting A common solution for covariate shift is *Instance Weighting* [91], which re-weights the instances in the source collection to simulate the data distribution in the target collection. The word instance here refers to a single data point in machine learning. However, for L2R, instances may refer to three different things according to the ranking model being trained: documents, document pairs, or queries. The weighted source collection data is then used to train a target collection-specific ranking function.

Instance weighting is a widely used approach in transfer learning and is proposed to tackle the covariate shift problem. Most supervised learning algorithms follow a risk minimization framework as follows:

$$\hat{\theta} = arg \min_{\theta} \int \int p(\mathbf{x}, y)\ell(h(\mathbf{x}; \theta), y)d\mathbf{x}dy \tag{2.37}$$

where $p(\mathbf{x}, y)$ is the true density of the instance $(\mathbf{x}, y)$ in the collection, $h(.)$ is a hypothesis, and $\ell(.)$ denotes a loss function. Usually the true density is not known and instead empirical data is used to estimate the risk:

$$\hat{\theta} = arg \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(h(\mathbf{x}_i; \theta), y_i), \text{where } \forall i, (\mathbf{x}_i, y_i) \sim p(\mathbf{x}, y) \tag{2.38}$$

To train a model for the target collection, one needs to minimize the risk over the target collection as:

$$\hat{\theta} = arg \min_{\theta} \int \int p^{ta}(\mathbf{x}, y)\ell(h(\mathbf{x}; \theta), y)dxdy \tag{2.39}$$

Since the target data is not sufficient for training the model, the source data is used for training. However, as the source data may be distributed differently from the target, each source instance will need to be weighted in order to make source distribution closer to the target collection:

$$\hat{\theta} = arg \min_{\theta} \int \int \frac{p^{ta}(\mathbf{x}, y)}{p^{so}(\mathbf{x}, y)} p^{so}(\mathbf{x}, y)\ell(h(\mathbf{x}; \theta), y)d\mathbf{x}dy \tag{2.40}$$

In practice, the expected risk is estimated by the empirical risk:

$$\begin{aligned}\hat{\theta} = \underset{\theta}{arg\ min}\ \frac{1}{N^{ta}} &\sum_{(\mathbf{x},y)\sim P^{ta}(\mathbf{x},y)} p^{ta}(\mathbf{x},y)\ell(h(\mathbf{x};\theta),y) \\ +\frac{1}{N^{so}} &\sum_{(\mathbf{x},y)\sim P^{so}(X,Y)} w(\mathbf{x},y)p^{so}(\mathbf{x},y)\ell(h(\mathbf{x};\theta),y)\end{aligned} \tag{2.41}$$

As we will show later in the unsupervised TR scenario, $w(\mathbf{x},y) \approx \frac{p^{ta}(\mathbf{x},y)}{p^{so}(\mathbf{x},y)}$.

The $CLRank_{ins}$ TR algorithm [17] made the first attempt to use instance weighting for supervised TR problems. $CLRank_{ins}$ was a pair-wise TR algorithm that was designed for use with RankSVM [38]. As a result, the instances for the algorithm are pairs of documents in the same list. The algorithm first trains a model $\hat{f^{ta}}$ using labeled data from target collection. For each document pair $(\mathbf{x}_{ij}, \mathbf{x}_{ik})$ in the source collection, the weight is computed as:

$$w(\mathbf{x}_{ij}, \mathbf{x}_{ik}) = \begin{cases} 0, & \hat{f^{ta}}(\mathbf{x}_{ij}, \mathbf{x}_{ik}) \neq I(r_{ij} > r_{ik}) \\ prec(q_i, \hat{f^{ta}}), & \hat{f^{ta}}(\mathbf{x}_{ij}, \mathbf{x}_{ik}) = I(r_{ij} > r_{ik}) \end{cases} \tag{2.42}$$

where $prec(q_i, \hat{f^{ta}})$ denotes the precision of the rank function $\hat{f^{ta}}$ on the preferences of pairs of documents in query q_i. $I(r_{ij} > r_{ik})$ denote the ground-truth preferences of $\mathbf{x}_{ij}$ and $\mathbf{x}_{ik}$, while $\hat{f^{ta}}(\mathbf{x}_{ij}, \mathbf{x}_{ik})$ is the preference predicted by the ranker. The weights were then used with Equation 2.42 to train the transferred model.

$CLRank_{ins}$ was tested on LETOR3.0 datasets as well as AP and WSJ datasets from TREC Collections [92]. The algorithm did not show significant improvement over the model trained with a small number of queries from the target collection (called the Target-only Model).

Reliably estimating the appropriate weights for training instances for L2R is very difficult. Under the supervised TR scenario, if a biased weighting strategy is used, more noise will be introduced to the training set and can result in poor transfer effectiveness. (In other words, a negative transfer can occur.)

Sample Selection Apart from instance weighting, an alternative solution to supervised TR is sample selection (aka instance selection [91]). The idea behind sample selection is to enrich the target training set by injecting source training instances that coincide with the underline distribution of the target collection.

Sample selection can capture the functional change, where $P^{so}(Y|X) \neq P^{ta}(Y|X)$. Similar to instance weighting approaches, sample selection requires a procedure to assign importance weights to source instances, so that more important source instances are selected for adding to the training set.

Chen et al. [77] proposed a sample selection method called "TransRank". TransRank computes importance weights for source queries using a method called "ranking direction" and injects the top k source queries to the target collection for training. During the weighting step, queries are represented by the "ranking direction" (rd), which is the vector from the centroid of irrelevant documents to the centroid of the relevant documents:

$$rd = \frac{\sum_{r_{ij}>0} \mathbf{x}_{ij}}{|\{\mathbf{x}_{ij}|r_{ij} > 0\}|} - \frac{\sum_{r_{ij}=0} \mathbf{x}_{ij}}{|\{\mathbf{x}_{ij}|r_{ij} = 0\}|} \tag{2.43}$$

TransRank calculates query importance weights with a utility function comprised of the separation score and similarity score to the target collection. The algorithm first models the target ranking direction (rd^{ta}) using the vector pointing from centroid of all irrelevant documents to all the relevant documents in the target collection as the target ranking direction.

Next, for a source query, it computes the cosine similarity of its ranking direction and the target ranking direction as the similarity score to the target collection ($cosine(rd^{ta}, rd_i^{so})$).

Furthermore, a separation score for the query is computed to indicate how separable the feature vectors of the relevant and irrelevant documents are in the query. The query importance scores were then used for selecting top k source queries for transferring. The algorithm later uses a feature augmentation method to minimize the distribution difference at the feature level. TransRank was tested on OSHUMED, transferring from WSJ and AP datasets [92] and in both cases showed a significant improvement over the model trained with target data only, it also does better than the model trained with mixed data from both collections. The computational cost of the method is very high and the concept of ranking direction needs further investigation. Moreover, it is not clear how the method can be applied for multigrade relevance.

Duh and Fujino [19] provide another approach to supervised TR with sample selection. They try to select the most similar queries from source domain according to their "relatedness" to the target domain. They model the query with the influence of each feature to the ranking. Thus the algorithm trains a linear ranker for each query in both collections and represents each query with the weights of each feature in the ranker. The relatedness of queries is calculated using the density ratio estimation approach KEILP[63]. The most related queries are selected for training. Similar to supervised instance weighting approaches, the effectiveness of the Duh and Fujino [19] approach depends on the available queries that have labels. The algorithm was tested on both the LETOR3.0 and Yahoo datasets and showed significant improvements over the Target-only Model. Since the Yahoo dataset has a large number of queries, the algorithm shows solid improvements in this collection. However, this approach requires significant computational time as well as memory to train rankers for each query.

One critical problem of sample selection is the threshold that is used to decide which query to choose. Chen et al. [77] showed that the performance of the algorithm will reach a peak once a certain proportion of queries were chosen and then will drop after more queries are selected. The work does not provide any solution for the selection of the appropriate threshold. The approach by Duh and Fujino [19] has a similar limitation to Chen et al. [77], and they provide no conclusion on how to choose the number of queries for training. For the sample selection approach, more studies may be needed to develop a methodology to help choose the appropriate threshold.

Feature Engineering Feature engineering is a transfer learning technique which looks to identify a feature space that can minimize the difference between the data distribution of two datasets. Two datasets may differ in one feature space but be similar in another feature space. Thus, the transfer can be conducted in the identified feature space, or "common feature space". Imagine two L2R datasets have the same feature space, which is made up of only three features: tf-idf, PageRank, and doclength. The data distribution of <td-idf, PageRank, doclength> is different between source collection and target collection. Feature engineering tries to map source and target collection data into a different feature space so that the difference can be reduced.

The common feature space can be detected by *Subspace Finding*, *Feature Augmentation*, or *Latent Space Learning*. Subspace finding is a straightforward approach that studies the distribution of each feature dimension and finds a subspace of the original feature space, in which the two datasets have a similar distribution, as in the example we described. Feature augmentation is a feature construction method that combines both common features and target-task-specific features. Latent space learning is a technique that learns a latent feature space, in which the difference between the source and target collection is minimal.

The feature engineering methods have also been used for supervised TR. They can not only be used for homogeneous supervised TR but also tackle the heterogeneous supervised TR problem.

The TransRank [77] algorithm discussed in the last section also involves a feature engineering step that uses the feature augmentation method developed by Daum III [93]. Daum III [93] constructs a common feature space for both the source and target collection by a mechanism called feature augmentation. Assume X^{ta} and X^{so} are in the same space $\mathbb{R}^{\delta}$ $(\delta > 0)$[8]. The feature augmentation method maintains three versions of feature space: *general version*$(\mathbf{x}^{g})$, *source-specific version*$(\mathbf{x}^{so})$ and *target-specific version*$(\mathbf{x}^{ta})$. The mapped feature space is $< \mathbf{x}^{g}, \mathbf{x}^{so}, \mathbf{x}^{ta} >$. For source collection data $\mathbf{x}$, the algorithm map the original feature to $< \mathbf{x}, \mathbf{x}, \mathbf{0} >$, and for target collection data, the feature space is mapped to $< \mathbf{x}, \mathbf{0}, \mathbf{x} >$, where $\mathbf{0}$ is a zero vector $< 0, 0, 0, ... >$ with the same length as the original feature spaces. For example, if the feature space for both collections is <LM, tf-idf, pr>, where *LM* is the language model, tf-idf is the TF-IDF score, and pr is the PageRank score. If we only have one data in each set, for instance, $< 0.2, 0.15, 0.1 >$ in the source collection, and $< 0.5, 0.3, 1 >$ in the target collection, the feature space would be $< 0.2, 0.15, 0.1, 0.2, 0.15, 0.1, 0, 0, 0 >$ and $< 0.5, 0.3, 1, 0, 0, 0, 0.5, 0.3, 1 >$ respectively. The feature augmentation method naturally combines the three versions of features into training. The trained model is a combination of the three versions of features. Thus, the final ranker will also be a combination of the general and target-specific model since the features are 0s in the source version features. The algorithm is easy to implement. TransRank was tested by transferring between three different collections, OHSUMED in LETOR3.0 and two other L2R collections created using the Associate Press (AP) and Wall Street Journal (WSJ) corpus and queries from TREC2 to TREC5 with the same features implemented for OHSUMED. The results show that TransRank can outperform

[8] δ is the number of dimension

a RankingSVM model trained on the target data only as well as the model trained with a mix of both the source and target data.

$CLRank_{feat}$ [17] used the multi-task feature learning method proposed by Argyriou et al. [16]. The algorithm learns a high-level feature representation for both collections to minimize the difference. The high-level features are linear combinations of the original features that are learned from both collections. Different from Argyriou et al. [16], $CLRank_{feat}$ used pair-wise loss function. The algorithm was tested on LETOR3.0 [37], WSJ and AP [92]. The result showed significant improvements over the models trained with small numbers of target data. Moreover, it shows advantage over their instance weighting approach, $CLRank_{ins}$.

The *Transfer Boosting* algorithm proposed by Zhou et al. [81] is also a feature engineering method that learns a set of "super-features" for the original features in both collections. The algorithm learns the super-features together with their coefficient weights through a boosting algorithm. At each iteration, the algorithm learns both the super-features and the weighting using the following objective function:

$$\underset{g(\mathbf{x}),w^{ta},w^{so}}{argmin} \sum\nolimits_{(\mathbf{x},y)\ P^{so}(X,Y)} \ell(f(w^{so},g(\mathbf{x})),y) + c\sum\nolimits_{(\mathbf{x},y)\ P^{ta}(X,Y)} \ell(f(w^{ta},g(\mathbf{x})),y) \tag{2.44}$$

where $g(\mathbf{x})$ are the super-features, w^{ta} are the weights or coefficients of these super-features in the target collection and w^{so} are the weights for the source collection. The parameter c controls the proportion of target collection data. The optimization is then achieved following a stage-wise fashion with a boosting algorithm. It uses the ranking function trained in the last iteration to find the super-features and their coefficient weights, and then use the super-features and coefficient weights to update the ranker. Several results on different transferring settings suggest that Transfer Boosting can help improve the performance of the target model. However, one problem with the algorithm is the parameter c in Equation 2.44. As demonstrated in the paper, the performance of the algorithm varies with different settings of the parameter c, and it does not correlate with the performance.

In Geurts and Louppe [84], the authors tried using different TR approaches by utilizing different parts of data from the source and target, as well the features in both collections. They found that adding the similarity scores predicted using the source model as an extra feature, and training on the target data by combining the ranking models trained with the source model as a feature can improve the effectiveness of the target ranker. Inspired by the idea, Macdonald et al. [85] further tested the algorithm using the ClueWeb data and showed significant improvements. However, the method is limited as it only works for the dataset where the source and target data share some common features and the improvements were small.

Model Adaptation Apart from the instance and feature-based approaches, some other studies have attempted to adapt the existing source model with a few labeled training data from the target collection. Most model adaptation-based algorithms can reuse the source ranker as a base model or as a feature for training. As most model adaptation algorithms rely on the source model instead of source training data, they can be useful when the data in the source cannot be accessed due to privacy restrictions.

Gao et al. [79] have explored two different types of model adaptation - one is the model interpolation and the other is the boosting-based approach. The model interpolation-based method learns an ensemble of both source models and target models:

$$f(\mathbf{x}) = \sum_{f_i \in \{f^{so}\} \cup \{f^{ta}\}} \alpha_i f_i(\mathbf{x}) \tag{2.45}$$

The coefficient weights of the models can be learned with the validation set via L2R models or cast as a multi-dimensional optimization problem. It is obvious that the algorithm will depend on the representative of the validation set. In order to improve performance, the authors employed a method to expand the validation set by selecting samples from the source collection similar to the validation set.

The other solution, "error-driven learning", LambdaBoost and LambdaSMART [78], is using the source model as the base model and then using gradient boosted tree to update the model based on the target data. Both of the algorithms have some limitations as they depend on the labeled training data from the target collection. The results showed that the model interpolation based model performs better when the source and target collection are very different, while the boosting method works better when the source and target collection are similar.

Apart from the learning- and optimization-based adaptation method, others have looked at ways to tune source models in order to fit the target collection. For example Chen et al. [77] and Bai et al. [88] proposed algorithms that were designed specifically for tree-based L2R algorithms. Chen et al. [77] proposed an algorithm called Trada, which tunes the tree splits as well as the weights based on the tree model trained on the source collection with the labeled data from the target collection. The experiment result shows that the algorithm can improve the effectiveness of the Target-only Model but then it is affected significantly by the number of labeled queries from the target collection. The algorithm in Bai et al. [88] is an extension to Trada in that it uses pairwise preferences data to guide the tree adaptation. The result showed a significant improvement over Trada.

An alternative approach for model adaptation is via coefficient transformation. For example, in Wang et al. [89], the authors developed an algorithm that can learn a transformation matrix which transforms the original coefficients of a global ranking model to enable personalization. The transformation of coefficients is done by scaling and shifting groups of features and the ranking function is then transformed into:

$$f^{ta}(\mathbf{x}) = \sum_{k=1}^{K} \sum_{g(i)=k} (a_k w_i^{so} + b_k)\mathbf{x}_i \tag{2.46}$$

where $g(.)$ is a grouping function, $g(i) = k$ denotes $\forall$ feature i in group k, a_k is the scaling factor for group k features, and b_k is the shifting factor for group k. The feature grouping can be done either manually or automatically with co-clustering or k-means. The proposed algorithms were tested on large-scale query logs from a commercial search engine and showed significant improvements over both the source and target models and other supervised TR algorithms.

Co-regularization Besides methods to reduce the difference between the data distribution of the source and target collection, others have investigated solutions for supervised TR using knowledge learned from the source collection as prior information in the target model. For example, in Geng et al. [20], the authors developed an algorithm named *RA-SVM*, which utilized the parameters in the source ranker in the regularization form of RankSVM:

$$\begin{aligned} \vec{\theta} = \underset{\vec{\theta}}{arg\ min} \quad & \frac{1-\delta}{2}||\vec{\theta}||^2 + \frac{\delta}{2}||\vec{\theta} - \vec{\theta}^{so}||^2 + C\sum_{ijk}\xi_{ijk} \\ s.t. \quad & \mathbf{x}_{ij}\vec{\theta}^{\top} - \mathbf{x}_{ik}\vec{\theta}^{\top} \geq 1 - \xi_{ijk}, \\ & \xi_{ijk} \geq 0 \\ for \quad & \forall\{\mathbf{x}_{ij}, \mathbf{x}_{ik}\} \in \mathcal{L}_l^{ta}\ with\ r_{ij} > r_{ik} \end{aligned} \tag{2.47}$$

where $\vec{\theta}^{so}$ is the parameters for the source ranker, $\vec{\theta}$ is the learned parameters target ranker, $\delta \in [0,1]$ is a parameter that controls the regularization $||\vec{\theta}||$ and the similarity of the source ranker.

The assumption is that the source ranker is a global ranking function. Moreover, the target ranker and source ranker will have a similar shape in the function space. The algorithm was tested with two TR settings: 1) transferring from TD2003 to TD2004 in LETOR3.0, and 2) transferring from Web Page Search to Image Search. The results demonstrated that *RA-SVM* outperforms the source ranker, the ranker trained with target data only, and the ranker trained with both the source and target data. However, *RA-SVM* assumes that the source and target model shares the same parameter space, which may not be useful when it comes to more complicated models like LambdaMART.

2.5.1.2 Supervised Heterogeneous TR

The literature in the last section is mainly focused on scenarios where the source and target collections share the same feature space. However, in some cases, the source and target collections may have heterogeneous features, in which case the aforementioned TR algorithm cannot fit. This could arise when a model is needed to be transferred to a different domain, for example from a web search domain to a job search domain. In this section, we review a few attempts that have been made to tackle the heterogeneous TR problem.

The basic idea to solve heterogeneous TR is to build a bridge between the source and target feature spaces, which could be accomplished by identifying a common feature subspace, learning a latent feature space for both the source and target collection, or mapping the source feature space of the target feature space. Wang et al. [80] have developed an algorithm named HCD Ranking (Heterogeneous Cross-domain Ranking) that learns a projection matrix, U, in order to map both the source and target features to a latent feature space ($\mathbf{x}' = U^\top \mathbf{x}$). HCD Ranking first represents the original feature spaces into an augmented feature space, that is, the value of a feature that does not exist in a domain/collection will be set to zeros. It then learns the projection matrix and model coefficients by:

$$\begin{aligned} \arg\min_{w^{so}, w^{ta}, U} & \sum_{(\mathbf{x}_{ij}, \mathbf{x}_{ik}) \in L^{ta}} [1 - S_{ijk} f(w^{so}, U^\top (\mathbf{x}_{ij} - \mathbf{x}_{ik}))] \\ & + C \sum_{(\mathbf{x}_{ij}, \mathbf{x}_{ik}) \in \mathcal{L}^{so}} [1 - S_{ijk} f(w^{ta}, U^\top (\mathbf{x}_{ij} - \mathbf{x}_{ik}))] \\ & + \lambda ||W||_{2,1}^2 \\ & \quad s.t.\ U^\top U = I \end{aligned} \tag{2.48}$$

where $S_{ijk} = \begin{cases} 1, r_{ij} > r_{ik} \\ -1, r_{ik} < r_{ik} \end{cases}$, C is the parameter that controls the imbalance between the source and target samples, $||W||_{2,1}^2$ is the regularization term that penalizes the complexity of the model, and λ is the parameter controlling the tradeoff between empirical loss and the penalty. The algorithm then solves Equation 2.48 in a two-stage learning manner. It first learns to construct the projection matrix with an equivalent form of Equation 2.48 and then learns the weights for the models.

The *RA-SVM* algorithm proposed in Geng et al. [20] also has two variants that can deal with heterogeneous TR settings. However, the assumption is slightly different from the previous case. *RA-SVM* solves the problem of transferring knowledge from a generic ranking model to a domain-specific ranking model; heterogeneous TR happens when there exist some domain-specific features in the target domain. The two variants of *RA-SVM*, *RA-SVM-MR*, and *RA-SVM-SR*, tackled the problem by rescaling the classification margin or the slack variables with the similarities of two documents in the domain-specific feature space. The algorithms were tested by transferring from a web search domain to an image search domain and showed significant improvements. Similar to *RA-SVM*, the algorithms can hardly be generalized to other L2R algorithms.

In Long et al. [82], the authors have investigated the heterogeneous TR problem when the source and target domain have some overlapped features. The authors developed a probabilistic model called PCDF that can learn the latent factors between the two feature spaces. More specifically, PCDF assumes that the common feature spaces of the source and target domain are generated from a distribution that is factored by a mapping function:

$$\mathbf{X}^{(tc)} \sim p(\mathbf{X}^{(tc)}|f(\mathbf{Z}^{(tc)}; P^{(c)}) \tag{2.49}$$

$$\mathbf{X}^{(sc)} \sim p(\mathbf{X}^{(sc)}|f(\mathbf{Z}^{(sc)}; P^{(c)}) \tag{2.50}$$

where $\mathbf{X}^{(tc)}$ and $\mathbf{X}^{(tc)}$ are the source and target training data (the variable) in the common feature space, respectively, $\mathbf{Z}^{(tc)}$ and $\mathbf{Z}^{(tc)}$ are the correlation matrix of the corresponding common features, $P^{(c)}$ is the shared parameter of both domain and the function $f(.)$ maps the distinct latent factors to the common features of the two domains. While for the domain-specific feature space the data of the two domains are modeled as the latent matrix factored by two different mapping functions:

$$\mathbf{X}^{(td)} \sim p(\mathbf{X}^{(td)}|f(\mathbf{Z}^{(td)}; P^{(td)}) \tag{2.51}$$

$$\mathbf{X}^{(sd)} \sim p(\mathbf{X}^{(sd)}|f(\mathbf{Z}^{(sd)}; P^{(sd)}) \tag{2.52}$$

the output space of the training data is modeled as preferences using a latent relevance score y:

$$\mathbf{R}^{(s)}_{ijk} \sim p(\mathbf{R}^{(s)}_{ijk}|r(y^s_{ij}, y^s_{ik})) \tag{2.53}$$

$$\mathbf{R}^{(t)}_{ijk} \sim p(\mathbf{R}^{(t)}_{ijk}|r(y^t_{ij}, y^t_{ik})) \tag{2.54}$$

and the latent relevance score is generated conditioned on the latent features of the source and target domain:

$$y^s_{ij} \sim p(y^s|h(Z^{(s)}; w)) \tag{2.55}$$

$$y^t_{ij} \sim p(y^t|h(Z^{(t)}; w)) \tag{2.56}$$

$$\tag{2.57}$$

where $Z^{(s)} = [Z^{(sd)}Z^{(sc)}]$ and $Z^{(t)} = [Z^{(td)}Z^{(tc)}]$. As a result, the PCDF algorithm derives a Bayesian network to model the relationship between the feature spaces, the preferences output, the latent matrix, and the corresponding parameters. The model parameters are then learned with a stochastic gradient descent algorithm that is equivalent to maximizing the likelihood. After the latent feature representation $Z^{(s)}$ and $Z^{(t)}$ are learned, a ranking function can be used to train the model in the latent feature space. The results tested on a large commercial web search engine demonstrated that PCDF can outperform many other existing solutions for heterogeneous transfer learning.

2.5.2 Unsupervised TR

Unlike supervised TR, under the unsupervised TR scenario, no relevance labels are provided in the target collection. As a result, the algorithms are not able to detect the relatedness of the joint distribution of the source and target domains, which poses additional challenges. As the difference in the mapping function cannot be modeled without the presence of any relevance labels in the target collection, most unsupervised TR can only tackle one aspect of the problem, which is the distribution change in the input feature space. However, this is still very useful in some scenarios when the source and target collections share the same mapping function, for example when a ranking algorithm must be updated after a period of time. In this section, present algorithms to tackle the unsupervised TR problem will be reviewed, including the instance weighting approach and weak supervision-based methods.

2.5.2.1 Instance Weighting

Instance weighting is one of the most widely used solutions for the covariate shift problem in transfer learning, and it has also been used to address the covariate shift problem in ranking problems. Covariate shift is the main problem that unsupervised TR aims to tackle.

As has been discussed in the section 2.2, the training data for L2R are used in different ways. As a result, the instance weighting for L2R can be conducted at different levels, i.e., document level, pair level, and query level. In Gao et al. [18] , the authors generated instance weights at different levels for L2R datasets. Although not pointed out, their methods are similar to classification-based density-ratio estimation; they built a classifier hyperplane between source and target documents and used a sigmoid function of the distance of a target document to the hyperplane at the document level. Since documents are independent of each other, the document-pair weights are the multiplication of the documents' weights. The query weights were generated by the average weights of document pairs in the query. They tested their instance weights with RankSVM and RankNet (two pairwise L2R algorithms[9]) on the six topic sets in LETOR3.0 and showed some significant improvements. Cai et al. [73] further improved the algorithm by classifying the queries directly. The algorithms were tested on a set of a small dataset and showed only limited improvements in ranking effectiveness.

An importance-weighted AdaRank approach (wAdaRank) was proposed by Ren et al. [72]. The authors used the Kullback-Leibler Importance Estimation Procedure (KLIEP) [63] to estimate document weights, which were then incorporated into the AdaRank algorithm. However, the algorithm was not tested under an unsupervised TR scenario. Instead, the authors tested the algorithm in a supervised learning environment. The density ratio was estimated according to the test set and was tested on the test set as well.

2.5.2.2 Weak Supervision

One difficulty of instance weighting for unsupervised TR is the necessity to characterize the data distribution of the L2R training data. However, most effective L2R algorithms use training data at the query level as well as for the evaluation of the effectiveness of ranking models is at the query level. Measuring the divergence of query distribution can be very difficult as queries are represented as lists of feature vectors. More details on the difficulty of implementing query-level instance weighting will be discussed in Chapter 4.

Instead of estimating the density weights for each training example in the source collection, an alternative approach is to generate imputed relevance labels to the query-document pairs in the target collection with the auxiliary from a source collection.

Weak supervision-based unsupervised TR solutions have not been well studied. Gao and Yang [76] have developed an algorithm that can use a weakly supervision algorithm called multi-view learning to generate imputed labels to enable ranking model adaptation. Multi-view learning [94] is a set of semi-supervised algorithms that can propagate labels to unlabeled instances in the training set by the consistency of label prediction with models trained with different feature sets. The AdaCoList algorithm developed by Gao

[9]AdaRank and LambdaMART are more effective [36].

and Yang [76] is a co-training algorithm that can automatically identify two views from the original feature set. This is done by minimizing the combined loss in the two views in the source collection while maximizing the ranking agreements of the two views in the target collection.

The weights of the two views were then used to predict the ranking of unpredicted target queries. The top k queries with the most confident ranking predicted will be added to the source collection in order to update the ranking function. The source queries are weighted by the NDCG score achieved with the current model so that the algorithm focuses more on source queries that are close to the target collection. The algorithm then iteratively updates the weights for the model and adds more consistent ranked queries to the source collection until no more target queries can be added. AdaCoList was tested on LETOR3.0 and Yahoo transferring settings, and the result suggests that AdaCoList can gain significant improvements with proper settings. However, the performance of the algorithm may vary with different parameter settings.

2.5.3 Other TR-Related Problems

Apart from the common TR scenario that the knowledge transfer is from one collection to another, there exist some other cases where the situation may change. For example, Goswami et al. [23] attempted to look at the case when there exist multiple source collections. The authors tackled the problem by selecting the most similar collection for each query by comparing the similarity of the td-idf distribution. The algorithm then uses the most-related collection to generate pseudo relevance labels for the documents belonging to the query and uses the labels for training. In the experiments, CLEF-3, and TR EC 3,4,5,6 were used as source collections and they tested the algorithms on TR EC 7,8, and WT10G as well as Gov2 collections. The effectiveness of the algorithm was compared with BM25, language model, and LGD [95] as well as a RankSVM model trained with source collection data, and it consistently outperformed these models significantly.

Others have also attempted to solve the multi-task learning problem for L2R. Multi-task learning is a technique that learns multiple models simultaneously for different related tasks. Most of the solutions (Bai et al. [86], Tang and Hall [90]) to multi-task L2R look to find commonalities in the latent feature space in order to share knowledge. For example, Bai et al. [86] assume that some “super features” are shared by different tasks. It uses a boosting algorithm to iteratively learn the super features and the coefficients. Chapelle et al. [15, 87] also developed an algorithm called multi-boost that can deal with multitask learning problems for boosting algorithms. It is demonstrated that multi-boost can be used for GBDT (gradient boosted decision tree) to solve the multi-task L2R problem.

TR can also be used to help active learning for L2R, for example, Cai et al. [75] tried to use source collection data to help select queries from target collection for judgment. It combines the techniques of transfer learning with active learning. The result showed that the proposed approach can bring a significant improvement over the normal active learning setting.

2.6 Summary

Although all the algorithms that have been discussed focused on knowledge transfer between different ranking tasks, the scenarios can greatly differ. The choice of TR algorithms for a particular scenario solely depends on the accessible resources for the task. Moreover, some algorithms may not be flexible enough to apply to a different L2R algorithm. In this thesis, we limit our focus to unsupervised TR where there are no relevance labels from another collection. Unsupervised TR is a problem that has not been well-studied and yet has many real applications, including dataset shifting overtime, cross-lingual transferring, etc.

Chapter 3

Dataset Shift in L2R

One fundamental issue that TR aims to tackle is the dataset shifting problem. The ranking model trained with a small number of observed training instances from the target collection or the model trained with data from the source collection cannot generalize to the unseen target data. In this section, the data generating process of L2R datasets, which is the underlying true distribution that is generating the data, will be discussed and it will be used to explain the causes and effects of different distribution changes within L2R datasets. Furthermore, the generalization of different L2R algorithms with respect to different types of dataset shift is investigated so that insight can be gained in order to better understand the correlation between dataset shift and performance changes.

3.1 Introduction

TR algorithms are designed to tackle the scenario where a reliable ranking model cannot be trained with existing training data and labels in the target collection and the ranking model trained within the source collection does not generalize well to the target collection. One fundamental question that needs to be answered before investigating any TR algorithms is when TR should be applied.

If the training data and labels in the target collection are sufficient to train an effective ranker, or the source ranker can be directly applied to the target collection without any performance degradation, TR should not be chosen in these two scenarios. Obviously determining that this is, in fact, the case may be a non-trivial problem if insufficient test data on the target collection is available. As a result, it is important to investigate the generalization of L2R algorithms with respect to the training size and changes in the data distribution.

The study of generalization of machine learning algorithms with respect to the training size is a general research topic, and some previous works [24, 25, 96] have also investigated the generalization ability of L2R algorithms. Most of those studies have investigated the generalization bound of L2R algorithms at either the document level or the query level. Macdonald et al. [97] carried out empirical analysis of the document pooling methods for L2R collections and its impact on training. The result showed that the impact of number of documents for each query will not significantly affect the performance unless it is too small. In Yilmaz and Robertson [98], the authors have also found that training L2R with shallow document depth but with more queries is better than training with more documents per query and fewer queries. Later in Chen et al. [99], a generalization bound has been derived to take account of both the document and query sample size. They concluded that both document count and query count have an impact on the generalization bound, while a cost-effective approach to increase generalization is to increase the query count over document count.

The second question we want to answer is the generalization of L2R algorithms across different collections. To the best of our knowledge, there has been no previous work investigating this problem. The generalization issues of learning algorithms when training and testing in different collections are usually referred to as dataset shift [8] problems in machine learning. In this chapter, we investigate the cross-collection generalization abilities of different types of L2R algorithms to provide us better insight into differences that exist among L2R collections and also the impact of those variations on the generalization of L2R algorithms. By understanding the cause of variations in the dataset, one can determine particular change in the data distribution and should be able to identify the appropriate solution to the problem.

In this chapter, we first discuss the data generating process of L2R datasets as it is different from conventional machine learning datasets from multiple perspectives. In the following sections, we characterize different types of distribution shifts in L2R datasets, which allow us to attribute the distribution shifts to a particular difference in the dataset generation process. Finally, a set of experiments were performed to test the generalization abilities of different L2R algorithms across different test collections based on particular changes in the collection. We found that: 1) different parameter changes in the dataset can cause a generalization gap for L2R algorithms; 2) the impact on the generalization abilities of different L2R algorithms with respect to a particular type of change in the dataset may be different; and 3) the cross-collection generalization ability of an L2R algorithm is also determined by its in-collection generalization ability.

3.2 Data Generating Process

To quantify the data distribution of L2R datasets, we formalize the data generation process for L2R datasets. An L2R collection is comprised of a set of queries, and a set of (retrieved) documents for each query together with relevance labels. We can model the queries and documents in an L2R collection as random samples drawn from the query space $\mathcal{Q}$ and the document space $\mathcal{D}$ respectively according to the following probabilities:

$$q_i \sim P_q \quad i \in \{1, \ldots, M\} \tag{3.1}$$

$$d_j \sim P_d \quad j \in \{1, \ldots, N\} \tag{3.2}$$

The relevance of a document to a query is controlled by the relevance probability:

$$r_{ij}|q_i, d_j \sim P_{r|q,d} \quad i \in \{1, \ldots, M\}, j \in \{1, \ldots, N\} \tag{3.3}$$

The relevance probability conditioned on the query and document is determined by specific domains, and it may shift under different contexts. It determines the (marginal) probability that a user having submitted query q would consider documents relevant (at a certain level on a relevance scale). Thus the relevance itself only reflects the degree that a document is related to the information need. A relevance label $r_{ij} = y$, meanwhile, is an ordinal number that quantifies the degree of relevance, either being binary, multi-graded or even real-valued, and is usually determined by the relevance judgment process. We assume that the relevance label mapping is governed by a probability $P_{y|r}$ [1]. Each

[1]For the simplicity consideration, in other chapters, we will use r_{ij} to denote the relevance labels, and the probability $P_r|q, d$ is determined by both the domain and the judgement process.

query-document pair is represented by a feature vector $\mathbf{x}_{ij}$ mapping from the pair to a d dimensional vector ($\mathcal{Q} \times \mathcal{D} \mapsto \mathbb{R}^d$). To simplify, we assume that the feature vector is controlled by a mapping function ϕ, with parameters λ_ϕ:

$$\mathbf{x}_{ij} = \phi(q_i, d_j; \lambda_\phi) \tag{3.4}$$

If all the relevance labels are present in the L2R collection, the distribution over the feature vector and the relevance labels would be:

$$P(\mathbf{x}_{ij}, y_{ij}) = P_q(q_i)P_d(d_j)P_{\mathbf{x}|q,d,\phi}(\mathbf{x}_{ij}|\phi(q_i, d_j; \lambda_\phi))P_{r|d,q}(r_{ij}|q_i, d_j)P_{y|r}(r_{ij}) \tag{3.5}$$

where $P_q(q_i)$ is the probability of a query q_i given the query distribution P_q, $P_d(d_j)$ is the probability of a document d_j, given the document distribution. $P_{\mathbf{x}|q,d,\phi}(\mathbf{x}_{ij}|\phi(q_i, d_j; \lambda_\phi))$ denotes probability of a feature vector $\mathbf{x}$, given the q_i and d_j, and this probability is governed by a conditional distribution of $P_{\mathbf{x}|q,d,\phi}$, where ϕ is a mapping function. $P_{r|d,q}(r_{ij}|q_i, d_j)$ is the probability of relevance, given q_i and d_j. Relevance labels can be annotated in multiple ways, so the relevance label y is drawn from a conditional distribution of $P_{y|r}(r_{ij})$.

However, in practice, it is not feasible to assess the relevance of every document in the corpus for every query. Moreover, if most of the documents in a corpus are irrelevant to a search topic, it is meaningless to spend the effort judging the relevance for large volumes of irrelevant documents. Lastly, the computational burden for an L2R algorithm can be very heavy in dealing with a collection with full sets of documents judged. Instead, only a subset of the documents corresponding to a query is used for training, and it is obtained by a pooling method. A typical pooling method uses a conventional IR retrieval model, e.g., BM25, to return a ranked list of documents for each query, and then preserve the top-k documents in the retrieved ranked list. However, the base retrieval models and pooling depth k may change in different collections. To simplify, we use o_{ij} to denote the whether d_j is observed in q_i with a pooling method. We assume o_{ij} is drawn from a distribution that is governed by the distribution of $P_{o|d,q}$:

$$o_{ij} \sim P_{o|d,q}(q_i, d_j) \tag{3.6}$$

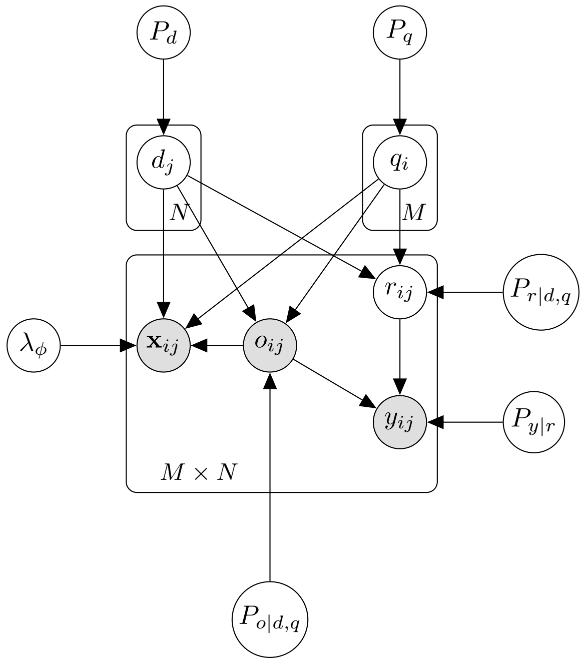


FIGURE 3.1: Model of training/test corpus generation for learning to rank: shaded (/unshaded) round nodes represent observed (/latent) variables

As a result, the data in the collection is the observed feature vectors and their corresponding relevance labels and the data forms a joint probability of $P(\mathbf{x}_{ij}, r_{ij}|o_{ij} = 1)$. The joint probability can be rewritten as:

$$\begin{aligned}
& P(\mathbf{x}_{ij}, y_{ij}|o_{ij} = 1) \\
= \; & \frac{P(o_{ij} = 1|\mathbf{x}_{ij}, y_{ij}) \cdot P(\mathbf{x}_{ij}, y_{ij})}{P(o_{ij} = 1)} \\
= \; & \frac{P_{o|q_i,d_j}(o_{ij} = 1|q_i, d_j) \cdot P(\mathbf{x}_{ij}, y_{ij})}{P(o_{ij} = 1)} \\
\propto \; & P_q(q_i)P_d(d_j)P_{\mathbf{x}|q,d,\phi}(\mathbf{x}_{ij}|\phi(q_i, d_j; \lambda_\phi)) \qquad (3.7) \\
& \times P_{r|d,q}(r_{ij}|q_i, d_j)P_{y|r}(r_{ij})P_{o|q_i,d_j}(o_{ij} = 1|q_i, d_j)
\end{aligned}$$

Figure 3.1 summarizes the data generating process of L2R collection as a Bayesian network.A Bayesian network is a graphical model for the joint probability for a set of variables. Each node in the graphic denotes a variable that is observed or hidden in the dataset. The arrows denote the relationship between different nodes, i.e., different variables pointing to the same variables are the ones that control the distribution of the variable. The plates in the graphics denote the size of a set of variables. The shaded nodes are variables that were observed in the collection, the transparent nodes denote the latent variables. The directional arrows show the parameters that control the generating of a variable. It is clear from Figure 3.1 that the data distribution in an L2R collection is controlled by the following parameters:

1. The document distribution P_d
2. The query distribution P_q
3. The conditional probability of relevance, given a query and document, $P_{r|d,q}$
4. The relevance judgment process, controlled by $P_{y|r}$, the probability of the label given its true relevance.
5. The conditional probability of relevance, given a pair of query and document, $P_{r|d,q}$
6. The condition of probability of a document being observed (and subsequently labeled) for a query after pooling, $P_{o|d,q}$
7. The parameter λ_ϕ controls the feature mapping function $\phi(q, d; \lambda_\phi)$

In the next sections, the impact of these factors will be discussed in order to quantify the dataset shifts in L2R collections.

3.3 Generalization of L2R Models

Typically, an L2R algorithm looks to minimize the gap between the optimal ranking and the predicted ranking of all the queries in the population:

$$\mathcal{R}(h) = \mathbb{E}_{(X,R)\sim P_{(X,R)}}[\ell(R, h(X))] = \int\int p(X, R)\ell(R, h(X))dXdR \tag{3.8}$$

where (X, R) is a set of retrieved document feature vectors paired with their relevance labels, $P_{(X,R)}$ is the distribution of (X,R), h is the hypothesized optimal ranking function.

However, L2R models are usually trained using observed sampled data with corresponding relevance labels, and the expected risk is approximated by the empirical risk:

$$\tilde{\mathcal{R}}_{emp}(h) = \frac{1}{N_X} \sum_{(X_i,R_i)\in\mathcal{L}_{emp}} [\ell(R_i, h(X_i))] \tag{3.9}$$

Ideally, the hypothesis should only be accepted as a solution when the gap between the two risks (generalization gap) is small enough:

$$P(\sup_{h\in\mathcal{H}} |\mathcal{R}(h) - \mathcal{R}_{emp}(h)| > \epsilon) < \delta \tag{3.10}$$

Under the TR setting, the hypothesis is trained with the data drawn from the source distribution, while it will be applied to data from a different data distribution. As a result, the source hypothesis is trained by minimizing the following risk:

$$\tilde{\mathcal{R}}^{so}(h) = \frac{1}{N_{so}} \sum_{(X_i,R_i)\sim P^{so}(X,R)} [\ell(R_i, h(X_i))] \tag{3.11}$$

However, the expected model is that which can minimize the expected risk over the target distribution:

$$\mathcal{R}^{ta}(h) = \mathbb{E}_{(X,R)\sim P^{ta}(X,R)}[\ell(R, h(X))] \tag{3.12}$$

Issues arise when the observed data is generated from a different distribution, causing the generalization gap, $|\mathcal{R}^{ta} - \tilde{\mathcal{R}}^{so}|$, to be even larger. The ranking function trained on the source collection will not generalise to target collection, and the performance will be degraded.

3.4 Characterizing Dataset Shifts in L2R Datasets

Quantifying if the data distribution difference between source and target is a critical step in studying transfer learning algorithms. However, the training data in L2R is controlled by many factors, as has been shown in the data generation process. In this section, we attribute different types of distribution changes to specific changes in the data generating process.

According to Quionero-Candela et al. [8], the joint distribution change of the input and output space can be a result of the following shifts:

- **Covariate Shift**

- **Prior Probability Shift**
- **Domain Shift**

3.4.1 Covariate Shift

Covariate shift is the scenario where the distribution of data instances over the input feature space changes, $p^{so}(\mathbf{x}) \neq p^{ta}(\mathbf{x})$, while other probabilities remain the same. It is obvious that covariate shift has an impact on the generative learning models as they model the joint distribution $p(\mathbf{x}, y)$. However, most L2R algorithms are discriminant learning models that directly learn the conditional distribution $p(y|\mathbf{x})$. From the probability perspective, the distribution of $p(\mathbf{x})$ has no impact on the conditional distribution $p(y|\mathbf{x})$, which means $p(y|\mathbf{x}^{so})$ can be used to predict $\mathbf{x}^{so}$ accurately. However, as the models are trained using empirical data through risk minimization, the model may not be able to generate the global conditional probability. In other words, the trained model may be locally minimized and cannot generalize well to another collection. To illustrate this, we provide a covariate shift example in Figure 3.2. Assume the conditional distribution of the label y, given x, $p(y|\mathbf{x})$ is controlled by the dashed line in the figure. If all the data is present in the training set, we could train a linear regression model to approximate the probability, which is denoted as the solid black line in the figure. However, in many cases, the input features may not distribute evenly across space. This is usually caused by the sample selection bias for specific domains. The feature distribution of a specific domain may be biased compared with the population of the universe. For example, as illustrated in the figure, the observed instances in the source collection is skewed towards lower ranges of the values. Using an empirical risk minimization-based solution, the source model (the red line) will be trained. However, the trained source model does not generalize well to the target collection instances, the black triangles in the figure which are concentrated at higher values of $\mathbf{x}$..

Covariate shift can also happen in L2R datasets, and the shift may occur at different levels. For pointwise L2R algorithms, the input feature space is the feature vectors extracted from query-document pairs. According to the data generating model that we discussed in the last section, the distribution of the input feature space is controlled by the document distribution, query distribution, and the pooling method. The factors are still valid when it comes to pairwise algorithms. For listwise algorithms, the sets

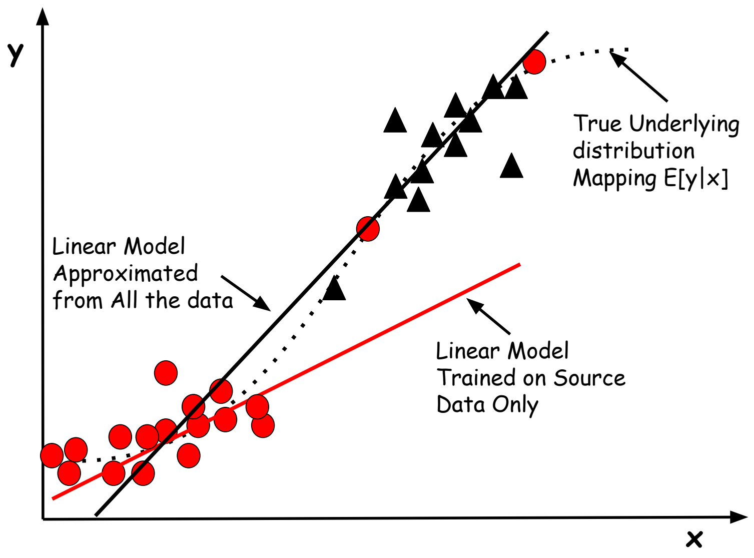

y
x
True Underlying distribution Mapping E[y|x]
Linear Model Approximated from All the data
Linear Model Trained on Source Data Only

FIGURE 3.2: The red dots are instances observed in the source collection, the black triangle points are the instances in the target collection. All the instances are governed by the dashed line, which is the underlying distribution. The black line is the model trained when all the instances were observed, while the red line is the model when only the source collection data was observed

of feature vectors are deterministic when the distribution of the feature vectors are fixed. As a result, the change in document distribution, query distribution, and pooling methods are the three main factors that contribute to covariate shift.

3.4.1.1 Document Distribution Shift

Changes in document distribution are one of the most common causes of covariate shift in L2R datasets. Notice that such changes should not introduce the domain shift, which would cause the conditional probability change. For example, if the document corpus was changed from the IT domain to the medical domain, the conditional probability of relevance as well as the query distribution could also be affected. However, whether the change of document corpus will impact on the conditional probability is not easy to detect. For example, it has been shown by some other studies [4, 5, 100] that the choice of document corpus will have a significant impact on the effectiveness of retrieval systems.

In some scenarios, we could assume that the global definition of relevance does not change over different collections, in which case the feature distribution change will be the main cause of the distribution change. For example, in some collections, the document corpus may be updated dramatically after a period of time. Unfortunately, there is no public test collection for us to validate how the document shifting over time could cause a covariate shift.

The other situation when the document distribution shifts is when the document type changes. For example, the characteristics of HTML files and pdf files can be very different, which may lead to a dramatic difference in the feature distribution of documents. In order to observe the covariate shift when the document type changes, we separate the MQ2007 dataset from the LETOR4.0 collection into a collection with only HTML files and a collection with only pdf files. We then randomly sampled 1,000 query-document pairs from both collections and compared two features present in both collections, BM25 of the whole document and the document length of the whole documents.

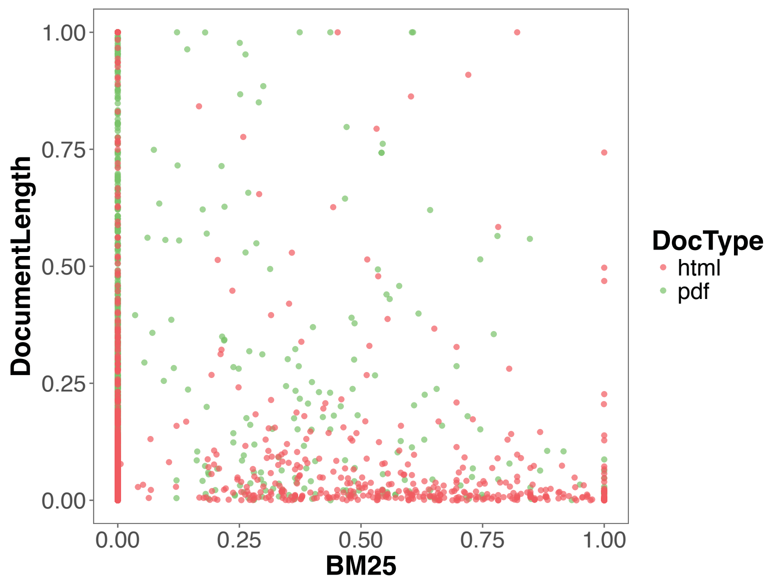
The scatter plot and the density plots in Figure 3.3 show that the features of the collections distribute differently in the feature space. Pdf files are more likely to have lower BM25 scores, while the document length tends to be less skewed. When it comes to the high-dimensional feature space, the joint distribution of the features would be more sensitive to the changes in the document space.

3.4.1.2 Query Distribution Shift

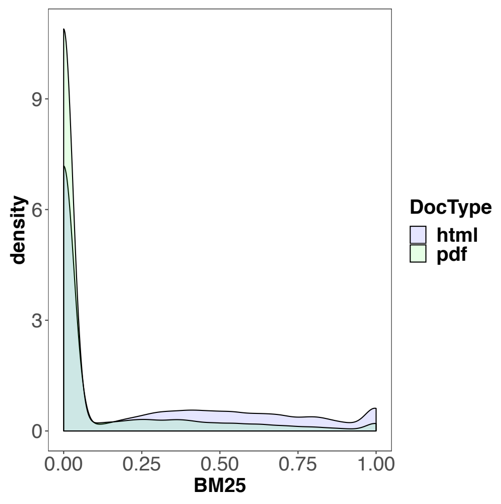
Changes in the distribution of queries are also one of the causes of covariate shift in L2R datasets. According to the data generating process, the presence/absence of certain feature vectors is also determined by the queries. Moreover, most of the state-of the–art algorithms train the ranking function using query-level loss function and the performance is also measured at the query level. As the distribution of queries shifts, it will very likely cause covariate shift in the L2R training set.

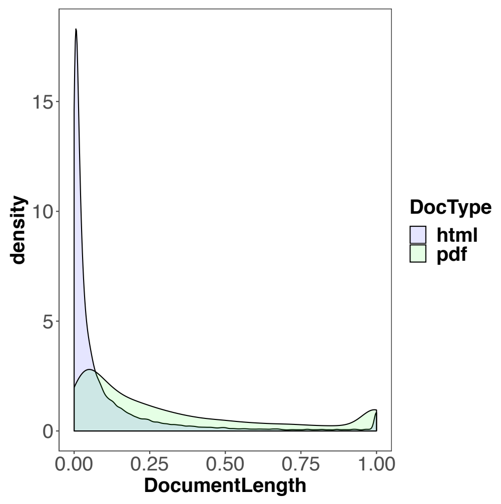
The changes of query distribution may also introduce domain shift, in which case the criteria of relevance may shift. In covariate shift, we are only concerned with the case when the query probability shifts, while the information needs expressed by the queries remains the same.

According to Weber and Castillo [101], users' search behaviors, including queries issued, are correlated with their demographics. As a result, query distribution change may happen when an IR system is applied to a different market, for example, from the US market to the Australian market.



(a) Scatter plot for the feature distribution of html and pdf files in LETOR4.0





(b) BM25 distribution of html and pdf files in LETOR4.0

(c) Document length distribution of html and pdf files in LETOR4.0

FIGURE 3.3: Feature distribution of html and pdf files in LETOR4.0

3.4.1.3 Pooling Method Change

The pooling methods can also affect the input feature distribution. However, as the pooling method is usually a controllable factor at the retrieval stage, it is less of a concern for covariate shift. The pooling strategy and depth will only impact the probability of a document being observed for a query. Some [102] attempts have been made to use counterfactual learning methods to solve the observation bias issues for training ranking functions with click data. Others [103] have investigated the relationship between the robustness of L2R algorithms and the pooling methods used.

In many cases, the covariate shift is caused by a combination of two or three changes in the aforementioned factors. To mitigate the problem, one will need to tackle the shift from different perspectives respectively.

Ideally we would like to check that covariate shift has occurred, and measure and visualise shift between the datasets. But doing so for learning-to-rank algorithms is not possible because of the structure of the query space, which consists of a set of feature vectors (one for each candidate document) rather than a single feature vector, as would be the case for conventional machine learning algorithms. Also, density estimates in high dimensions (where the number of dimensions is comparable to the number of data points) are inherently unreliable. For this reason, most previous studies of covariate shift in Machine Learning have made use of synthesised datasets (sometimes exclusively). We were unable to generate synthetic datasets because the data distribution of learning to rank datasets are controlled by too many factors.

3.4.2 The Effect of Covariate Shift on L2R Algorithms

Although covariate shift is shown to affect the generalization ability of machine learning algorithm, how it will affect L2R algorithms is not clear. As discussed above, covariate shift may occur in different ways, how different types of covariate shifts will impact the training of different L2R algorithms needs further investigation.

For pointwise algorithms, the ranking functions are trained and optimized at the document level, which means that the learning algorithms minimize the expected loss over all of the training instances. As a result, any changes in the data will affect the generalization of L2R algorithms significantly.

For pairwise algorithms, the input features are pairs of query-document feature vectors, that are determined by the distribution of query-document feature vectors. As a result, changes in the document distribution, query distribution, as well as pooling methods used will all affect the generalization performance of pairwise algorithms.

For listwise algorithms, the algorithms directly train ranking functions via optimizing at the query level. The algorithms are likely more sensitive and affected by the distribution in the query distribution. As a consequence, the change in the query set would likely have more influence on the generalization performance of listwise L2R algorithms.

3.4.3 Prior Probability Shift

A *Prior probability shift* is the case when the prior probability of the labels shifts, $(p^{so}(y) \neq p^{ta}(y)$. A common scenario for prior probability shift is that distribution of the labels, e.g., the density of positive and negative labels, are different in the source and target collection.

For a generative learning model, the conditional probability is obtained by:

$$p(y|x) = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \tag{3.13}$$

Even if the distribution of $\mathbf{x}$, $p(\mathbf{x})$ remains the same, the change in the class distribution $p(y)$ may also result in poor prediction accuracy for a model trained on data with a different label distribution.

For different L2R collections, the labeling strategy could be different, for example, some collections may use the binary definition for relevance while others may use graded relevance. Even with exactly the same strategy for labeling, the distribution of the relevance labels may also be different due to the document or query distribution. For example, for navigational queries, there might be just one or two relevance documents while for other ad-hoc queries the number of relevance document might be much larger. However, most advanced learning-to-rank algorithms are trained to minimize pairwise preference error. As a result, the impact of the differences in relevance labels on the training of those pairwise and listwise algorithms is smaller than it is on the pointwise algorithms.

3.4.4 Domain Shift

Domain Shift is the other cause for dataset shift. Under domain shift, the main change in the dataset is in the mapping function $p(y|\mathbf{x})$, while the covariate distribution remains the same, $p^{so}(\mathbf{x}) = p^{ta}(\mathbf{x}), p^{so}(y|\mathbf{x}) \neq p^{ta}(y|\mathbf{x})$.

As mentioned before, discriminant learning algorithms directly model the prediction function. However, in many cases, the prediction function may change in different collections, which means that the definition of relevance may drift. As a result, domain shift is also called “concept drift” in some literature [104].

Some document collections may be domain specific and thus when a trained ranking model is applied to a different collection, the learned mapping function may not hold. For example, the performance of a ranking function trained on a medical domain will be degraded when applied to an IT collection as the definition of relevance may be different across the collections.

Domain shift is not easy to detect and manage when there are not any relevance labels in the target collection. As a result, the solutions for domain shift usually require a small amount of target relevance labels.

In reality, dataset shift may be caused by multiple factors. For example, when a search engine is applied to a different language market, the document corpus, query distribution, and even the mapping function itself may shift. In such cases, one needs to use available resources in both the source and target collection to tackle different types of changes separately.

3.5 Empirical Results

In order to investigate the effect of different types of dataset shift on the generalization ability of learning-to-rank algorithms, three representative learning-to-rank algorithms were tested on different dataset shift cases, using synthetic data generated from public L2R datasets.

3.5.1 Dataset

Some of the existing collections were split in order to simulate certain types of dataset shift. In this section, we create different dataset shift scenarios to test the correlation between the generalization performance of L2R algorithms and the presence of particular types of shifts in the dataset.

3.5.1.1 Document Corpus Change

To test and analyze the impact of document corpus change on the generalization performance of L2R algorithms, the MSLR collection was split into two collections according to document length. More specifically, we select documents whose length lies above the 25th percentile and below the 75th percentile into two collections: *MSLR-LongDoc* and *MSLR-ShortDoc*. As a result, *MSLR-LongDoc* and *MSLR-ShortDoc* shares the same query set. In order to reduce the impact of pooling, we only retain the top 20 documents for every query in both collections.

3.5.1.2 Query Set Change

Query set shift is one of the most common scenarios when covariate shift happens. The distribution of the queries determines the set of documents being retrieved. As a result, the concept of query distribution cannot leave out the retrieved document set. Whether the characteristics of the queries themselves, such as the stream length and frequency, affect the distribution of input feature space requires further investigation. In the LETOR4.0 dataset, there are two query sets from the Million Query Track corresponding to two different years. We will use the query sets from the two years to investigate the impact of query distribution change. Moreover, in the MQ2008 dataset from LETOR4.0, there are four categories of queries [58]: *short-govslant*, *long-govslant*, *short-heavy* and *long-govheavy*. According to Allan et al. [58], short queries are those queries that contain less than six words, while long queries are those with more than six words. The "heavy" queries are those queries that have greater than three clicks while "slant" queries are those that have less than three clicks. We split the MQ2008 set using two different collections, *long queries* and *short queries*, to see if it will cause any covariate shifts for the L2R algorithms.

3.5.1.3 Domain Shift

Domain shift happens when the source and target collections come from different domains, and as a consequence, the mapping function from the feature space to relevance labels changes. In this experiment, we use the two collections built from the Gov and OHSUMED corpus in LETOR3.0 to test the impact of domain shift on L2R. The Gov collection contains six queries and the document corpus is the gov corpus, which is crawled from the "gov" domain, while OSHUMED is based on a corpus of medical publications [37].

3.5.2 Algorithms

Four algorithms are tested to make the comparison, a pointwise algorithm based on multiple additive regression tree (MART) [105], a pairwise algorithm (RankNet [42]) and two listwise algorithms (ListNet [40] and LambdaMART [53]). All the algorithms are implemented using the ranklib library (v2.1)[2].

3.5.3 Experiment Set up

The experiments are performed following a five-fold cross-validation strategy, by which the original data is randomly split into five folds. All the models are trained on four out of five folds and tested on the remaining fold. Each dataset will have five models and the results presented are averaged over the five folds. When comparing the performance of a ranker from another collection, we use all the data from the collection for training and test on the entire target collection. As a result, we compare the performance of a source ranker with the cross-validated performance on the same collection.

The performance of different models on various collections is measured using Normalized Discounted Cumulative Gain (NDCG) [106] with the standard rank-plus-one discount function and exponential gain [42]. We measure NDCG at rank cutoff 10. We conducted two-tailed t-tests between results with significant level α set to 0.05. To quantify the changes in performance, we also report the effect size of the differences.

3.6 Results and Discussion

The results of the generalization test are presented in this section.

3.6.1 How does document corpus change affect generalization of L2R algorithms?

The results of cross-collection tests with different learning algorithms when the document corpus changes occur is presented in Table 3.1. Each row in Table 3.1 corresponds to a group of experiments, where the source and target collections are specified under

[2]`https://sourceforge.net/p/lemur/wiki/RankLib/`

Table 3.1: Results on different algorithms when the document corpus changes: each row is a group of experiments; the first column under the "NDCG@10" corresponds to the cross-collection training, while the second column is the in-collection training. Italic fonts denote the significant decrease in performance, the bold fonts denote the significant increase in the performance compared with the target model. The effect sizes are shown in the last column.

Algorithm	Dataset		NDCG@10		Effect Size
	Source	Target	Source Model	Target Model	
BM25	MSLR-LongDoc	MSLR-ShortDoc	0.4745	0.4745	
	MSLR-ShortDoc	MSLR-LongDoc	0.5236	0.5236	
MART	MSLR-LongDoc	MSLR-ShortDoc	*0.5400*	0.6620	-0.534
	MSLR-ShortDoc	MSLR-LongDoc	*0.5738*	0.5903	-0.074
RankNet	MSLR-LongDoc	MSLR-ShortDoc	*0.4385*	0.4419	-0.029
	MSLR-ShortDoc	MSLR-LongDoc	*0.4526*	0.4561	-0.039
ListNet	MSLR-LongDoc	MSLR-ShortDoc	0.4407	0.4411	
	MSLR-ShortDoc	MSLR-LongDoc	**0.4540**	0.4537	+0.020
LambdaMART	MSLR-LongDoc	MSLR-ShortDoc	*0.5682*	0.6609	-0.435
	MSLR-ShortDoc	MSLR-LongDoc	*0.5667*	0.6231	-0.282

the "Dataset" column. The column "Target Model" under the "NDCG@10" is the performance (NDCG@10) when the algorithm is trained and tested on the same collection while the previous column (Source Model) is the performance for the cross-collection case, where the algorithm is trained with the source dataset and tested on the target dataset. The italic fonts in the table denote a significant performance decrease as compared with a model trained on the same collection (target model), the bold texts indicate the cases when the performance is increased. The effective size, measured using the Cohen's d [107], of the change is also reported in the last column. Given two groups of variables X_1 and X_2, Cohen's d measures the effect size of the difference by:

$$d = \frac{mean(X_1) - mean(X_2)}{SD_{pooled}} \tag{3.14}$$

where SD_{pooled} is the pooled standard derivations computed as:

$$SD_{pooled} = \sqrt{\frac{SD^2_{X_1} + SD^2_{X_2}}{2}} \tag{3.15}$$

MART is a multiple additive regression tree model (MART) that trains an ensemble of regression trees to fit the relevance labels. Somewhat unexpectedly, the pointwise algorithm performed acceptably well in the MSLR collection. One reason is the size of training data in MSLR is very large while the MART model can fit the training data well. Similar pattern has been observed in [108]. However, the MART model is generally more sensitive to the document corpus change, where we observe 0.534 and 0.074 of performance decrease in effect size when ranking using a model trained with a different document corpus (the first four rows in Table 3.1). As the input features for pointwise algorithms are the query document feature vectors, a small change in the document corpus could impact the generalization of the algorithm.

RankNet [42] is a pairwise L2R algorithm that aims to optimize the ranking function by minimizing the loss between the pairwise probability inferred by the relevance labels and the probability predicted using the model scores. As a result, the impact of document corpus change is not direct, and the relative difference in pairs of documents is more important. When the algorithm was tested based on a cross-corpus test on the long and short document corpus in MSLR, it showed less decrease in performance with respect to MART.

The listwise algorithm, ListNet, optimizes its ranking function by minimizing loss in terms of a predicted permutation probability. As a result, the distribution of the lists of documents is more critical to ListNet algorithms, whereas the distribution of documents has a lesser impact. No significant performance decline was observed when the document corpus was changed.

It is noticeable that, although the effectiveness of MART and LambdaMART has largest degradation when applied to the target collection, the effectiveness of the two models are still the highest compared to other algorithms in such scenarios.

The document corpus change has a big impact on the generalization ability of the LambdaMART algorithm as can be seen from the last set of results in Table 3.1. Although LambdaMART was listed as a listwise algorithm in some literature, it optimizes its ranking function by minimizing a pairwise loss penalized by the impact on a listwise metric score, for example, the difference in NDCG@10 when swapping the pair of documents in the pair. As a result, LambdaMART is affected by the document corpuss has a larger impact.

Table 3.2: Results on different algorithms when the query set changes on LETOR4.0: each row is a group of experiments; the first column under the "NDCG@10" corresponds to the cross-collection training, while the second column is the in-collection training. Italic fonts denote the significant decrease in performance, bold fonts denote the significant increase in the performance compared with the target model. The effect sizes are shown in the last column.

Algorithm	Dataset		NDCG@10		Effect Size
	Source	Target	Source Model	Target Model	
BM25	MQ2007	MQ2008	0.3981	0.3981	
	MQ2008	MQ2007	0.2986	0.2986	
MART	MQ2007	MQ2008	*0.6103*	0.6765	-0.290
	MQ2008	MQ2007	*0.4707*	0.4907	-0.095
RankNet	MQ2007	MQ2008	*0.6024*	0.6741	-0.315
	MQ2008	MQ2007	***0.4548***	0.3852	+0.277
ListNet	MQ2007	MQ2008	**0.6912**	0.6766	+0.123
	MQ2008	MQ2007	**0.4636**	0.4911	-0.219
LambdaMART	MQ2007	MQ2008	0.6908	0.6761	
	MQ2008	MQ2007	*0.4850*	0.5173	-0.149

3.6.2 How does query set change affect generalization of L2R algorithms?

Table 3.2 displays the cross-collection test results for different algorithms on the two years of LETOR4.0: MQ2007 and MQ2008. MQ2007 and MQ2008 are two datasets released in 2007 and 2008 respectively. While the two datasets share the same document corpus, the query sets are different. However, the query types of MQ2007 and MQ2008 are similar, so there exists no domain shift in such scenario, while the distribution of 'queries' may be different. Notice that the 'query' here is not referring to the characteristics of queries, e.g., query length or population of queries, which will also be demonstrated.

When the query distribution changes, the distribution of the corresponding documents will be affected. As a result, the performance decrease of MART is seen.

Similarly, the pairwise L2R algorithm RankNet is also affected and we saw a significant performance decline when the model trained on MQ2007 is applied on MQ2008. However, when the model is trained on the MQ2008 dataset, it generalizes well to the MQ2007 dataset, and the performance is significantly better than the target model. RankNet seems to perform well on the MQ2008 dataset, while poorly on the MQ2007 dataset. A possible explanation is that although MQ2007 has more queries and deeper

Table 3.3: Results on different algorithms when the query set changes on long and short queries in MQ2008: each row is a group of experiments; the first column under the "NDCG@10" corresponds to the cross-collection training, while the second column is the in-collection training. Italic fonts denote the significant decrease in performance, bold fonts denote the significant increase in performance compared with the target model.

Algorithm	Dataset		NDCG@10		Effect Size
	Source	Target	Source Model	Target Model	
BM25	MQ2008-Long	MQ2008-Short	0.4096	0.4096	
	MQ2008-Short	MQ2008-Long	0.3864	0.3864	
MART	MQ2008-Long	MQ2008-Short	0.6659	0.6756	
	MQ2008-Short	MQ2008-Long	0.6464	0.6450	
RankNet	MQ2008-Long	MQ2008-Short	0.6852	0.6838	
	MQ2008-Short	MQ2008-Long	*0.6495*	0.6603	-0.151
ListNet	MQ2008-Long	MQ2008-Short	0.6913	0.6581	
	MQ2008-Short	MQ2008-Long	*0.6372*	0.6703	-0.250
LambdaMART	MQ2008-Long	MQ2008-Short	0.7007	0.6806	
	MQ2008-Short	MQ2008-Long	0.6644	0.6723	

pooling depth, the pooling method used in MQ2007 makes it harder for the learning algorithm to generalize well.

When a listwise algorithm, ListNet, is applied, the model trained on MQ2007 can outperform the target model on MQ2008 since MQ2007 contains a wider range of queries. As a result, the query distribution for MQ2007 is less biased than MQ2008, and when the model trained on MQ2008 is applied on MQ2007, we see a performance decrease.

LambdaMART is affected by both the pairwise distribution as well as the query distribution, and as a result, its performance only decreases when the MQ2008 target model is applied to MQ2007. However, the effectiveness of the model is still the best among all the other algorithms although the algorithm is applied to a different query set.

Table 3.3 shows the results when different algorithms are trained and tested on the long and short queries in the MQ2008 dataset from LETOR4.0. The change in the query length has no significant impact on the pointwise L2R algorithm, MART, since the document distribution is not affected. Significant performance drops were observed when using the RankNet and ListNet models for MQ2008-Long on the short query set MQ2008-Short. LambdaMART showed the highest performance on both the source and target models in both transfer directions. The changes of this query characteristic may not affect the overall distribution of the query-document feature space. However, the inherent document feature distribution within queries may change.

In a summary, the characteristic distribution of the query strings may mot have a direct impact on covariate shift, whereas the distribution of different types of queries may have a significant impact on data distribution of L2R collections.

3.6.3 How does domain change affect generalization of L2R algorithms?

Table 3.4 shows the results when the two datasets come from different domains. With the pointwise algorithm, MART, a significant decrease occurs when the model trained with the Gov collection is used for the OHSUMED collection, where the effect size is 1.185. However, the MART model trained on Gov can outperform the OHSUMED model on the OHSUMED collection. For the pairwise algorithm, the impact of domain shift is less sensitive. The other observation is that the listwise algorithm, ListNet, is ineffective for the Gov collection. The reason was that there exist six query sets with different types of information needs, which causes the poor generalization of listwise algorithms which looks to optimize the query-level ranking performance. With the LambdaMART algorithm, we also saw a significant performance drop when using the OHSUMED model for the Gov collection, while no significant difference is observed when the Gov model is used for the OHSUMED model.

3.6.4 Discussion

According to the empirical results that we discussed in the last sections, the shift of document set, query set, or domain may degrade the effectiveness of an L2R-trained ranking function, however, the impact of different changes on different types of L2R algorithms is different. Pointwise algorithms appear to be more sensitive to changes in distributions than other learning algorithms. For the pairwise algorithm, the impact on the document set changes appear smaller, while it showed some performance variation when the query set changed. Listwise algorithms appear most sensitive to any changes in the query set. Moreover, when the query characteristics were used as distinguishing features to divide the collections, the change in the query distribution caused the change the distribution of lists of retrieved documents, which appears to be the main cause of

Table 3.4: Results on different algorithms on the Gov and OHSUMED collection in LETOR3.0: each row is a group of experiments; the first column under the "NDCG@10 corresponds to the cross-collection training, while the second column is the in-collection training. Italic fonts denote the significant decrease in performance, bold fonts denote the significant increase in performance compared with the target model. The effect sizes are shown in the last column.

Algorithm	Dataset		NDCG@10		Effect Size
	Source	Target	Source Model	Target Model	
BM25	OHSUMED	Gov	0.5114	0.5114	
	Gov	OHSUMED	0.4000	0.4000	
MART	OHSUMED	Gov	*0.0336*	0.5297	-1.185
	Gov	OHSUMED	**0.4113**	0.3754	+0.201
RankNet	OHSUMED	Gov	*0.4695*	0.5223	-0.137
	Gov	OHSUMED	0.4038	0.4101	
ListNet	OHSUMED	Gov	**0.4799**	0.2332	+0.570
	Gov	OHSUMED	0.3720	*0.3720*	-0.3542
LambdaMART	OHSUMED	Gov	*0.0856*	0.5339	-0.3542
	Gov	OHSUMED	0.3934	0.3917	

performance degradation. Although LambdaMART is a listwise L2R algorithm, the effectiveness is largely affected by the document and query set shift due to the fact that LambdaMART optimizes similarity scores (between query and document) for the individual document while the optimization process considers both pairwise loss and query-level impact. Moreover, when domain change occurs, there is not necessarily a change in the relevance mapping function. Most interestingly, it seems that in most cases, when an effective ranking function is trained on the source collection, it can improve the ranking effectiveness for the target collection when the target model is ineffective. For example, although MQ2008 contains fewer queries than MQ2007, RankNet is ineffective for MQ2007, whereas the effective RankNet model trained in MQ2008 can outperform the ranking function trained on MQ2007.

3.7 Conclusion

In this chapter, the data generating process for L2R training data and the impact of dataset shifts on L2R are discussed. Different types of dataset shifts may occur for two L2R collections and can impact the L2R algorithms in different ways. The empirical study shows that different L2R algorithms will have different reactions towards different types of shift in the datasets. Pointwise algorithms are easily affected by both document and query set shift, pairwise algorithms are less impacted by document set shift while more likely to be affected by the changes in the query set, the listwise algorithm is heavily affected when the query set changes as a result of corresponding changes to the document list distribution. Different L2R collections may differ in various ways. Identifying the shift present in the dataset is critical to determine whether any extra work is needed to improve the effectiveness when one ranking function is applied to a different collection. Moreover, understanding different changes in the dataset can help find better solutions for TR. For example, tackling covariate shift for document features may help develop effective TR techniques for pointwise learning algorithms when document set shift occurs, while minimizing the query distribution difference may be important for improving the transfer effectiveness of listwise learning algorithms when covariate shift problems arise.

Chapter 4

Instance Weighting Methods for Unsupervised TR

It was mentioned in the previous chapter that the change in the data distribution is the main cause of the degradation of a pre-trained ranking model. Weighting the source instances during training to approach the target data distribution is one of the widely-used strategies for tackling the data distribution shift problem. Such an instance-weighting-based solution has been widely used in transfer learning and also some TR problems. This chapter will investigate existing and new query-level instance weighting techniques for unsupervised TR.

Instance weighting at the query level has been the most popular solution for the unsupervised TR problem as the effectiveness of L2R algorithms is evaluated at the query level. Past work has shown that this approach can be used to significantly improve effectiveness. In this chapter, this approach is re-examined on a wide set of publicly available L2R test collections with more advanced learning-to-rank algorithms. Different query-level weighting strategies are examined using two TR frameworks: AdaRank and a new weighted LambdaMART algorithm. Our experimental results show that the effectiveness of different weighting strategies, including those shown to perform well in past work, vary greatly under different transfer environments. In particular, (i) Kullback-Leibler based density-ratio estimation tends to outperform a classification-based approach and (ii) aggregating document-level weights into query-level weights is likely superior to direct estimation using a query-level representation. The Nemenyi statistical test, applied across multiple datasets, indicates that most instance-weighting-based transfer learning methods do not significantly outperform baselines, although there is potential for the further development of such techniques.

4.1 Introduction

Unsupervised TR assumes that there are no relevance judgements available in the target collection. However, this does not exclude other knowledge, such as details of queries that have previously been submitted to search the target collection [18, 76, 89]. To any transfer learning problems, one needs to analyze the types of differences that exist between the source and target collection. Under the unsupervised TR scenario, there is no information about the output space of a target collection, thus it is impossible to model the prior probability shift or domain shift problems that were discussed in the previous chapter. As a result, most unsupervised TR solutions assume that the only difference between the source and target collection is the result of covariate shift. However, unlike problems in natural language processing where the predictive models have large dependence on the problem domain, the task of IR systems is less dependent on the domain since most of the features that are used are based on linguistic statistics (for example, term frequency instead of semantic relations). The correlation of the features with document relevance may not shift substantially when the domain changes. For example, it was shown in the previous chapter that when transferring between collections from the web search domain and the medical domain, we did not always observe performance degradation (Table 3.4). It is also noticeable that the assumption may not hold when a ranking function designed for IR is applied to a different task, e.g., product search. A good use case of unsupervised TR is when the training set for an application is outdated. For example, if the ranking function for a search engine of an e-commerce site was built with data from 1 year ago, the performance may not be optimal for the current corpus as the input feature space may change.

Among the solutions to unsupervised TR, *instance weighting* is a common technique, which assigns weights to training examples in the source collection to adjust source data distributions to better match those in the target collection. Instance weighting can be regarded as the first step for TR, and thus is independent of the learning algorithm used. The difficulty of applying instance weighting to ranking problems comes from the complexity of the training data. For conventional classification or ranking problems, the training data consists of sets of feature vectors, which form a multivariate distribution over training instances, and instance weighting can be applied directly to this distribution. L2R algorithms train and test models at a query level, which consist of a list of feature vectors over documents. Therefore, instance weighting at the query level rather than the individual document level is appropriate but not obvious how to implement (due to the extreme sparsity of the query-level representation). Ways to obtain query-level weights have been investigated, by either aggregating document-level weights or by directly estimating query weights with query representation methods. Some attempts [18, 72] have also been made to apply these techniques to ranking problems, and have shown improvements on some collections. A deeper understanding of the problem is still required, however, such as experimental analysis on a much wider set of TR environments: in terms of both test collections and L2R methods.

Previous research has shown that the effectiveness of transfer learning varies across different environments [67], and depends greatly on the similarity of the source and target collection. TR appears to exhibit a similar phenomenon [21]. L2R datasets can be different from each other in many ways, and thus it is more difficult to evaluate the effectiveness of TR algorithms. To solve all these problems, one needs a better understanding of the problem. In the previous chapter, we demonstrated that the effectiveness of L2R algorithms can be degraded when applying to a different collection, and the change may be affected by various types of shift occurring across the collections.

In this chapter, we answer the following research questions.

- How effective are query-level instance weighting techniques for unsupervised TR problems? Here, we consider how the effectiveness of instance weighting algorithms varies when applied to different L2R algorithms, and investigate the generalization ability of different L2R algorithms.
- How do differences in test collections affect the performance consistency of unsupervised TR algorithms, and how should unsupervised TR algorithms be evaluated across different test collections?
- What is the best way to conduct query-level instance weighting for unsupervised TR?

A key contribution of this chapter is a thorough examination of different query-level instance weighting strategies for unsupervised TR. Different approaches for generating query-level weights are explored and their effectiveness is compared across different TR environments. In particular, the weights are applied in two unsupervised TR frameworks: an existing query-importance-weighted AdaRank, and second, a new weighted version of a state-of-the-art L2R algorithm, LambdaMART. A second contribution is an investigation of different query representation techniques. Past work has considered aggregating document features to generate query-level representation. We systematically explore these approaches, and also introduce a new approach, which is based on the Jensen-Shannon Divergence between features and a base ranker.

With respect to previous work on unsupervised TR techniques, we explore a much wider range of environments. We make use of two datasets from LETOR4.0, two datasets from the Yahoo! Learning to Rank Challenge, and also set up a transfer environment between the MSLR-Web10K dataset and the LETOR 4.0 dataset. We excluded some other transfer cases that were used to demonstrate the dataset shift problems in Chapter 3 due to the small sample size of some collections. Here we created a series of transfer settings that could be as close to reality as possible. More advanced L2R algorithms are studied in this chapter, and we also introduce a visualization method to compare the effectiveness of different models across different datasets.

The results show that the effectiveness of different weighting algorithms, including those that have seen shown to be effective on a particular test set, are inconsistent on other datasets. Our experiments suggest that Kullback-Leibler divergence-based density estimation methods are more effective than classification-based methods. Moreover, aggregating document-level weights appears to outperform direct estimation with a query representation method. We apply a statistical test (Nemenyi) comparing algorithms across multiple datasets and unlike past work, find no significant improvements from instance weighting techniques over the non-weighted models, in contrast to the findings resulting from the more narrow evaluation carried out in past work.

As it was demonstrated in the previous chapter, since L2R collections are represented by document–query vectors of features, when attempting TR from one collection to another, one can examine changes in the make-up of queries, as well as variations in the documents composing the collections. Changes in documents will affect the distribution of features across the collection, as well as the actual features used. Changes in query sets can happen in two ways: query distribution and query type. For query distribution, some queries may appear more often in one dataset than another. For query type, e.g. ad-hoc and navigational queries, some types of queries might appear more often than others. In effect, the definition of relevance has changed across the collections, which impacts on the ranker. Past work in this area [19] only considered the distribution of queries across collections.

4.2 Related Work

Unsupervised TR is a more difficult (and argubly more useful) problem than supervised TR due to the eschewal of the need for supervision information from the target collection. Most of the existing work on unsupervised TR is based on instance weighting.

In Gao et al. [18], the authors generated instance weights at different levels for L2R datasets. Although not pointed out in the paper, their methods are similar to classification-based density-ratio estimation. The authors built a classification hyperplane between the source and target documents and used a sigmoid function of the distance of a target document to the hyperplane at the document-level to weight the document instances. Since documents are independent of one another, the document-pair weights are the multiplication of the weights for the pair of documents. The query weights were generated by averaging the weights of document-pairs in the query. The authors tested their instance weights with RankSVM and RankNet (two pair-wise L2R algorithms[1]) on the six topic sets in LETOR3.0, and showed some significant improvements. Cai et al. [109] further improved the algorithm by classifying the queries directly. In our experiments, the algorithm was also implemented, but instead of using the probability transferred from the distance, we employed a logistic regression classifier which can output the probability directly.

An importance weighted AdaRank approach (wAdaRank) was proposed by Ren et al. [72]. The authors used the Kullback-Leibler Importance Estimation Procedure (KLIEP) to estimate document weights, which were then incorporated into the AdaRank algorithm. However, the algorithm was not tested under an unsupervised TR scenario.

[1]AdaRank and LambdaMART are more effective [36].

Instead, the authors tested the algorithm in a supervised learning environment. The density-ratio was estimated according to the test set, and evaluated on the test set as well.

There are other approaches to unsupervised TR. For example, instead of learning with data from the source collection, Goswami et al. [23] tried to predict relative relevance judgement for document pairs in the target collection and then use the judgements to train a ranking function for the target collection.

4.3 Instance Weighting

Instance weighting aims to address the problem of covariate shift, as described in the previous chapter, by weighting source datapoints using the ratio of their density in the target and source distributions: $w(\mathbf{x}) = \frac{p^{ta}(\mathbf{x})}{p^{so}(\mathbf{x})}$. Assuming conditional distributions are the same across the source and target datasets (i.e. $p^{so}(y|\mathbf{x}) = p^{ta}(y|\mathbf{x})$), then training on the weighted source data will minimize the expected loss (denoted $l(\mathbf{x}, y, \theta)$) under the target distribution:

$$\begin{aligned} f^* &= \underset{\theta}{argmin} \frac{1}{N^{so}} \sum_{(\mathbf{x}_i, y_i) \sim p^{so}} w(\mathbf{x}_i) l(\mathbf{x}_i, y_i, \theta) \\ &\approx \underset{\theta}{argmin} \int p^{so}(\mathbf{x}, y) w(\mathbf{x}) l(\mathbf{x}, y, \theta)\ dxdy \\ &= \underset{\theta}{argmin} \int p^{so}(x, y) \frac{p^{ta}(\mathbf{x})}{p^{so}(\mathbf{x})} \frac{p^{ta}(y|\mathbf{x})}{p^{so}(y|\mathbf{x})} l(\mathbf{x}, y, \theta)\ d\mathbf{x}dy \\ &= \underset{\theta}{argmin} \int p^{ta}(\mathbf{x}, y) l(\mathbf{x}, y, \theta)\ d\mathbf{x}dy \\ &= \underset{\theta}{argmin}\ \mathbb{E}_{p^{ta}(\mathbf{x}, y)}[l(\mathbf{x}, y, \theta)] \end{aligned} \tag{4.1}$$

Many techniques have been developed to efficiently estimate the density-ratio $w(x)$ at the source data points. We examine two popular techniques, namely the Kullbak-Leibler Importance Estimation Procedure (KLIEP) [63] and a classification-based approach. In comparison with the density ratio estimation method in Huang et al. [110], KLIEP is considered state-of-the-art. The KLIEP technique aims to learn a function that minimizes the divergence between the weighted source collection and the target collection:

$$\hat{w}(\mathbf{x}) = \underset{w(\mathbf{x})}{argmin}\ KL(w(\mathbf{x}) p^{so}(\mathbf{x}) || p^{ta}(\mathbf{x})) \tag{4.2}$$

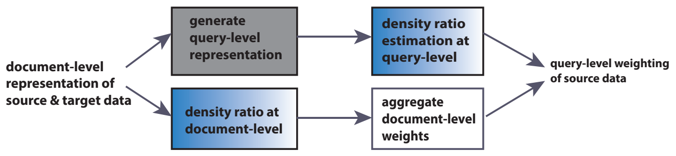

document-level representation of source & target data
generate query-level representation
density ratio at document-level
density ratio estimation at query-level
aggregate document-level weights
query-level weighting of source data

FIGURE 4.1: Pipelines for query-level instance weighting

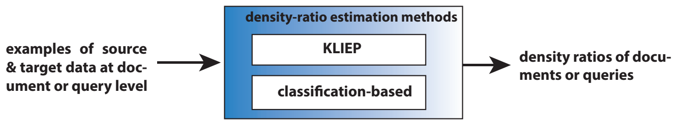

examples of source & target data at document or query level
density-ratio estimation methods
KLIEP
classification-based
density ratios of documents or queries

FIGURE 4.2: Density-ratio estimation methods

As suggested by Sugiyama et al. [64], the density ratio can also be estimated using a classification-based approach. The technique involves combining a sample of data points from the source and target domains, and then training a probabilistic classifier to classify the domain of each instance. Having trained such a model, the density ratio can be estimated as:

$$\hat{w}(\mathbf{x}) = \frac{N^{so}}{N^{ta}} \frac{p(c = target|\mathbf{x})}{1 - p(c = target|\mathbf{x})} \tag{4.3}$$

where N^{so} and N^{ta} are the number of instances in the source and target collection respectively, and $P(c = target|\mathbf{x})$ is the classifier's estimate of the class probability. For example, a logistic regression classifier (LR) can be employed to generate a probabilistic model, $p(c|\mathbf{x}, \theta^*)$.

4.3.1 Instance Weighting for TR

Instance weighting has been used previously for TR. The type of rank learning algorithm (pointwise, pairwise, listwise) determines how instance weighting is estimated. Instance weighting for point-wise algorithms is similar to instance weighting for conventional transfer learning, with density-ratio estimation done at the document level (or individual feature vectors). For pair-wise algorithms, weighting can be done at the document-pair level (on pairs of feature vectors). For list-wise algorithms, it is natural to calculate instance weights at the query level (on sets of feature vectors).

Query-level weighting can be performed by either (i) first estimating document-level weights and then aggregating the weights into query-level values, or (ii) computing a query level representation (from the set of document feature vectors for each query) and then performing density-ratio estimation directly in that space.

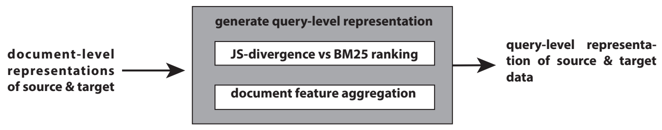

document-level representations of source & target
generate query-level representation
JS-divergence vs BM25 ranking
document feature aggregation
query-level representation of source & target data

FIGURE 4.3: Query representation methods

Estimation of the density ratio of the documents is straightforward, while it is more difficult to estimate the density ratio for queries, since queries are lists of documents instead of single data points. Representing queries in a feature vector space is possible, but there is a danger that the representation could lose the structural information in queries. Similar ideas have been applied for query-dependent learning algorithms, where the difference of queries has been considered for ranking [20, 111]. We explore two approaches to representing queries, as shown in Figure 4.3. The two methods have been successfully used by query-clustering-based L2R algorithms.

4.3.1.1 Document Feature Aggregation

A straightforward method to represent queries is to construct a vector space by aggregating features of the documents retrieved for a query. For example, the following is the representation for the j_{th} query in a collection:

$$\vec{q}_j =< \frac{1}{n}\sum_i \mathbf{x}_{j,i,1}, \cdots, \frac{1}{n}\sum_i \mathbf{x}_{j,i,m} > \tag{4.4}$$

where n is the number of documents in the retrieved list, m is the number of features and $\mathbf{x}_{j,i,k}$ denotes the value of the k_{th} feature in the i_{th} document for query j.

4.3.1.2 Feature Divergence

A simple term representation of queries is unlikely to be effective as term overlap is likely to be low. Equally, comparing document-query features alone in each of the collections is unlikely to be effective, as the value of document-query features is likely to be different between the two collections. We therefore apply a weighting method, inspired by an approach proposed by Peng et al. [111], who used a baseline ranker to act as a normalizing pivot against which document-query features in the collections were measured and compared.

Peng et al. [111] suggested that the effect of a feature on a query can be represented by its divergence from a baseline ranker, such as BM25. The divergence represents how much a document ranking has been changed by a particular feature. This provides a way to normalize measurement of features across collections. Both Kullback-Leibler [112] and Jensen-Shannon (JS) Divergence [113] were used.[2] Since it is symmetric, we use the JS Divergence to calculate a query-level feature vector $\vec{JS}(q_j) = \langle JS_k(q_j), \ldots JS_n(q_j) \rangle$ using the divergence of the feature distribution and the distribution of BM25 in that query as follows:

$$JS_k(q_j) = JS(\mathbf{x}_{j,\cdot,k} || \mathbf{x}_{j,\cdot,b}) = \frac{1}{2}(KL(\mathbf{x}_{j,\cdot,k} || \mathbf{x}_{j,\cdot,b}) + KL(\mathbf{x}_{j,\cdot,b} || \mathbf{x}_{j,\cdot,k})) \tag{4.5}$$

where $\mathbf{x}_{j,\cdot,b}$ is the vector of values for the baseline feature (assumed to be BM25) for query j, and $\mathbf{x}_{j,\cdot,k}$ denotes the vector of values for the k^{th} feature. $KL(.)$ denotes the Kullback-Leibler Divergence [112] between two distributions, calculated as:

$$KL(\mathbf{x}_{j,\cdot,k} || \mathbf{x}_{j,\cdot,k'}) = \sum_{i=1}^{N} \mathbf{x}_{j,i,k} \log_2 \frac{\mathbf{x}_{j,i,k}}{\mathbf{x}_{j,i,k'}} \tag{4.6}$$

Here, $\mathbf{x}_{j,i,k}$ is the document score of d_i assigned by the k^{th} feature and N is the number of documents in a ranked list. The JS Divergence method for query representation is illustrated in Figure 4.4.

This query representation is similar as that used by Peng et al. [111], which to the best of our knowledge has never been applied to TR before. Notice that Equations 4.5 and 4.6 are usually applied to probability distributions, but in Peng et al. [111] un-normalised feature vector values were used in the equations. Hence in this work, we followed their implementation.

4.4 Unsupervised TR Frameworks

We describe two TR frameworks that can incorporate the query weights into training: an existing framework modified from AdaRank for importance weighting [72]; and a new weighted LambdaMART approach that we developed.

[2]Peng et al. used divergence of features to represent queries for clustering.

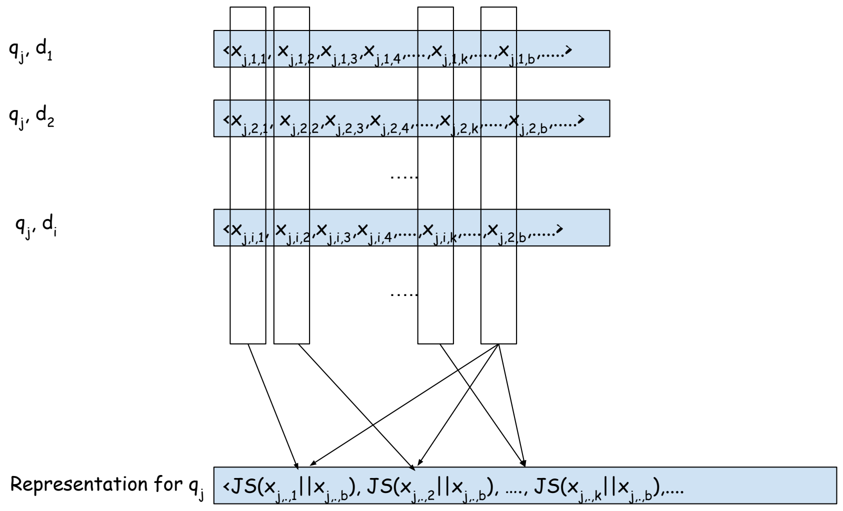

q_j, d_1
<x_{j,1,1}, x_{j,1,2}, x_{j,1,3}, x_{j,1,4}, ..., x_{j,1,k}, ..., x_{j,1,b}, ...>
q_j, d_2
<x_{j,2,1}, x_{j,2,2}, x_{j,2,3}, x_{j,2,4}, ..., x_{j,2,k}, ..., x_{j,2,b}, ...>
q_j, d_i
<x_{j,i,1}, x_{j,i,2}, x_{j,i,3}, x_{j,i,4}, ..., x_{j,i,k}, ..., x_{j,2,b}, ...>
Representation for q_j
<JS(x_{j,.,1}||x_{j,.,b}), JS(x_{j,.,2}||x_{j,.,b}), ..., JS(x_{j,.,k}||x_{j,.,b}),....

FIGURE 4.4: JS Divergence Representation for Queries

4.4.1 Weighted AdaRank (wAdaRank)

We integrate query weights into AdaRank, a listwise L2R model [51].[3] AdaRank learns an ensemble model F, which is a linear combination of weak rankers:

$$F(\mathbf{x};\beta,\alpha) = \sum_{i=1}^{ta} \beta_i h_i(\mathbf{x};\alpha_i) \tag{4.7}$$

where and $h_i(\mathbf{x};\alpha_i)$ is the weak learner added in the i^{th} iteration (with parameter α_i) and β_i is the corresponding weight.

AdaRank learns the ensemble using boosting. At each iteration a new weak ranker is added to the ensemble that provides maximum effectiveness improvement on the weighted training set. Weights are assigned at the query level based on the current performance of the ensemble.

Thus adding density-ratio weights to AdaRank is straightforward and involves modifying the initial weighting of queries in the AdaRank algorithm. Following query importance weighted AdaRank (wAdaRank) [72], density-ratio weights are assigned to queries at the initial stage. When updating query weighting at the end of each iteration, the query distribution will be determined by density-ratio weights together with their performance weights.

[3]Our weighting strategy can be used for any listwise L2R algorithm.

4.4.2 Weighted LambdaMART (wλMART)

LambdaMART [53] is an L2R algorithm which uses gradient-boosted regression trees to optimize a listwise objective function, which depends on the chosen evaluation metric. Thus, similar to AdaRank, LambdaMART also relies on a boosting-based technique that outputs an ensemble of weak learners as in Equation (4.7), except that the weak learners $h(\mathbf{x};\alpha_i)$ are regression trees rather than single feature-based predictors as was the case for AdaRank. The reason for introducing regression trees is that they have proven effective for training ranking models. At each iteration, LambdaMART fits a new regression tree to the gradient of the objective function of the current ensemble.

Since the Lambda gradient provides a score for each query and document-pair (denoted λ_{ij}), which estimates the ranking performance improvement that would result from increasing the score of the document for the query, in order to modify LambdaMART to handle query-level weights we need only modify the tree learning part of the algorithm to make use of weighted examples.

For a regression tree, the prediction at each leaf of the tree is simply the average value of the training examples assigned to the leaf. (The average value is chosen because it minimizes the squared error of the prediction at the node). Trees are grown by recursively splitting the data present at each leaf node. For each leaf, the feature k and split-point s is chosen that results in the minimum sum of squared errors across the resulting branches:

$$(k,s)^* = \underset{k,s}{argmin} \sum_{i:\mathbf{x}_{ik}\leq s} (\lambda_i - \bar{\lambda}_L)^2 + \sum_{i:\mathbf{x}_{ik}>s} (\lambda_i - \bar{\lambda}_R)^2 \tag{4.8}$$

Here $\bar{\lambda}_L$ and $\bar{\lambda}_R$ denote the average values on the left and right of the split-point s. If weights are associated with data points, then we can learn a weighted regression tree by using the weighted squared error as the objective:

$$(k,s)^* = \underset{k,s}{argmin} \sum_{i:\mathbf{x}_{ik}\leq s} w_i(\lambda_i - \bar{\lambda}_L)^2 + \sum_{i:\mathbf{x}_{ik}>s} w_i(\lambda_i - \bar{\lambda}_R)^2 \tag{4.9}$$

Where $\bar{\lambda}_L = \frac{1}{V_L}\sum_{i:\mathbf{x}_{ik}\leq s} w_i\lambda_i$ now denotes the *weighted average* on the left side of the split, (since that is the prediction that minimizes the weighted squared error for the data on the left), and $V_L = \sum_{i:\mathbf{x}_{ik}\leq s} w_i$ denotes the sum of the weights on the left of the split. ($\bar{\lambda}_R$ and V_R are defined analogously.) The optimization objective can then be rewritten and simplified to:

$$\begin{aligned}
(k,s)^* = & \underset{k,s}{argmin} \textstyle\sum_{i:\mathbf{x}_{ik}\leq s} w_i(\lambda_i^2 - 2\lambda_i\bar{\lambda}_L + \bar{\lambda}_L^2) \qquad (4.10)\\
+ & \textstyle\sum_{i:\mathbf{x}_{ik}>s} w_i(\lambda_i^2 - 2\lambda_i\bar{\lambda}_R + \bar{\lambda}_R^2)\\
= & \underset{k,s}{argmin} \textstyle\sum_i w_i\lambda_i^2 + \sum_{i:\mathbf{x}_{ik}\leq s}(-2w_i\lambda_i\bar{\lambda}_L + w_i\bar{\lambda}_L^2)\\
+ & \textstyle\sum_{i:\mathbf{x}_{ik}>s}(-2w_i\lambda_i\bar{\lambda}_R + w_i\bar{\lambda}_R^2)\\
= & \underset{k,s}{argmin} \textstyle\sum_i w_i\lambda_i^2 - 2\bar{\lambda}_L \sum_{i:\mathbf{x}_{ik}\leq s}(w_i\lambda_i) + \sum_{i:\mathbf{x}_{ik}\leq s} w_i\bar{\lambda}_L^2\\
- & 2\bar{\lambda}_L \textstyle\sum_{i:\mathbf{x}_{ik}>s}(w_i\lambda_i) + \sum_{i:\mathbf{x}_{ik}>s} w_i\bar{\lambda}_R^2\\
= & \underset{k,s}{argmin} \textstyle\sum_i w_i\lambda_i^2 - 2\bar{\lambda}_L^2 V_L + V_L^2\bar{\lambda}_L^2\\
- & 2\bar{\lambda}_R^2 V_R + V_R^2\bar{\lambda}_R^2\\
= & \underset{k,s}{argmin} \textstyle\sum_i w_i\lambda_i^2 - (V_L\bar{\lambda}_L^2 + V_R\bar{\lambda}_R^2)
\end{aligned}$$

And since the first term $\sum_i w_i\lambda_i^2$ is constant (independent of the chosen split-point), it can be dropped from the equation. For speed, we calculate running weighted sums $S_L = \sum_{i:\mathbf{x}_{ij}\leq s} w_i\lambda_i$ and $S_R = \sum_{i:\mathbf{x}_{ij}>s} w_i\lambda_i$ and maximise the following objective:

$$(k,s)^* = \underset{k,s}{argmax} \frac{(S_L)^2}{V_L} + \frac{(S_L)^2}{V_R} \tag{4.11}$$

Thus the difference between the wλMART and normal LambdaMART is that the regression tree is built using instance weights, as shown in Algorithm 1[4]. Note that query-level weights are passed down to the document-level (i.e., all the documents in a query will be assigned the query-level weight).

4.5 Experiments

Several experiments were conducted to analyze the effectiveness of different TR techniques.

[4]The weights are also used when updating the tree outputs with the Newton step.

Algorithm 1: Pseudo-code for growing the weighted regression tree for the wλMART algorithm

```
Input: $\{\mathbf{x}_i, \lambda_i, w_i\}_{i=1}^N$, document feature vectors, corresponding $\lambda$-gradient values,
       and instance weights for instances
$leafCount = 1$;
WeightedTree($\{\mathbf{x}_i, \lambda_i, w_i\}_{i=1}^N$)
    if $leafCount \geq 10$ then
        return leaf with prediction: $\bar{\lambda} = \frac{\sum_i w_i \lambda_i}{\sum_i w_i}$;
    else
        /* find best feature and split point */
        $bestgain = 0$;
        $k = -1; s = -1$;
        for $feature \in \{1, \ldots, m\}$ do
            $sort(\{x_i\}_{i=1}^N, feature)$ $S_L = 0$; $S_R = \sum_i w_i \lambda_i$;
            $V_L = 0$; $V_R = \sum_i w_i$;
            $previous = -\infty$;
            for $split \in possible\_splits(X, feature)$ do
                $\Delta S = \sum_{i:previous < \mathbf{x}_{i,feature} \leq split} w_i \lambda_i$;
                $\Delta V = \sum_{i:previous < \mathbf{x}_{i,feature} \leq split} w_i$;
                $S_L = S_L + \Delta S$; $S_R = S_R - \Delta S$;
                $V_L = V_L + \Delta V$; $V_R = V_R - \Delta V$;
                $gain = (S_L)^2 / V_L + (S_R)^2 / V_R$;
                if $gain > bestgain$ then
                    $bestgain = gain$;
                    $k = feature$; $s = split$;
                end
                $previous = split$;
            end
        end
        if $bestgain > 0$ then
            /* recurse to child nodes */
            $leafCount = leafCount + 1$;
            $left$ =WeightedTree($\{\mathbf{x}_i, \lambda_i, w_i\}_{i:\mathbf{x}_i k \leq s}$);
            $right$ =WeightedTree($\{\mathbf{x}_i, \lambda_i, w_i\}_{i:\mathbf{x}_i k > s}$);
        else
            return leaf with prediction: $\bar{\lambda} = \frac{\sum_i w_i \lambda_i}{\sum_i w_i}$;
        end
    end
```

4.5.1 Collections

To validate the TR techniques, three existing most widely-used L2R collections were used: the LETOR 4.0 dataset, the Microsoft Learning to Rank datasets (MSLR[5]), and

[5]We used the subset of MSLR: MSLR-10K. To keep denotation simple, we refer to the subset as MSLR.

Table 4.1: Transfer settings for testing different algorithms

Yahoo! Learning to Rank						
	Collection	Queries	Features	Collection	Queries	Features
Source	Set 1	19,944	415	Set 2	6330	415
Target Training	Set 2	5064	415	Set 1	15955	415
Target Testing	Set 2	1266	415	Set 1	3989	415
LETOR 4.0						
	Collection	Queries	Features	Collection	Queries	Features
Source	MQ2007	1,700	46	MQ2008	800	46
Target Training	MQ2008	640	46	MQ2007	1440	46
Target Testing	MQ2008	160	46	MQ2007	360	46
LETOR 4.0 and MSLR						
	Collection	Queries	Features	Collection	Queries	Features
Source	LETOR 4.0	2,340	45	MSLR-10K	10k	45
Target Training	MSLR-10K	8k	45	LETOR 4.0	1,872	45
Target Testing	MSLR-10K	2k	45	LETOR 4.0	468	45

the Yahoo! Learning to Rank (Yahoo!L2R) datasets, see Table 2.1. Each collection was set up to contain a pair of datasets to simulate transfer from source to target. In LETOR 4.0 we use the same document collection, but different query sets so we can examine how different weighting methods perform when the source and target collection are drawn from the same distribution.

The Yahoo!L2R collection is composed of two datasets, which were created to test TR. The documents and queries of Set 1 are pooled from a non-English search engine, and the size of query set is smaller than that of Set 2. The two datasets share 415 features in common, and we used the 415 features in our experiment. As Yahoo has anonymised the features, we could not provide any details for those features.

We also examined TR using the MSLR dataset and the LETOR 4.0 dataset. The two datasets share relatively few commonalities: the document and query sets are different, the pooling strategies are different, the relevance judgments are different, and even the feature normalizations are different. However, these two datasets share 45 features (feature 1 - feature 45 of LETOR 4.0), which gave us an opportunity to study unsupervised TR in a more realistic scenario. When transferring between LETOR 4.0 and MSLR, we merged the two query sets in LETOR 4.0 to make a larger collection.

In each group, we select source and target collections, and then randomly split the target collection into five folds for cross-validation evaluation. In each experiment, four folds were taken as training data, and we removed the labels of the training set to simulate an unsupervised TR experiment. On both test collections, we assigned one of the collections as the source and the other as the target The TR algorithms were tested in both "directions", first with an initial assignment and then with the assignments reversed. The transfer settings are shown in Table 4.1.

Note, we did not include the LETOR3.0 test collections in this experiment (although it has been used in past work) as the number of queries is too small for accurate density-ratio estimation.

4.5.2 Measures

The effectiveness measure and training objective function used was Normalized Discounted Cumulative Gain (NDCG) [106] with the standard rank-plus-one discount function and exponential gain [42]: the details of NDCG can be referred to in section 2.2.3 in Chapter 2. We measure NDCG at rank cutoff 10. We conducted two-tailed t-tests between results with α (significance level) set to 0.05 and the Friedman test.

4.5.3 Setup

The implementation of the algorithms used the open source L2R library Ranklib2.1.[6] For all AdaRank-based algorithms, we set the iteration number to 500. For all LambdaMART-based algorithms, we trained 1,000 trees with ten leaves with jForests-0.5 library [114][7]. The learning rate was set at 0.1. Sugiyama-Sato's KLIEP code[8] was used to estimate density-ratio at the document or query level.

4.5.4 Comparison Models

Different models were investigated to compare different aspects of instance weighting techniques for unsupervised TR. Two TR frameworks, wAdaRank and wλMART, were used in the experiments. The following baselines are used as comparison points:

[6] http://sourceforge.net/p/lemur/wiki/RankLib/
[7] https://github.com/yasserg/jforests
[8] http://www.ms.k.u-tokyo.ac.jp/software.html

- **BM25** was used as a baseline to examine whether a TR algorithm can exceed the performance of a static ranker. Notice that we used the BM25 feature provided by the collections as we do not have access to the original corpus.
- **Ada.source** was the model trained by AdaRank with all the data from the source collection. This is a simple TR algorithm with the source model applied directly to the target collection with no adjustment.
- λ**MART.source** was the model trained by LambdaMART with all the data from the source collection. Again the source model is applied directly to the target dataset with no adjustment.
- **Ada.target** was the model trained by AdaRank with data from the target collection; the results were measured using 5-Fold cross validation. One can view the performance of the target model as an upper bound to which the designer of the TR algorithms aspires to achieve.
- λ**MART.target** was the model trained by LambdaMART with data from the target collection, the results were measured from 5-Fold cross validation. Again, this model trained on the target should provide an upper bound on performance.

The following instance weighting algorithms were investigated in the experiments:

- **kliep.doc**: density-ratio estimation at document level using KLIEP, aggregating document weights of queries into query-level weights. kliep.doc with wAdaRank (wAdaRank-kliep.doc) was proposed in Ren et al. [72].
- **kliep.avg**: density-ratio estimation at query level using KLIEP, with the feature aggregating representation method.
- **kliep.js**: density-ratio estimation at query level using KLIEP, with a JS divergence based representation.
- **class.doc**: density-ratio estimation at document level using a classification-based method, aggregating document weights of queries into query level weights. This approach was most close to the weighting strategy proposed by Gao et al. [18].
- **class.avg**: density-ratio estimation at the query level using a classification based method, with a feature aggregating representation method.
- **class.js**: density-ratio estimation at query level using a classification based method, with a JS divergence-based representation.

Table 4.2: Effectiveness (NDCG@10 score) on different transfer settings with wAdaRank. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than Ada.source, ↓ denotes the figure is significantly worse than Ada.source. $p < 0.05$

	MQ2007-MQ2008	MQ2008-MQ2007	Yahoo.Set 1-Yahoo.Set 2	Yahoo.Set 2-Yahoo.Set 1	MSLR-LETOR 4.0	LETOR 4.0-MSLR
BM25	0.335	0.249	0.540	0.507	0.276	0.180
Ada.source	0.495	0.353	0.658	0.701	0.367	0.251
kliep.doc	0.329 ↓	**0.431** ↑	**0.708** ↑	0.704 ↑	0.286 ↓	0.196 ↓
kliep.avg	0.379 ↓	0.383 ↑	0.684 ↑	0.695 ↓	0.370 ↑	0.281 ↑
kliep.js	0.493	0.384 ↑	0.694 ↑	0.705 ↑	0.402 ↑	0.274 ↑
class.doc	0.497	0.424 ↑	0.690 ↑	0.688 ↓	0.362 ↓	0.303 ↑
class.avg	**0.501**	0.265 ↓	0.566 ↓	0.605 ↓	0.362 ↓	0.140 ↓
class.js	0.363 ↓	0.383 ↑	0.561	0.667	0.370 ↑	0.281 ↑
Ada.target	0.494	0.417 ↑	0.698	**0.710** ↑	**0.447** ↑	**0.304** ↑

Table 4.3: Effectiveness (NDCG@10 score) on different transfer settings with wλMART. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than λMART.source, ↓ denotes the figure is significantly worse than λMART.source. $p < 0.05$

	MQ2007-MQ2008	MQ2008-MQ2007	Yahoo.Set 1-Yahoo.Set 2	Yahoo.Set 2-Yahoo.Set 1	MSLR-LETOR 4.0	LETOR 4.0-MSLR
BM25	0.335	0.249	0.540	0.507	0.276	0.180
λMART.source	**0.505**	0.407	0.718	0.702	*0.236*	0.197
kliep.doc	0.499 ↓	0.412 ↑	0.712 ↓	0.703	*0.273* ↑	0.200 ↑
kliep.avg	0.498 ↓	0.384 ↓	0.705 ↓	0.697	0.271 ↑	0.180 ↓
kliep.js	0.473 ↓	0.395	0.710 ↓	0.700 ↓	0.295 ↑	0.222 ↑
class.doc	0.496 ↓	0.413 ↑	0.710 ↓	0.697 ↓	0.289 ↑	0.202 ↑
class.avg	0.495 ↓	0.408	0.693 ↓	0.690 ↓	0.289 ↑	0.213 ↑
class.js	0.466 ↓	0.392 ↓	0.698 ↓	0.686 ↓	*0.273* ↑	0.226 ↑
λMART.target	0.501	**0.455** ↑	**0.763** ↑	**0.742** ↑	**0.463** ↑	**0.429** ↑

4.6 Result Analysis

The effectiveness of the instance weighting algorithms with wAdaRank and wλMART are now discussed.

4.6.1 Comparing AdaRank and LambdaMART

LambdaMART was found to be more effective than AdaRank on all the six datasets (as shown in the last lines of Tables 4.2 and 4.3). However, when we use models trained with data from a source collection and apply them directly to a target collection (the source models, i.e. Ada.source and λMART.source), the effectiveness varies. We represent the effectiveness of AdaRank and LambdaMART models on different datasets and examine how the trained model can best be generalized to another collection.

The document collection is common across MQ2007 and MQ2008. Since MQ2007 contains more queries, both AdaRank and LambdaMART trained on MQ2007 were more effective than the models trained on MQ2008 when testing on both collections.

The difference between Set 1 and Set 2 in Yahoo!L2R are bigger than the difference between the two datasets in LETOR 4.0. When transferring from Set 1 to Set 2, both Ada.source and λMART.source saw a 6% performance decrease compared with target models (Ada.target and λMART.target). However, in the opposite direction, Ada.source is 1% worse than Ada.target while LambdaMART suffers larger performance decrease (6%). However, source modes trained with LambdaMART are still better than those trained with AdaRank.

In the last group of transfer settings, where the dissimilarity is also the largest, the decrease of model performance is even greater. Compared with Ada.target, Ada.source saw a 17.9% and 17.4% decrease in system effectiveness when transferring from MSLR to LETOR 4.0 and from LETOR 4.0 to MSLR respectively. LambdaMART suffers from an even greater drop; the λMART.sources are 49% and 54.1% worse than the λMART.targets in the two transferring scenarios. During the training, LambdaMART looks into the feature space to find the best splits to minimize the loss function, while AdaRank is just looking to find the best features that gain the best performance on the query sets. Thus, LambdaMART is more sensitive to the distribution of the feature space. In the last group of collections, the feature spaces of LETOR 4.0 and MSLR are very different from one another. Moreover, the features in LETOR 4.0 have been normalized while MSLR datasets kept the original feature values.

4.6.2 Transferring with Different Algorithms

We now discuss how the performance of the weighted wAdaRank and wλMART perform differently even with the same weighting strategies, and on the same datasets. As shown in the third columns of Tables 4.2 and 4.3, when transferring from Set 1 to Set 2 of Yahoo!L2R dataset. The kliep.doc, kliep.avg, kliep.js, and class.doc methods are significantly better than the Ada.source when they are used in the wAdaRank framework. However, with the same set of weighting algorithms, wλMART decreases the effectiveness of λMART.source. Similarly, all the wλMART algorithms outperform the LambdaMART without weights (λMART.source), when transferring from MSLR to

LETOR 4.0, while some of the wAdaRank algorithms reduce the performance under the same environment, as shown in the last columns of Table 4.2 and 4.3.

The instance-weighting strategies appear more effective with wAdaRank than with wλMART. Among all the six datasets, in 17 out of 36 cases, wAdaRank models are significantly better than Ada.source, and in 11 cases, the algorithms are significantly worse than Ada.source, while with wλMART, there are only 12 of 36 wining cases, with 19 losing cases.

4.6.3 Transferring Across Different Datasets

The effectiveness of different instance-weighting algorithms vary under two TR frameworks and also under different collections. We analyze the different weighting algorithms in different datasets in this section.

4.6.3.1 LETOR 4.0

As we mentioned, MQ2007 and MQ2008 are two samples from the same distribution, while MQ2007 has a larger query set size than MQ2008. Transferring from MQ2007 to MQ2008 appears not to work at all with all weighting methods under either TR framework. However, when transferring is executed in the opposite direction, some improvements occur with different weighting approaches. Most of the figures from the third column of Table 4.2 are significantly better than Ada.source, except the wAdaRank.class.avg. We speculate that since MQ2007 has a larger query set, this tends to be less biased than MQ2008, and since MQ2007 and MQ2008 are two samples from the same distribution, estimating the density-ratio with respect to a larger sample would improve accuracy. Moreover, since the size of MQ2008 is small, the test set would contain a small number of instances for testing, which could cause the variation in test results.

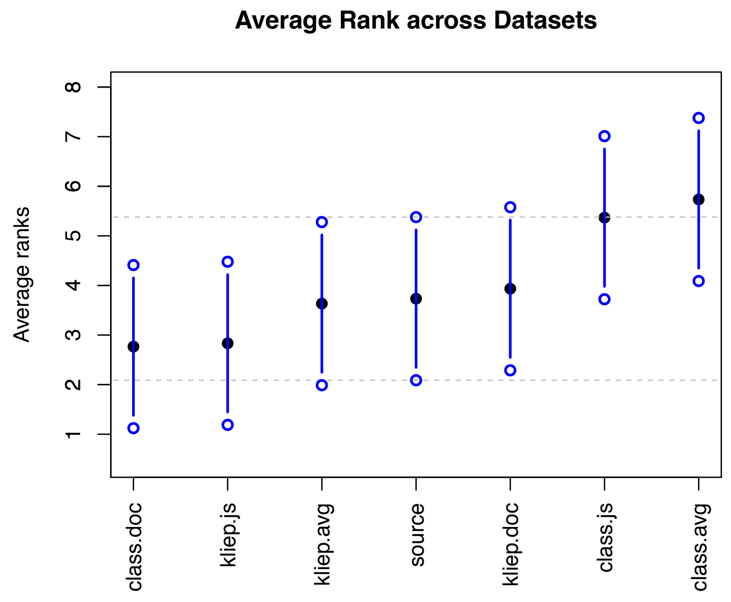
Yahoo!L2R The two ranking frameworks show differences. KLIEP-based weighting strategies appear to work with the wAdaRank framework on the Yahoo!L2R datasets. However, when the technique is combined with wλMART, it shows no improvements in both transfer directions. Close investigation shows that there were many missing feature values in the Yahoo!L2R datasets, which likely increases the difficulty (lowers the accuracy) of estimating the density-ratio at both the document and query levels.

MSLR-LETOR 4.0 Transferring between MSLR and LETOR 4.0 is the most challenging task among all the transferring settings, since the datasets are dissimilar. Although the source models suffer from a substantial effectiveness drop compared with target models, many of the examined weighting algorithms appear to work. Consistent improvements are observed from kliep.js and class.js with both frameworks. When transferring from MSLR to LETOR 4.0, the wAdaRank.kliep.js algorithm in Table 4.2 gains 9.5% effectiveness compared with the Ada.source. When the same weighting approach was used in wλMART, a 25% performance boost was achieved compared with λMART.source under the same transferring setting. Transferring in the other direction, from LETOR 4.0 to MSLR, also shows improvements, from the weighting algorithms. For example, wAdaRank.class.doc outperforms Ada.source by 5.2%, while wλMART.class.js increased effectiveness by 25% from λMART.source.

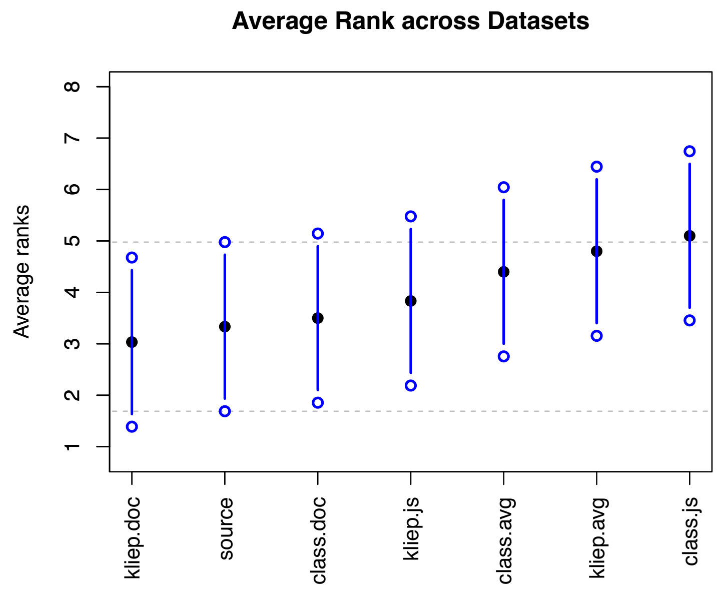
4.6.4 Does Query-Level Instance Weighting Work?

A challenging question for unsupervised TR is how to measure the performance of a transferred ranking function. In reality, this is not possible as there are no relevance labels in the target collection. Instead, we want to develop an unsupervised algorithm that performs robustly well across different scenarios. As a result, we compare the performance of different unsupervised TR algorithms across different transfer settings. Friedman's rank-based test [105] and its post test has the capacity to measure the differences in the ranks of performances of systems, which has been previously used by comparing classification algorithms across different test collections [115]. In this chapter, we use the similar method to compare the performance of different unsupervised TR algorithm.

None of the six weighting algorithms consistently improved effectiveness over the source models in all transferring scenarios. It would appear that in some scenarios, source models can be easily transferred to gain better performances, for example, transferring from MSLR to LETOR 4.0. However, there are also cases, for a particular transferring direction, where none of the algorithms work at all.



(a) Compare all Adarank-based models



(b) Compare all LambdaMART-based models

FIGURE 4.5: Plots of average rank across the 6 test environments for the 6 different Transfer Learning techniques and the "source" baseline system (where no TR was applied), the critical distance (CD) for the Nemenyi test (at the 5% confidence level). The lower the rank the better performance of the approach.

It is very difficult to distinguish the better algorithms from the poorer ones based on inspection of the results tables. Thus we visualize the results in Figure 4.5 by computing the average rank[9] for each approach across all datasets (and all folds). The Nemenyi test is used to determine whether there is significant difference between the average

[9]The average rank of a system across different test datasets is calculated as $\overline{r}_j = \frac{1}{N}\sum_i r_i^j$, where N is the number of datasets, and r_i^j is the rank of j_{th} model in i_{th} dataset.

rank of any two systems. It can be performed after first checking with the Friedman test (a non-parametric alternative to repeated measures ANOVA) that the systems are not independent of rank (across the datasets). The similar method has been used to compare classification algorithms across multiple datasets [115].

The differences between models are compared against the critical distance (CD)[10]; two models are not significantly different if the average ranks $\overline{r}$ of two models are within the CD. We examine CD on the average rank graph to determine whether one model is more effective than another. The results of the tests are displayed in Figure 4.5, where the y-axis shows the average ranks of the models. The black dots show the average ranks of particular models, and the blue lines represent the CD. If a model's mean rank lies outside the CD for another model, then they are significantly different.

The tests on AdaRank (Figure 4.5(a)) show that some instance-weighting methods may be more effective than the non-weighted AdaRank. However, the differences are not significant. The document-level classification-based algorithm (*class.doc*) and the KLIEP method with JS query representation method (*kliep.js*) show some improvements over Ada.source (AdaRank-source in the figure). KLIEP-based weighting methods are better than classification based algorithms, as most KLIEP based algorithms are ranked above or around the Ada.source models. Two algorithms, *class.js* and *class.avg*, are most likely to be useless for wAdaRank since they are significantly worse than the non-weighted model.

The effectiveness of the weighting approaches are different in LambdaMART as compared with AdaRank. Most of the weighting methods are less effective than the λMART.source model, except the document-level KLEIP method (*kliep.doc*). The query-representation-based methods make things worse, shown by lower ranks compared with the λMART.source model, or even the other two document-level methods.

Query representation methods are an attempt to represent queries at a high level. If the method does not properly represent the properties of queries, for example, averaging document features ignores the ranking preferences of documents in the query, then the density of queries will be estimated incorrectly. Instead, density-ratio estimation at the

[10]When calculating the CD, 5 folds of all the six datasets were used, results of individual folds would likely show some correlation (due to the fact that the independently drawn data for each fold comes from the same distribution), but that any such correlation would inflate the false discovery rate, which is not an issue here since we are not claiming significant improvements.

document level estimates the distributions in the feature input space, which is also the direct inputs to an algorithm, which means that the density-ratio density estimation is then likely more accurate and meaningful. But then how best to use the document-level weights for ranking is another issue.

4.7 Conclusions

This chapter compared a number of query-level weighting algorithms for unsupervised TR. Query-level weights can be generated in two ways, by aggregating document-level weights, or by estimating query-level weights directly based on a query representation method. In this chapter, a set of query-level methods, with different levels, different query representations, and also different density estimation approaches were tested with two widely-used unsupervised TR frameworks, namely weighted versions of AdaRank and LambdaMART. The experiments were conducted on six large-scale unsupervised TR scenarios, which, to our best knowledge, has not been attempted before.

To answer the research question of whether query-level instance weighting is effective for solving unsupervised TR problems, we compare the effectiveness of different weighting methods as well as unsupervised TR algorithms. The results show that the generalization ability of different L2R algorithms are different, and that this strongly depends on the similarity of the datasets. AdaRank appears to have better generalization ability than LambdaMART, especially when the source and target collection are less similar. The effectiveness of instance weighting algorithms is also different when they are applied to different algorithms.

Experiments across different datasets showed that there are no consistent improvements over the non-weighted models for any of the weighting methods, which answered the research question of how do differences in test collections affect the performance consistency of unsupervised TR algorithms. Furthermore, the performance of different algorithms, including those tested in past work, varies substantially under different transferring environments. The visualization method of average rank provides a solution to the research question of how to evaluate unsupervised TR algorithms across different test collections. The Nemenyi test, comparing different models across all the testing datasets, shows that none of the weighting algorithms are significantly better than the original non-weighted models. Nevertheless, we have observed the improvements of some weighting algorithms with different unsupervised TR frameworks, which suggests there is some potential for these algorithms.

Different query-level weighting algorithms were compared to determine which is the best way to conduct query-level instance weighting for unsupervised TR. Query-level weighting methods work better with AdaRank than LambdaMART, since LambdaMART is a lower bias (higher variance) learner, which is more sensitive to changes in the feature space. As it turns out, aggregating document weights to generate query-level weights works better than estimating weights based on a query representation. However, queries are represented as a ranked set of documents, so generating representations for queries can be complicated, and without a qualified query representation method there is a risk that the density estimation could be meaningless.

The findings of this chapter illustrated that it is hard to capture the concept of "query distribution" from a mathematical viewpoint, which makes it even harder to model the distribution change at the query level. However, the ranking models are optimized and evaluated at the query level, and minimizing the gap between the source and target collection distribution is a necessary step before conducting any ranking transfer, which inspired our further study in self-labeling methods for TR.

Chapter 5

Self-Labeling Methods for Unsupervised TR

As has been shown in the previous chapter, query-level instance weighting for unsupervised TR is difficult due to the difficulty of measuring the data distribution for queries in L2R datasets. Alternatively, one can use knowledge in the source collection to estimate the relevance labels for queries in the target collection to enable better knowledge transfer. We propose three self-labeling methods for unsupervised TR: an expectation maximization-based method (RankPairwiseEM) for estimating pairwise preferences across documents, a hard assignment expectation maximization-based algorithm (RankHardLabelEM) which directly assigns imputed relevance labels to documents, and a self-learning algorithm (RankSelfTrain) which gradually increases the number of imputed labels. We compare the three algorithms on three large public test collections using LambdaMART as the base ranker and find that (i) all the proposed algorithms show improvements over the original source ranker in different transferring scenarios; (ii) RankPairwiseEM and RankSelfTrain significantly outperform the source rankers across all environments, and are not significantly worse than the model directly trained on the target collection; and (iii) self-labeling methods are significantly better than previous instance weighting-based solutions on a variety of collections.

5.1 Introduction

In the last chapter, we investigated the use of instance weighting techniques to tackle unsupervised TR problems: weights are assigned to training instances in the source collection to change the data distribution to be more like the distribution in the target. The objective of L2R algorithms is to maximize the ranking effectiveness of a ranking function for queries in a collection. As a result, instance weighting at the query level is a more natural and effective approach. However, queries are composed by a set of query-document pairs (represented by feature vectors), which makes it difficult to measure the density ratios for instance weighting. We demonstrated that the effectiveness of such algorithms varies substantially across different transfer scenarios in the previous chapter.

An alternative TR approach is to directly impute relevance labels for the query document pairs in a target collection and then use these imputed labels to train a rank learner on the target dataset. This *self-labeled* [116] solution is related to self-training and co-training methods, which have also been applied in transfer learning [117]. By gradually imputing new labels for unlabeled instances in the target collection, the algorithm can bypass the difficult problem of density ratio estimation for the L2R collections.

In this chapter, we propose three different self-labeling techniques: an expectation maximization (EM)-based TR algorithm (RankPairwiseEM), a "hard EM"-inspired TR algorithm (RankHardLabelEM), and a self-training for TR algorithm (RankSelfTrain). The RankPairwiseEM algorithm looks to improve the ranking function by iteratively estimating pairwise preference probabilities between documents in the unlabeled target data and then uses these probability estimates as weights in the learning algorithm. The other two algorithms, aim to directly impute relevance labels for the unlabeled query-document pairs in the target collection. RankHardLabelEM is inspired by a variant of the EM algorithm, which makes "hard" (non-weighted) assignments of relevance labels to unlabeled training instances, while RankSelfTrain is an application of the self-training algorithm for TR.

While these algorithms have been studied in other contexts, such as classification and regression problems, they could not be directly applied to TR algorithms for several reasons. Firstly, the data generating process of L2R datasets is different and more complicated than for conventional machine learning datasets. Secondly, most L2R-trained ranking functions only predict the rank order of documents, rather than the relevance labels of individual documents for a given query. This makes it difficult to determine the most likely relevance label for a document as well as the confidence of the prediction. Finally, unlike conventional classification or regression algorithms which look to minimize the expected loss on the instance-level, the effectiveness of a ranking function will be measured on a query-level basis.

The following research questions are addressed to gain a better understanding of the self-labeling process for the unsupervised TR:

- How can one apply the self-labeling methods to transfer knowledge from the source to the target collection within the L2R setting?
- Which self-labeling method is most effective in the L2R TR setting?
- Are self-labeling methods more effective and/or robust than instance-weighting methods for unsupervised TR?

We demonstrate that self-labeling methods are more reliable than instance-weighting for unsupervised TR and that the effectiveness of instance-weighting varies with source collections of different sizes. We test three unsupervised TR algorithms on three large public test collections and show that both RankPairwiseEM and RankSelfTrain have significantly better performance than a non-transferred source model. Moreover, both algorithms are not significantly worse than the target model.

5.2 Related Work

Apart from instance-weighting methods, an alternative approach to unsupervised transfer learning is self-labeling [118]. Self-labeling propagates labels from the source to the target data by directly imputing relevance labels for unlabeled instances in a target collection. A study by Triguero et al. [116] found that self-labeling methods are effective for various semi-supervised learning tasks.

Several solutions have been investigated to implement self-labeling, including EM algorithms [119], self-training algorithms [120], and multi-view learning [94], which includes co-training [121]. All three solutions were originally utilized for semi-supervised learning, but have been extended to unsupervised transfer learning by Chen et al. [117].

Preliminary work investigating self-training ideas in an unsupervised TR scenario was performed by Goswami et al. [23] who propagated initial pseudo-relevance preferences for pairs of documents drawn from related collections. A pairwise ranking function was trained iteratively with a discriminant classification EM algorithm beginning with the pseudo-preference labels. The results from that study suggested significant improvements in some TREC ad-hoc collections with eight term-based features. However, the algorithm was designed for a scenario where multiple source collections were available for selection, and the content of documents was known.

Drawing inspiration from Goswami et al. [23], our algorithms fit into the unsupervised TR scenario where only one source collection is available for transferring (and the source text for each document is not the primary information used to perform the transfer).[1]

The idea of applying self-labeling methods to unsupervised TR was inspired by two branches of prior work: a TR algorithm that infers labels from other collections [23] and pseudo-relevance feedback (PRF) [27]. Self-labeling by imputed relevance labels shares commonalities with PRF in that both algorithms make assumptions about relevance and the initial set. However, PRF is typically utilized for reformulating queries, while label imputation is used to train better ranking models. Moreover, PRF algorithms are usually conducted on a per-query basis, while label imputation is performed on a per-collection basis.

5.3 Problems with Instance Weighting for TR

The core challenge of transfer learning is that the source and target instances are drawn from different distributions. Instance weighting looks to solve a special case of the problem, *covariate shift* [8], where the conditional distribution of the class label remains unchanged across the source and target collections ($p^{so}(y|\mathbf{x}) = p^{ta}(y|\mathbf{x})$), while the input (feature) distribution has changed ($p^{so}(\mathbf{x}) \neq p^{ta}(\mathbf{x})$). Covariate shift can be addressed by

[1]We note that while inspired by their work, the algorithms we develop in this paper are quite different (and in a sense more general) than those of the work of Goswami et al. [23]. Indeed they are not even directly comparable given that they are tackling different problems with different (and in their case more specific) assumptions.

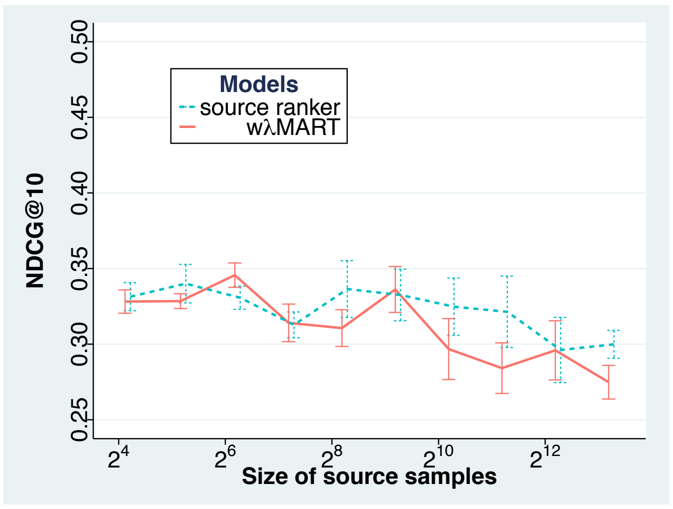


FIGURE 5.1: Effectiveness of wλMART versus source sample size

re-weighting source samples such that the source distribution approximates the target one. However, for listwise L2R algorithms, training is performed at the query level. Consequently, instance weighting is more meaningful and natural at the query level rather than the document level.

Query-level instance weighting attempts to re-weight source queries to approximate the query distribution in the target collection: $w(q)p^{so}(q) \approx p^{ta}(q)\ \forall\ q \in Q^{so}$, where $p^{ta}(q)$ and $p^{so}(q)$ denote the densities over queries in the target and source collection respectively. The rank learner is trained on weighted training data, where the weight for each source query q_i^{so} is set to approximate the density ratio $w(q_i^{so}) = p^{ta}(q_i^{so})/p^{so}(q_i^{so})$. By doing this, the loss function used during training tends to follow the desired loss function on the target collection.

In the previous chapter, we have demonstrated how the effectiveness of different instance-weighting methods varies across transferring settings. In this section, we take a different approach to investigate the reliability of instance-weighting algorithms by controlling the sample sizes of the source collection while keeping all the other settings the same. Figure 5.1 shows the effectiveness of query-weighted LambdaMART (wλMART)[2] based on the Kullback-Leibler Importance Estimation Procedure (KLIEP) [122], measured with NDCG@10, when it was trained with different sizes of source queries pooled from MSLR[3] and tested on LETOR4.0. The settings of the transfer are similar to Li et al. [122] except that the test set is used for density ratio estimation.

[2]The algorithm used in here was the document-level-weight-aggregation version, kliep.doc.
[3]https://www.microsoft.com/en-us/research/project/mslr/

The results in Figure 5.1 show that the effectiveness of the source ranker on the target dataset varies across training samples and degrades with the size of the training sample. More concerning is the fact that the performance of the instance-weighting algorithm is not consistent, but jumps above and below the blue line (representing the source ranker). Notice that we also saw a slight decreasing of performance with the increase of source sample size. As the sample size increases, the ranking function is fitting better for the source training data, which caused the decreased performance in the unseen target collection data.

Thus far, we have seen that the performance of instance weighting can be unreliable. Two factors can be the cause of this issue: the inaccuracy of the density estimation for the queries, or the unrealistic assumption that the mapping from documents to relevance judgments, $p^{so}(r|x) = p^{ta}(r|x)$, remains the same across the collections. Moreover, the fact that in the standard learning-to-rank setup, the learnt ranking function is actually only *re-ranking the top-k documents* (as selected by an initial base ranker) means that even if only covariate shift is present, the *resulting* conditional distribution will likely be different across the source and target collections.

5.4 Expectation-Maximization (EM) for Unsupervised TR

Parameter estimation using the Expectation-Maximization (EM) algorithm has been widely studied for training semi-supervised models when there is an absence of adequate labels [123]. The EM algorithm can potentially be used for solving TR problems because of its ability to leverage unlabeled training data.

The EM algorithm is used to generate maximum likelihood estimates for the parameters of a statistical model via iterations. Given a joint distribution of $p(X, Z|\theta)$ governed by parameters θ, where X are the observed variables, and Z are some hidden or missing values, the EM algorithm attempts to estimate parameters by maximizing the likelihood $p(X|\theta)$ as follows:

1. Initialize parameters $\theta^{(0)}$.
2. E-step: Evaluate $p(Z|X, \theta^{(t-1)}) \propto p(X, Z|\theta^{(t-1)})$.

3. M-step: Evaluate $\theta^{(t)}$ by:

$$\theta^{(t)} = \underset{\theta}{arg\ max} \sum_{Z} p(Z|X, \theta^{(t-1)}) \log\ p(X, Z|\theta) \tag{5.1}$$

4. Repeat steps 2 and 3 until parameters or log likelihood (summation in 3) converges.

5.4.1 EM Algorithm for TR with Pairwise Preferences

In this section, we apply a modified EM algorithm to tackle the TR problem. Assuming the unlabeled target data is drawn from a joint distribution of $p(X, R|\theta)$, governed by some parameters θ. X is a set of observed feature vectors for a document set, and R is their unobserved relevance labels. An EM algorithm estimates the parameters θ by maximizing the likelihood, $p(X, R)$. In the E-step, the EM algorithm computes the probability of each discrete value for individual document, $p(r = 1|\mathbf{x}, \theta)$ and $p(r = 0|\mathbf{x}, \theta)$. We assume the parameters θ to be the parameters of a function mapping a query document pair to a relevance label ($\gamma(\mathbf{x}, \theta) \mapsto r$). This mapping function can be decomposed into two functions, a scoring function which estimates a similarity score for a query document pair, and a (possibly stochastic) assignment function which maps each query-similarity score to a relevance label.

Estimating $p(R|X, \theta)$ requires making strong assumptions about how scores map to relevance levels. We can avoid this issue by instead using the pairwise ranking preferences as the hidden values. The pairwise probability of a document pair $\{d_{ij}, d_{ik}\}$ can be estimated using a logistic function as in Burges et al. [42]

$$p(r_{ij} > r_{ik}) = \frac{1}{1 + e^{-\sigma\ \Delta s_{ijk}}} \tag{5.2}$$

Here σ is a parameter controlling the shape of the logistic function[4], $\Delta s_{ijk} = s_{ij} - s_{ik}$ is the difference between the query-similarity scores for the two documents as predicted by a ranking function.

We propose a pairwise preference-based EM algorithm, called RankPairwiseEM, to tackle the unsupervised TR problem. Here we consider the joint distribution of $p(X^2, \Delta R|\theta)$ over pairs of documents with different relevance labels $X^2 = \{(\mathbf{x}_{ij}, \mathbf{x}_{ik})\}_{i,j<k}$ $s.t.$ $r_{ij} \neq$

[4]Later in the experiments, σ was set to 1, which is the same value used for LambdaMART.

r_{ik}, where ΔR denotes the ranking preferences ($\Delta r_{ijk} = 1$, if $r_{ij} > r_{ik}$; $\Delta r_{ijk} = -1$, if $r_{ij} < r_{ik}$).

In the E-step of EM, the algorithm evaluates the pairwise preference probability based on parameters estimated in the last iteration, $p(Y|\Phi, \theta^{(t-1)})$, and this can be approximated using the probability model:

$$\omega_{ijk}^{(t-1)} = p(r_{ij} > r_{ik}|\theta^{(t-1)}) = \frac{1}{1 + e^{-\sigma\ \Delta s_{ijk}^{(t-1)}}} \tag{5.3}$$

where $\Delta s_{ijk}^{(t-1)} = s_{ij}^{(t-1)} - s_{ik}^{(t-1)}$ is the difference in the document scores $s_{ij} = f(\mathbf{x}_{ij}; \theta^{(t-1)})$.

In the M-step, the estimation of the new parameters is performed by maximizing the expected likelihood based on the probabilities estimated in the E-step. Instead of maximizing the expected likelihood, however, we minimize the expected cost, which depends on the particular rank-learning algorithm being used. In this work, we apply the state-of-the-art L2R algorithm, LambdaMART [53] which has been used for the experiments in chapter 4. The detailed explantion of LambdaMART algorirthm can be found in section 2.2.4.

The LambdaMART algorithm iteratively builds an additive ensemble of regression trees for calculating document scores.

$$f(\mathbf{x}) = \sum_{l=1}^{L} \alpha_l\ h_l(\mathbf{x}) \tag{5.4}$$

At each iteration, the algorithm computes the cost between the ground-truth pairwise probabilities and the probabilities inferred by the current ensemble ($f^{(l-1)}$) using Equation 5.2. The ground truth pairwise probability is modeled as: $P_{ijk} = \frac{1}{2}(1 + \Delta r_{ijk})$. For each pair of documents for the same query, the cost function can be rewritten as:

$$C_{ijk} = |\Delta Z_{ijk}|(I_{[r_{ij} > r_{ik}]} \log(1 + e^{-\sigma\ \Delta s_{ijk}^{(l-1)}}) + I_{[r_{ij} < r_{ik}]} \log(1 + e^{\sigma\ \Delta s_{ijk}^{(l-1)}})) \tag{5.5}$$

where ΔZ_{ijk} is the change of the ranking evaluation score (e.g., NDCG) that results from swapping the position of documents d_{ij} and d_{ik}, while $I_{[.]}$ denotes an indicator function. The cost of an individual document $\mathbf{x}_{ij}$ is then aggregated over the pairs: $C_{ij} = \sum_{k:k\neq j} C_{ijk}$.

A regression tree is then trained to minimize the cost by fitting the derivatives of the cost, denoted λ_{ij}, with respect to the query-similarity score predicted using the current ensemble:

$$\lambda_{ij} = \frac{\partial C_{ij}}{\partial s_{ij}^{(l-1)}} = \sum_{k:k\neq j} |\Delta Z_{ijk}|(I_{[r_{ij}>r_{ik}]}\frac{-\sigma}{1+e^{\sigma\ \Delta s_{ijk}^{(l-1)}}} - I_{[r_{ij}<r_{ik}]}\frac{-\sigma}{1+e^{-\sigma\ \Delta s_{ijk}^{(l-1)}}})) \quad (5.6)$$

According to Burges [53], the value of the k^{th} leaf in the l^{th} tree is then updated using a second-order approximation:

$$\gamma_{km} = \frac{\sum_{d_{ij}\in R_{km}} \frac{\partial C_{ij}}{\partial s_{ij}^{l-1}}}{\sum_{d_{ij}\in R_{km}} \frac{\partial^2 C_{ij}}{\partial (s_{ij}^{l-1})^2}} = \frac{\sum_{d_{ij}\in R_{km}} \lambda_{ij}}{\sum_{d_{ij}\in R_{km}} \frac{\partial \lambda_{ij}}{\partial s_{ij}^{l-1}}} \quad (5.7)$$

Under the unsupervised TR scenario, the ground truth relevance labels are *unknown*, but since we have computed the pairwise probability for all the target document-pairs in the E-step, we can calculate expected costs for target documents:

$$\mathbb{E}[C_{ij}] = \sum_{k:k\neq j} |\Delta Z_{ijk}|(\omega_{ijk}^{(t-1)} \log(1+e^{-\sigma\ \Delta s_{ijk}^{(t,l-1)}}) + \omega_{ikj}^{(t-1)} \log(1+e^{\sigma\ \Delta s_{ijk}^{(t,l-1)}})) \quad (5.8)$$

where ω_{ijk} and ω_{ikj} are probabilities computed using Equation 5.3, and $\Delta s_{ijk}^{(t,l-1)} = s_{ij}^{(t,l-1)} - s_{ik}^{(t,l-1)}$ denotes the difference in the scores computed using the model with $(l-1)$ trees trained for t iterations. The corresponding derivative is:

$$\mathbb{E}[\lambda_{ij}] = \sum_{k:k\neq j} \mathbb{E}[|\Delta Z_{ijk}|](\frac{-\omega_{ijk}\sigma}{1+e^{\sigma\ \Delta s_{ijk}^{(t,l-1)}}} - \frac{-\omega_{ikj}\sigma}{1+e^{-\sigma\ \Delta s_{ijk}^{(t,l-1)}}}) \quad (5.9)$$

In this paper, we use NDCG@10 as the training metric for LambdaMART (i.e. $Z = NDCG@10$). Because the relevance labels, as well as ranking orders of documents, are unknown, we need to compute the expected $|\Delta NDCG@10|$[5] based on parameters trained in the last iteration, $\theta^{(t-1)}$. The query-similarity score predicted with the parameters trained in the last iteration for each document are used as the expected relevance labels: $\mathbb{E}[r_{ij}] \approx s_{ij}^{(t-1)} = f(\mathbf{x}_{ij};\theta^{(t-1)})$.

[5]Replacing ΔZ by the fixed value 1 was also investigated but resulted in poor performance.

$$\mathbb{E}[|\Delta NDCG@10_{ijk}|] = \frac{2^{\mathbb{E}[r_{ik}]} - 2^{\mathbb{E}[r_{ij}]}}{IDCG} \times (\frac{1}{\log_2(\pi_{ij}^{(t,l-1)}+1)} - \frac{1}{\log_2(\pi_{ik}^{(t,l-1)}+1)}) \quad (5.10)$$

where $\pi_{ij}^{(t,l-1)}$ denotes the rank of the j^{th} document for query i, according to the scoring function $f(\mathbf{x}_{ij};\theta^{(t,l-1)})$. The ground-truth labels for the documents for the queries are unknown, we use the similarity score predicted in the last iteration as the label for estimating IDCG. As a result, IDCG is calculated as:

$$IDCG = \sum_{g=1}^{10} \frac{2^{s^{(t-1)}_{i\pi^{-1}(g)}} - 1}{\log_2(g+1)} \quad (5.11)$$

where $s^{(t-1)}_{i\pi^{-1}(g)}$ is the score of the document ranked at g^{th} position of query i, with the ranking function $f^{(t-1)}$.

The expected lambdas $\mathbb{E}[\lambda]$ are then used to fit the regression trees. The expected value for each leaf is updated as:

$$\mathbb{E}[\gamma_{km}] = \frac{\sum_{d_{ij}\in R_{km}} \mathbb{E}[\lambda_{ij}]}{\sum_{d_{ij}\in R_{km}} \frac{\partial \mathbb{E}[\lambda_{ij}]}{\partial s_{ij}^{(t,l-1)}}} \quad (5.12)$$

The parameters will be updated after the ensemble has been trained, with the process repeated until convergence.

The implementation of the EM algorithm for TR (RankPairwiseEM) is presented in Algorithm 2. The parameters are initialized by training a LambdaMART with source data:

$$\hat{\theta}^{(0)} = \arg\min_{\theta} \sum_{q_i \in Q^{so}} \sum_{d_{ij}\in q_i} C_{ij} \quad (5.13)$$

In the E-step, each document is assigned a similarity score predicted by the ranking function, with parameters trained in the last iteration. The pairwise preference probability of document pairs is then computed using Equation 5.3. In the M-step, the parameters are re-estimated with the expected LambdaMART together with the labeled source data:

$$\hat{\theta}^{(t+1)} = arg\ min_{\theta} \sum_{q_i \in Q^{so}} \sum_{d_{ij}\in q_i} C_{ij} + \sum_{q_i \in Q^{ta}} \sum_{d_{ij}\in q_i} \mathbb{E}[C_{ij}] \quad (5.14)$$

Algorithm 2: Label-imputation via RankPairwiseEM

```
Input: Source queries $Q^{so}$ and judgements $R^{so}$, target queries $Q^{ta}$, max iterations $\Gamma$, $\tau$
       threshold $\epsilon$
Output: Ranking function $f$
RankPairwiseEM($Q^{so}$,$R^{so}$,$Q^{ta}$,$\Gamma$)
    Train ranker $f^{(0)}$ using ($Q^{so}$,$R^{so}$) with Eq. 5.13;
    for $t \in \{1, ..., \Gamma\}$ do
        /* E-step                                                      */
        foreach $\boldsymbol{x}_{ij} \in Q^{ta}$ do
            $s_{ij} = f(\mathbf{x}_{ij}; \theta^{(t-1)})$
        end
        foreach $\{\boldsymbol{x}_{ij}, \boldsymbol{x}_{ik}\} \in Q^{ta}$ do
            Estimate $p(r_{ij} > r_{ik})$ using Eq. 5.3;
        end
        /* M-step                                                      */
        Train $f(\mathbf{x}; \theta^{(t)})$ using pairwise probs, Eq. 5.14;
        if $\theta^{(t)} == \theta^{(t-1)}$ then
            return $f^{(t-1)}$;
        end
    end
    return $f^{(t)}$;
```

The algorithm repeats the E-step and M-step until the parameters converge or the maximum iteration Γ is met.

5.4.2 EM for TR with "Hard" Assignment

It has been shown that in certain situations an EM algorithm with hard deterministic label assignment can be more efficient and more effective than the original EM algorithm for particular tasks [124]. This so-called **hard EM** algorithm is a variant of the original EM algorithm, which assigns the best possible label to each training instance at the E-step, rather than computing the probability of each label. In the M-step, the hard EM algorithm updates the parameters using the updated labels.

To employ the hard EM algorithm for unsupervised TR, one needs to determine the most likely label for each unlabeled document in the target collection according to the current model. Here we only consider the binary relevance case and simply label documents with highest similarity scores as relevant. Intuitively, allocating the relevant labels to a smaller fraction of top-ranked documents will preserve more accuracy since on those top documents the ranker is most confidential, and tends to be better for model transferring.

Algorithm 3: Self-labeling via RankHardLabelEM

```
Input: Source queries $Q^{so}$ and judgements $R^{so}$, target queries $Q^{ta}$, stopping threshold
       $\epsilon$, max iteration $\Gamma$
Output: Ranking function $f$
RankHardLabelEM($Q^{so}$,$R^{so}$,$Q^{ta}$,$\epsilon$,$\Gamma$)
    Train ranker $f^{(0)}$ using ($Q^{so}$,$R^{so}$) with Eq. 5.13;
    for $t \in \{1, ..., \Gamma\}$ do
        /* E-step */
        Calculate scores for all query-doc pairs;
        Sort query-doc pairs by decreasing score;
        Label top $k\%$ as relevant, remainder irrelevant;
        /* M-step */
        Train $f(\mathbf{x}; \theta^{(t)})$ using Eq. 5.15;
        if $\theta^{(t)} == \theta^{(t-1)}$ then
            return $f^{(t-1)}$;
        end
    end
    return $f^{(t)}$;
```

In this work, only the top k percent documents with the highest ranker score will be labeled as relevant documents.

In the M-step, the ranking function will be updated by training using both the labeled source data and unlabeled target data together with the imputed relevance labels:

$$\hat{\theta}^{(t+1)} = \underset{\theta}{arg\ min} \sum_{q_i \in Q^{so}} \sum_{d_{ij} \in q_i} C_{ij} + \sum_{q_i \in Q^{ta}} \sum_{d_{ij} \in q_i} \hat{C}_{ij}(\hat{R}^{(t)}) \tag{5.15}$$

where $\hat{C}_{ij}(\hat{R}^{(t)})$ is computed with the imputed relevance labels, $\hat{R}^{(t)}$=$\{[s_{ij}^{(t)} \geq sort(\{s_{ij}^{(t)}\}_j)_k]\}_i$, generated at $(t+1)^{(th)}$ iteration according to the query-similarity scores predicted using ranker function trained at $t^{(th)}$ iteration.

With the updated ranker, the system can update the imputed labels iteratively.

The RankHardLabelEM algorithm is demonstrated in Algorithm 3. The algorithm first trains a source ranker with the labeled query-document pairs from the source collection together. In the E-step, the algorithm will compute the similarity scores for all query-document pairs and label the top $k\%$ pairs as relevant documents and the remainder as irrelevant. In the M-step, using labeled source data and the target data together with their imputed labels for training, the ranking function will be updated. The process

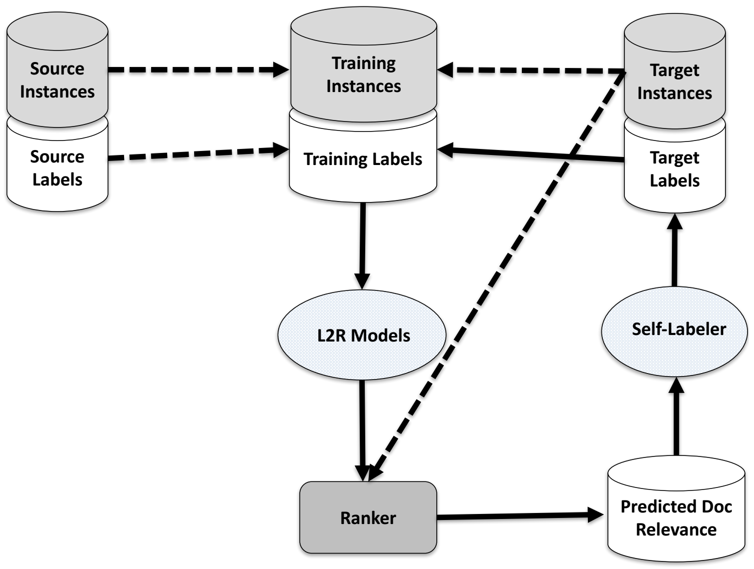


FIGURE 5.2: RankHardLabelEM & self-labeling paradigm

runs iteratively until the imputed label stops changing or the maximum iteration count is reached.

5.5 Self-Training for Unsupervised TR

A third self-labeling method for unsupervised TR is based on self-training: a form of semi-supervised learning [125, 126], with applications in natural language processing [120, 126] and transfer learning [117]. Self-training algorithms are similar to RankHardLabelEM except that instead of recalculating all of the predicted labels on each iteration, the predicted positive (i.e., relevant) documents are preserved from the previous iteration. In each subsequent iteration, the algorithms simply adds next documents to the relevant set on which it is most confident.

So the self-training algorithm (RankSelfTrain) gradually increases the number of imputed relevant documents via an iterative process. Both RankHardLabelEM and RankSelfTrain follows the self-labeling paradigm demonstrated in Figure 5.2. The system will initialize a ranking function by the source instances with their source labels using a particular L2R model. With the trained ranker, the system predicts relevance scores for all the unlabeled training instances in the target collection, and then uses a *Self-Labeler* to impute labels for all the unlabeled target instances. With the newly updated labels, the algorithm updates the ranker and conducts the self-labeling again. The process is run iteratively until convergence.

Unlike RankHardLabelEM algorithm, which updates imputed labels iteratively, the RankSelfTrain gradually adds confident labels to the training set. By gradually adding a small number of likely accurate predictions, it is hoped the self-trained ranker can update itself toward a ranking function that can generalize to the target collection. As a result, for RankSelfTrain algorithm, once a document has been added to imputed relevant set, the label will not change in the next iteration.

A confidence score is needed to allow label prediction. It is possible for some classification algorithms to produce such scores; for example, logistic regression can output a probability for a class label. However, it is not straightforward for ranking algorithms to produce such probabilities[6]. Thus we develop a methodology to predict the probability of a document being relevant or irrelevant, provided with their similarity scores predicted by a ranking function. The probability of relevance and irrelevance can later be used as the confidence in the predicted label.

Bayes rule for the probability of a document being relevant, given a similarity score gives:

$$p(r=1|s=\alpha) = \frac{p(r=1)p(s=\alpha|r=1)}{\sum_{v\in\{0,1\}} p(s=\alpha|r=v)p(r=v)} \tag{5.16}$$

where s denotes the score predicted by a ranking function. The densities $p(s=\alpha|r=1)$ and $p(s=\alpha|r=0)$ can be estimated via the kernel density estimation (KDE) [127] on a collection. The algorithm samples all the predicted scores for relevant documents and use KDE to measure the density of $p(s=\alpha|r=1)$. The same measurement is applied to the irrelevant documents. This approximates the densities of the scores in the distribution. The prior probability $p(r=1)$ is estimated by the proportion of relevant documents in the collection:

$$p(r=1) = \frac{|relevant\ documents|}{|documents|} \tag{5.17}$$

Initially, the target collection contains no imputed relevant documents so the probabilities can only be estimated using data from the source collection. As the relevance

[6]RankSVM [38] and other pairwise L2R algorithms might be able to output a probability for ranking preferences; however, the probabilities for preferences will not directly infer the labels of a document.

labels in some source collections are multi-graded, we regard all the documents whose relevance labels are larger than zero as relevant. In the following iterations, as some imputed labels have been generated, the conditional probability can be estimated on the target data together with the imputed labels:

$$p^{ta}(s = \alpha|r = 1) \approx p^{ta}(s = \alpha|\hat{r} = 1) \tag{5.18}$$

$$p^{ta}(s = \alpha|r = 0) \approx p^{ta}(s = \alpha|\hat{r} = 0) \tag{5.19}$$

where $\hat{r}$ denote imputed labels.

Since the imputed labels are gradually added to the imputed set, directly estimating the prior probability $p(r = 1)$ with the imputed labels will be unreliable. At the same time, the prior probability of the target collection can be different from the source collection. Thus we propose a Dirichlet smoothed estimation which can balance the impact of the source and the imputed labels from the target adaptively:

$$\begin{aligned} p^{ta}(r = 1) &\approx \frac{\sum_i \mathbb{I}(\hat{r} = 1) + \mu p^{so}(r = 1)}{|\hat{r}| + \mu} \\ p^{ta}(r = 0) &\approx \frac{\sum_i \mathbb{I}(\hat{r} = 0) + \mu(1 - p^{so}(r = 1))}{|\hat{r}| + \mu} \end{aligned} \tag{5.20}$$

where μ is set to be half of the number of training instances in the target collection. μ was applied to the prior probability for the source collection, the smoothing function was trying to reduce the importance of the source collection. As a result, we choose to use half, rather than the entire number of training instances as what normally is done in Dirichlet smoothing. As a result, probability can be estimated:

$$p^{ta}(r = 1|s = \alpha) = \frac{p^{ta}(r = 1)p^{ta}(s = \alpha|\hat{r} = 1)}{\sum_{v\in\{0,1\}} p^{ta}(s = \alpha|\hat{r} = v)p^{ta}(r = v)} \tag{5.21}$$

The process of the RankSelfTrain algorithm is shown in Algorithm 4. Initially, a source ranker f^0 is trained with labeled examples (Q^{so}, R^{so}) from the source collection. The source ranker is then applied to calculate similarity scores for all the query-document pairs in the target collection (line 4). In the first iteration, the algorithm calculates the relevance probability for each query-document pair via Equation 5.16 with probabilities in the source data. If the probability of a relevance label for a given pair is larger than the threshold η, which is a confidence threshold, the query-document pair will be added

Algorithm 4: SELF-TRAINING FOR RANKING

```
Input: Source queries $Q^{so}$ and judgements $R^{so}$, target queries $Q^{ta}$, confidence
       threshold $\eta$
Output: Ranking function $f$
SelfTrain($Q^{so}$,$R^{so}$,$Q^{ta}$,$\eta$)
    Initialize set of labeled docs to be empty: $\Omega^{(0)} = \emptyset$;
    Train ranker $f^{(0)}$ using $(Q^{so}, R^{so})$ with Eq. 5.13;
    for $t \in \{1, ...\}$ do
        Calculate similarities for all query-doc pairs;
        foreach unlabeled pair $\mathbf{x}_{ij} \notin \Omega^{(t-1)}$ do
            if $t{=}{=}1$ then
                Compute $p(r_{ij}|s_{ij})$ following Eq. 5.16;
            else
                Compute $p(r_{ij}|s_{ij})$ following Eq. 5.21;
            end
            if $p(r_{ij} = 1|s_{ij}) > \eta$ then
                Add $(\mathbf{x}_{ij}, 1)$ to $\Omega^{(t)}$;
            else if $p(r_{ij} = 0|s_{ij}) > \eta$ then
                Add $(\mathbf{x}_{ij}, 0)$ to $\Omega^{(t)}$;
        end
        if $(|\Omega^{(t)}| - |\Omega^{(t-1)}|) == 0$ then
            return $f^{(t-1)}$;
        end
        Train ranker $f^{(t)}$ using Eq. 5.15;
    end
```

to the labeled document set. The confidence threshold η will be set at a higher number to ensure the accuracy of the label imputation process. The system will then re-train a ranking function with both the data from the source collection and previously labeled documents from the target collection using Equation 5.15. In the following iterations, the algorithm will continue to compute the probabilities via the imputed labels from the target collection using Equation 5.21, conduct the labeling and update the ranker iteratively until no more confident labels can be added or the maximum iteration is met.

5.6 Data and Methods

5.6.1 Datasets

Three public L2R test collections used for the instance weighting algorithm from the last chapter are used in our experiments: MSLR, LETOR4.0, and the Yahoo! Learning to Rank (Yahoo!L2R) dataset.[7]

Similar to the previous chapter, three groups of transfer settings are studied:

1. Transferring between MQ2007 and MQ2008, which share the same document collection but have different query sets. Since the two datasets differ only on the queries, this can be viewed as an *in-domain* transfer.
2. Transferring between MSLR and LETOR 4.0: We merged the two datasets in LETOR 4.0 to make a larger dataset and then conducted the transfer between the merged LETOR 4.0 dataset and MSLR-WEB10K. The two datasets have few commonalities, with different document sets, query sets, and methods for gathering relevance. Thus transferring here can be viewed as a *cross-domain* transfer. In the experiments the 45 features common to both collections were used to train the L2R models.[8]
3. Transferring between Set 1 and Set 2 of Yahoo!L2R: each set represents web documents written in different regional languages, thus transferring between the two is also *cross-domain* transfer. The original Yahoo!L2R collection has 700 features. However, we found that only 415 were common to both sets, and utilized them in the experiments.

One dataset from each pairing was taken to be the *source* collection, and the other to be the *target*. Each target collection was split randomly into five folds for cross-validation-based evaluation. In each experimental run, four folds were utilized as examples for the target collection. To create an unsupervised TR environment, all relevance labels were removed from these folds. The remaining fold of the target collection was used to test the effectiveness of the transfer algorithms. We note that this setup, in which the target queries used during the transfer were not used for the evaluation, was particularly challenging. The details of the transfer settings are provided in Table 4.1. All reported results are averages over the five-fold cross-validation.

[7]Details of these collections are presented in Table 2.1.

[8]The features in LETOR 4.0 were normalized via a query-level normalization method [128] (min-max normalization on a per-query basis) and we conducted normalization for the MSLR collection as well. It turned out that conducting feature normalization, in the same way, can lead to a better generalization for another collection.

5.6.2 Setup and Measurements

The RankLib 2.1 implementation of LambdaMART was used as the base ranker.[9] The tree size was set to 1000, and the maximum number of leaves was set to 10. For the instance-weighting-based KLIEP method, we applied Sugiyama-Sato's Matlab implementation.[10]

For all the algorithms, we set the maximum iteration Γ as 20. The percentage of imputed relevance labels $k\%$ was set to 5% for the RankHardLabelEM algorithm. For the RankSelfTrain algorithm, the threshold of confidence η was set at 95%. The σ for pair-wise probability in Equation 5.2 was set to 1 in the RankPairwiseEM algorithm, which is aligned with the value of σ in the implementation of LambdaMART.

The following baselines were considered:

- **BM25:** Retrieved documents sorted by decreasing BM25 similarity score.
- **λMART.source:** LambdaMART trained with all the data from the source collection.
- **wλMART:** Weighted LambdaMART with the query-level instance weighting method proposed by Li et al. [122]. We used the "kliep.doc" method proposed in the paper, which aggregated the document-level weights for generating query-level weights. The document-level weights are estimated via the KLIEP algorithm [63].
- **λMART.target:** LambdaMART trained with data from the target collection via cross-validation.

The following label imputation algorithms were tested:

- **RankPairwiseEM:** EM-inspired self-labeling algorithm, using LambdaMART as the base ranker.
- **RankHardLabelEM:** "Hard EM"-inspired self-labeling algorithm, using LambdaMART as the base ranker.

[9]http://sourceforge.net/p/lemur/wiki/RankLib/
[10]http://www.ms.k.u-tokyo.ac.jp/software.html

- **RankSelfTrain:** Self-training-based algorithm, using LambdaMART as the base ranker.

All models were evaluated using normalized discounted cumulative gain (NDCG) [106], with a rank cut-off of 10. Statistical significance was tested using a two-tailed paired t-test, with a threshold of 0.05.

5.7 Results and Discussion

The experimental results are presented and discussed below.

5.7.1 Effectiveness of Self-Labeling Methods

We compare the three proposed self-labeling-based TR algorithms on various transfer settings. The most important aspect for distinguishing between the different transfer settings is the level of similarity between the source and target collections, which we consider in two cases impacts the effectiveness of various TR algorithms. *In-domain transfer* where the source and target were drawn from the same or similar distributions, and *cross-domain transfer* where the source and target data were drawn from quite different distributions.

The results of various algorithms on both in-domain and cross-domain transfer scenarios are illustrated in Table 5.1 and 5.2. In both cases, we observe that when a ranking function trained on the source data is applied to the target collection, it retains the advantage over the base ranker, BM25 (second row of both tables).

In-domain transfers. As mentioned before, the MQ2007 and MQ2008 are two query sets using the same document collection. Results demonstrate that λMART.source trained with the larger query set of MQ2007 generalizes well to the smaller set of MQ2008. λMART.source of MQ2007 is significantly better than λMART.target trained on the MQ2008 datasets. Conversely, λMART.source trained on MQ2008 is not as effective as λMART.target trained on MQ2008.

In this in-domain transfer scenario, all the unsupervised TR algorithms performed better, although not always significantly, than the source ranker. When transferring from the

Table 5.1: Effectiveness (NDCG@10 score) on in-domain transfer settings with label imputation methods. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than λMART.source, ↓ denotes the figure is significantly worse than λMART.source, † denotes the figure is significantly better than wλMART. $p < 0.05$

	MQ2007-MQ2008	MQ2008-MQ2007
BM25	*0.335* (-32.7%) ↓	*0.249* (-39.6%) ↓
λMART.source	0.498	0.412
wλMART	0.498	*0.384* (-6.8%) ↓
RankPairwiseEM	**0.507** (+1.8%) ↑†	0.434 (+5.3%) ↑†
RankHardLabelEM	0.501	0.426 (+3.4%) ↑†
RankSelfTrain	0.505 †	0.438 (+6.3%) ↑†
λMART.target	0.487 (-2.2%) ↓	**0.445 (+8%)** ↑†

larger sample, MQ2007 to the smaller sample, MQ2008, most of the unsupervised TR methods, including wλMART, did not show significant improvements, except for the RankPairwiseEM algorithm. In this particular transferring setting, the source data has a wider coverage of queries from the same distribution, which turned out to generate a more general ranking function that performs better than the target model (i.e., the model trained directly on the target data). The new transfer methods can further improve the effectiveness over the source ranker.

When the source collection has a smaller size (MQ2008-MQ2007), the generalization of the source ranker becomes so poor that it is not comparable with the target model. All the new proposed methods have shown to be significantly more effective than the source ranker on the target collection. Meanwhile, the previous instance-based transfer model, wλMART, has shown to be significantly worse than the source ranker. Transferring from MQ2008 to MQ2007 can be thought of as a special case of semi-supervised learning. The results in LETOR4.0 showed that self-labeling-based methods can help improve ranking effectiveness under the semi-supervised L2R/in-domain transfer setting.

Cross-domain transfers Transferring between MSLR and LETOR4.0 is the first cross-domain transfer scenario. As explained previously, conducting query-level feature normalization for both the source and target collection helps increase the generalization performance of LambdaMART over the target collection. In contrast to the results

Table 5.2: Effectiveness (NDCG@10 score) on cross-domain transfer settings with label imputation methods. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than λMART.source, ↓ denotes the figure is significantly worse than λMART.source, † denotes the figure is significantly better than wλMART. $p < 0.05$

	MSLR-LETOR4.0	LETOR4.0-MSLR	Yahoo.Set 1-Yahoo.Set 2	Yahoo.Set 2-Yahoo.Set 1
BM25	0.276 (-29.8%) ↓	0.180 (-7.2%) ↓	0.540 (-5.3%) ↓	0.507 (-27.6%) ↓
λMART.source	0.393	0.194	0.723	0.700
wλMART	0.367 (-6.6%) ↓	0.147 (-24.2%) ↓	0.712 (-1.5%) ↓	0.703 (+0.4%) ↑
RankPairwiseEM	0.402 (2.3%) ↑†	0.193 †	0.734 (+1.5%) ↑†	0.709 (+1.3%) ↑†
RankHardLabelEM	389 †	0.202 (+4.1%) ↑†	0.731 (+1.1%) ↑†	0.707 (+1%) ↑
RankSelfTrain	0.410 (+1.8%) ↑†	0.194 †	0.725 (+0.3%) ↑†	0.708 (+1.1%) ↑†
λMART.target	**0.461 (+17.3%) ↑†**	**0.423 (+11.8%) ↑†**	**0.761 (+5.3%) ↑†**	**0.743 (+6.1%) ↑†**

obtained by Li et al. [122], when transferring between MSLR and LETOR4.0, via query-level feature normalization, λMART.source shows better generalization on the target collection.

When transferring from MSLR to LETOR4.0, both RankPairwiseEM and RankSelfTrain significantly outperform λMART.source. All the proposed self-labeling algorithms have shown significant improvements over wλMART.

Transferring from LETOR4.0 to MSLR is harder than transferring in the opposite direction, as MSLR has a wider coverage of queries. wλMART failed to improve the performance of λMART.source. Moreover, both RankPairwiseEM and RankSelfTrain showed no significant improvement in this transfer setting. The RankHardLabelEM algorithm can significantly improve the effectiveness over λMART.source, and it is also significantly more effective than wλMART. Transfer learning from LETOR4.0 is a scenario that is unlikely to occur in reality as the source collection is too small for effective transfer to be possible.

Transferring between Yahoo!L2R Set 1 and Set 2 is a harder task because of the cross-language setting, and because Set 1 has a larger query set. When transferring from Set 1 to Set 2, the effectiveness of the all the proposed algorithms show significant improvements when compared with the λMART.source and the instance-weighting method wλMART. When transferring from the small set to the larger set (Set 2 to Set 1), all the algorithms can significantly outperform λMART.source.

Under the cross-domain transferring scenario, most of the new algorithms have shown some improvements over the source ranker. However, improvements can be varied under different test environments.

5.7.2 Consistency of Unsupervised TR Approaches

In this section, we compare the consistency of different algorithms across different settings. Although all the proposed algorithms showed better transfer effectiveness compared with the source ranker, it is not clear how consistent the performance is.

We compare the effectiveness of unsupervised TR algorithms using average-rank-based visualization [122]. The average rank of all the systems over all the folds in the different collections is computed, and shown in Figure 5.3. The average rank of a system across the test collections is calculated as $\overline{rank}_j = \frac{1}{N}\sum_i rank_{ij}$, where N is the number of collections, and $rank_{ij}$ is the rank of the j^{th} model in the i^{th} collection. We applied the Nemenyi test of significance [115].

The differences between models are compared against the critical distance (CD), i.e., two models are not considered significantly different if their average ranks lie within the CD. The results of the tests are displayed in Figure 5.3. The black dots show the average rank of each model, and the lines show the CD. If the average rank (dot) of a model lies outside the CD of another model, then they are significantly different.

According to Figure 5.3, under current settings, the average rank of all the proposed methods are lower (better) than the λMART.source. Among them, both RankPairwiseEM and RankSelfTrain are significantly better than the λMART.source across different collections, and they showed no significant difference from λMART.target. RankSelfTrain is also the most effective algorithm compared to all the other self-labeling methods.

Interestingly, wλMART appears less effective than the λMART.source, which disagrees with the previous chapter. The reason for this is that by performing query-level feature normalization on the MSLR dataset, the difference between the feature distributions has been reduced. As a result, MSLR showed better generalization on the LETOR4.0 dataset, and the instance-weighting methods failed to show their advantage in minimizing the gap between feature distributions. The query-level feature normalisation has

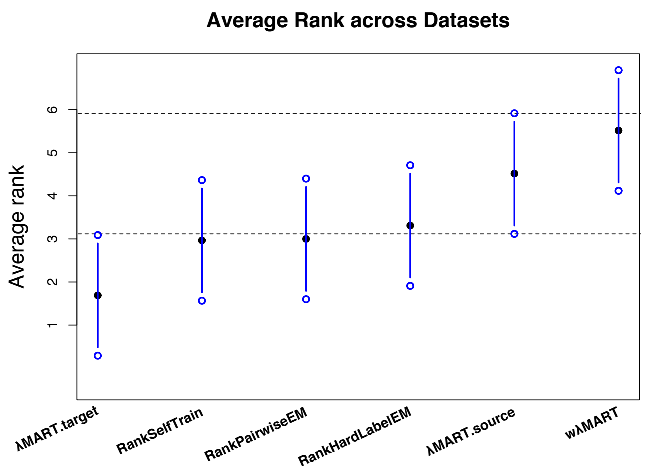


FIGURE 5.3: Plots of average rank across the six test environments for the six different transfer learning techniques and the λMART.source and baseline λMART.target system (where no TR was applied). The lower the rank the better performance of the approach. The critical distance (CD) for the Nemenyi test (at the 5% confidence level).

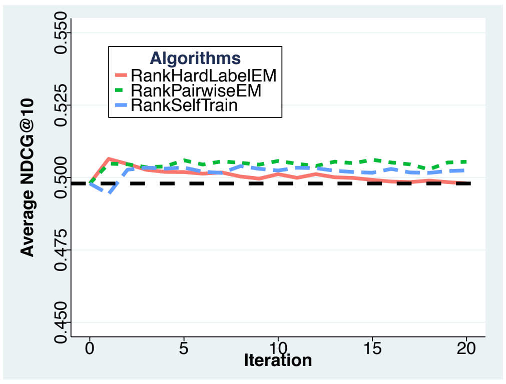
been applied to all datasets, and has shown relatively better generalisation abilities in most cases.

5.7.3 Analysis of Self-Labeling Methods

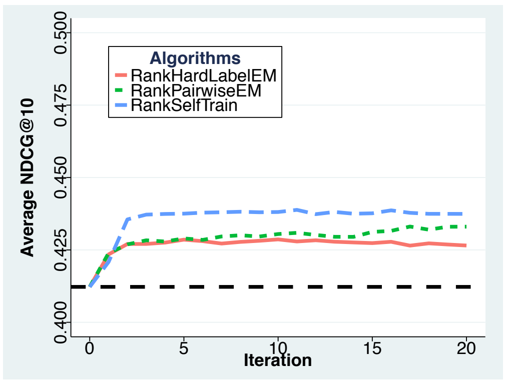
To gain a better understanding of different self-labeling-based approaches, the performance over iterations of the algorithms over the iterations of three proposed methods are illustrated in Figure 5.4. The learning curves presented are averaged over the five runs.

The x-axis in the figure represents the number of the iterations, starting from the 0^{th} iteration (where the source ranker was applied). The y-axis is the average performance of the rankers tested on the target training set, which is the unlabeled target set used for training, together with their ground-truth labels. The black dashed line in the figures shows the performance of the source ranker.

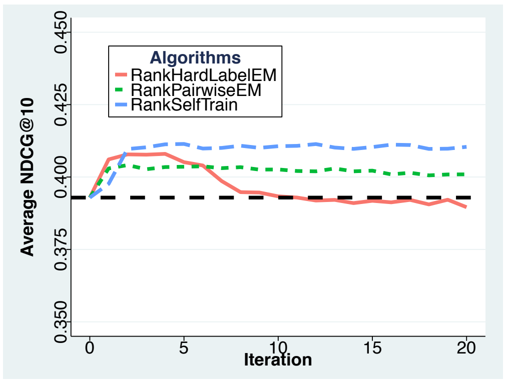
An ideal self-labeling algorithm would gradually increase its effectiveness on the target collection until the imputed labels converge. In most of the transferring settings, we have observed that both RankPairwiseEM and RankSelfTrain gradually update themselves to gain better effectiveness in the target collection. RankHardLabelEM, on the other hand, does not appear to be stable across all different transfer scenarios (collections).



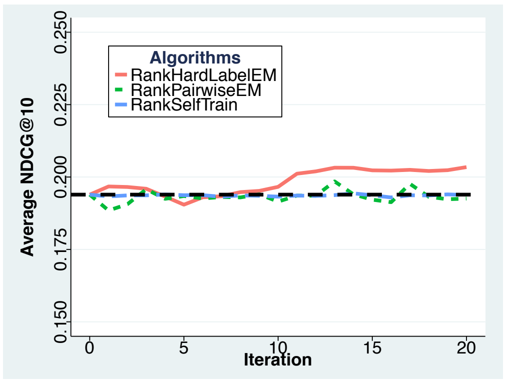
(a) LETOR4.0 MQ2007 to MQ2008



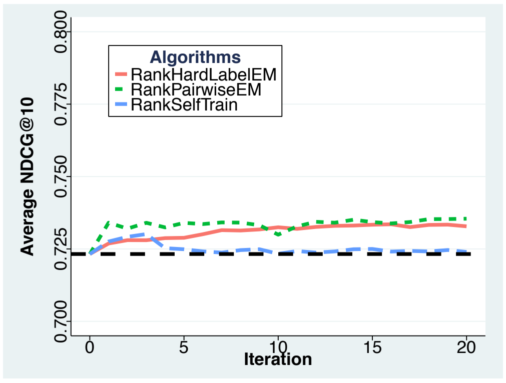
(b) LETOR4.0 MQ2008 to MQ2007



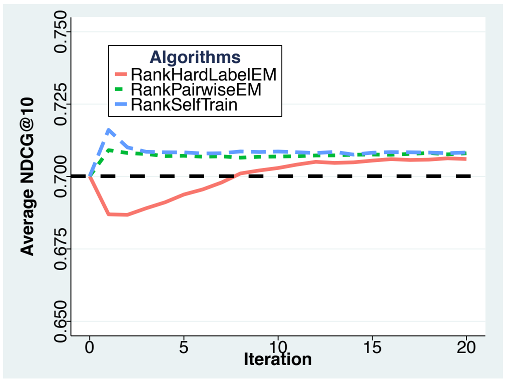
(c) MSLR to LETOR4.0



(d) LETOR4.0 to MSLR



(e) Yahoo Set 1 to Set 2



(f) Yahoo Set 2 to Set 1

FIGURE 5.4: Performance vs iteration curve of different self-labeling methods under various settings.

When transferring from LETOR4.0 to MSLR, none of the algorithms have performed as expected. We argue this is a challenging transferring scenario where there is a much smaller query coverage in the source collection, and the TR algorithm cannot transfer knowledge from the source to the target.

The performance of different algorithms is limited by the parameter selection. In the following section, the impact of the parameters on the performance of the algorithms will be analyzed.

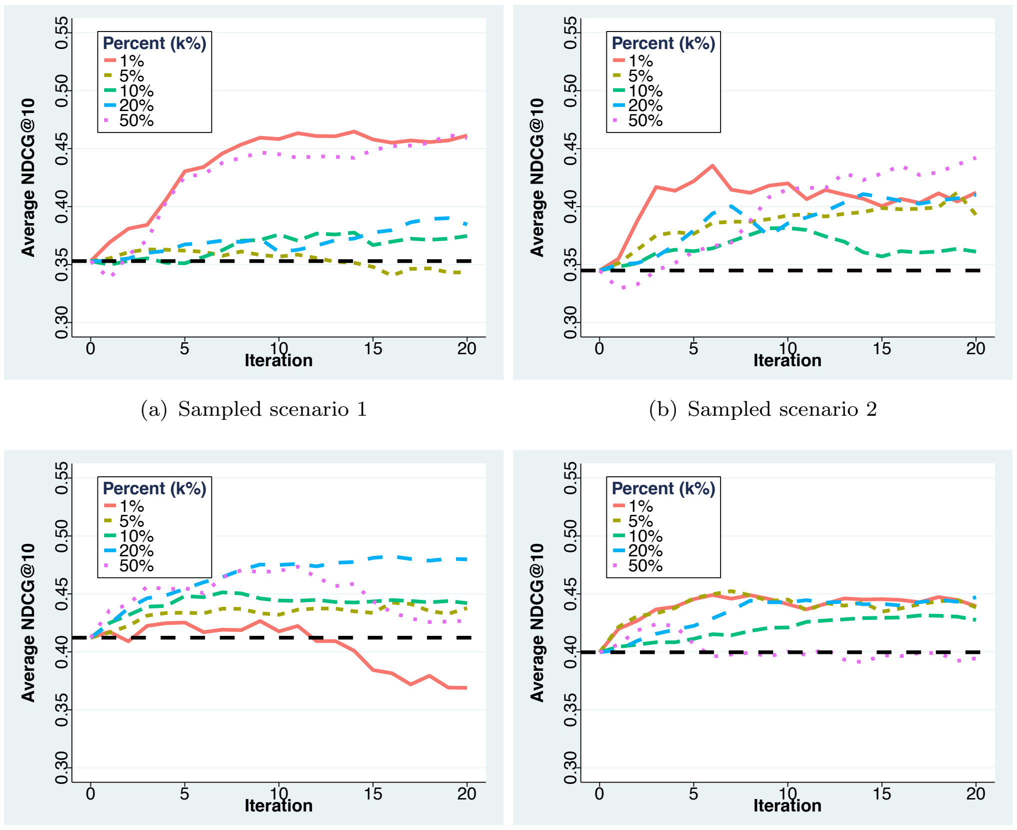
5.8 Sensitivity of Parameter Settings

The sensitivity of the parameter settings for different transfer algorithms will be discussed in this section. The RankPairwiseEM algorithm does not require any parameter settings while the RankHardLabelEM algorithm has a parameter k, which is the percentage of imputed relevant labels in each iteration. For RankSelfTrain algorithm, the percentage is controlled by a confidence score. Alternatively, the percentage can be set manually as it is for the RankHardLabelEM, both the manual setting and confidence score based methods will be compared in the following section.

5.8.1 Threshold Setting for RankHardLabelEM

In the RankHardLabelEM algorithm, the percentage of documents being labeled as a relevant document is manually defined. In this section, we compare the performance of the RankHardLabelEM algorithm with different parameter settings. As the source collection, we randomly sample 1,000 queries from the MSLR dataset; as the target collection, we sample 1,000 queries from the LETOR4.0 dataset. As a result, the target collection contains approximately 34k query-document pairs and the source collection contains nearly 120k query-document pairs. The RankSelfTrain algorithm with different settings for $k\%$ is evaluated for four times. The performance vs iteration curve for each of the four scenarios is shown in Figure 5.5.

The x-axis in Figure 5.5 is the number of iterations, the y-axis is the NDCG@10 scores measured on the unlabeled target set, the black dashed lines are the source rankers. In most cases, the effectiveness of the trained rankers is observed to increase over the iterations, but the increase is not monotonic. In some cases, RankHardLabelEM achieves more than 30% improvement over the source ranker. However, the algorithm performs variously at different runs with a different setting of $k\%$. For example, when $k\%$ was set to 1%, its performance increased gradually over the iterations at the first run (Figure 5.5(a)), while in the other cases, the performance kept dropping (Figure 5.5(c)), indicating a significant amount of variance in performance. Moreover, in some cases, we have



(a) Sampled scenario 1
(b) Sampled scenario 2
(c) Sampled scenario 3
(d) Sampled scenario 4

FIGURE 5.5: Comparing the parameter settings for RankHardLabelEM.

seen that the performance of the algorithm will start to decrease after a certain point (50% in Figure 5.5(c)), so it is also important to determine when to stop the iterations.

Under the unsupervised TR scenario, it is hard to determine the parameters without any supervised label information from the target collection. As a result, a smaller percentage was chosen based on previous experience in IR collections.

5.8.2 Confidence Versus Fixed Increments for RankSelfTrain

In the RankSelfTrain algorithm, we have determined to set a confidence threshold for the label prediction so that only the more confident labels are used (as impute labels) in the next iteration. Alternatively, at each iteration of the RankSelfTrain algorithm, one could label a fixed percentage ($\Delta k\%$) of unlabeled pairs as relevant, and leave the remaining pairs unlabeled as irrelevant. The top $\Delta k\%$ version RankSelfTrain is shown in Algorithm 5. The main difference between the fixed-increments-based RankSelfTrain

Algorithm 5: RankSelfTrain with top Δ percentage

```
Input: Source queries $Q^{so}$ and judgements $R^{so}$, target queries $Q^{ta}$, maximum number
       of iterations
Output: Ranking function $f$
1  SelfTrain($Q^{so}$,$R^{so}$,$Q^{ta}$, $\Gamma$ )
2      Initialize set of relevant docs to be empty: $\Omega^{(0)} = \emptyset$;
3      Initialize set of irrelevant docs to be empty: $\mho^{(0)} = \emptyset$;
4      Train ranker $f^{(0)}$ using $(Q^{so}, R^{so})$ with Eq. 5.13;
5      for $t \in \{1, ..., \Gamma\}$ do
           /* E-step                                                    */
6          Calculate scores for all query-doc pairs;
7          Sort unlabeled pairs $(i, j) \notin \Omega^{(t-1)}$ by score;
8          Label top $\Delta k\%$ pairs as newly relevant: $\Omega^{(t)} = \Omega^{(t-1)} \cup \{topk\}$;
9          Set remaining query-doc pairs as irrelevant: $\mho^{(t)} = X^{ta} - \Omega^{(t)}$;
           /* M-step                                                    */
10         Train ranker $f^{(t)}$ using Eq. 5.15;
11     end
12     Return $f^{(t)}$;
```

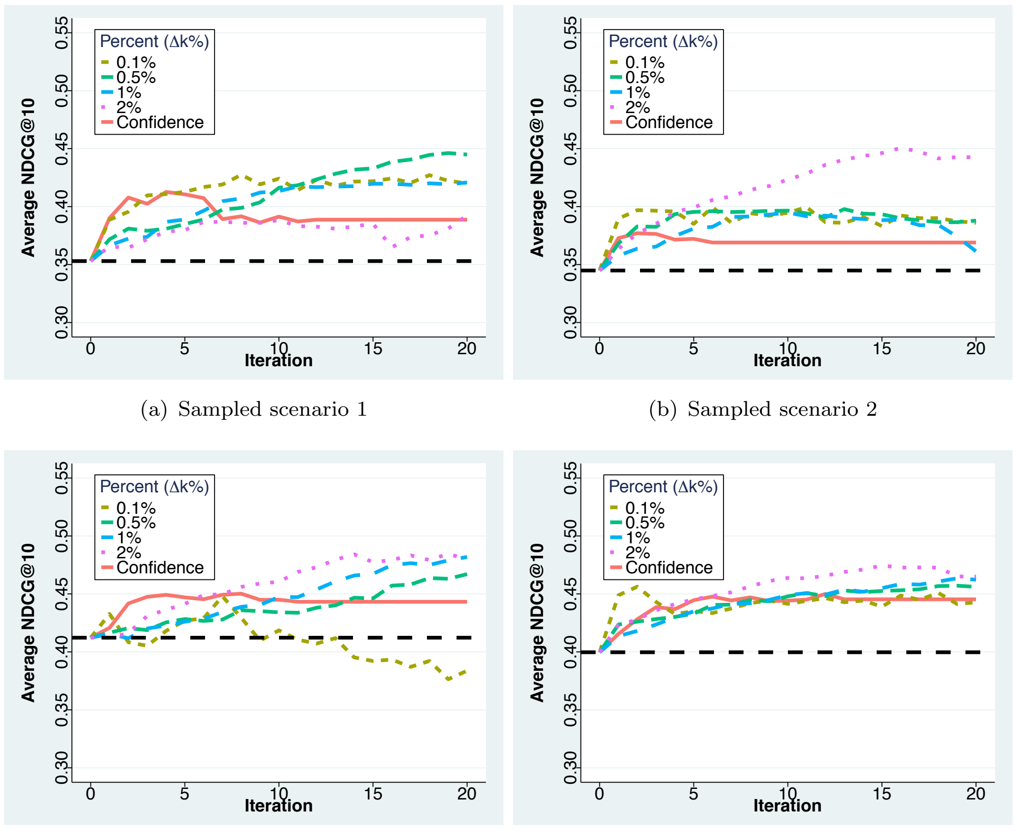
and confidence-based RankSelfTrain is that the number of relevant labels is fixed, and also all the unlabeled documents will be labeled as irrelevant.

The main challenge with this algorithm is how to set a proper parameter $\Delta k\%$ for a particular transfer setting. To compare the algorithms, we used the same sampling and testing strategy utilized in the previous section. The learning curves of different runs are plotted in Figure 5.6.

A glance at the figure illustrates the effectiveness of RankSelfTrain with different parameter settings. Most of the algorithms tested so far have shown a gradual increase in the effectiveness of the ranker with each iteration, starting from the source ranker (0^{th} iteration).

The algorithm performs variously with different parameter settings across multiple runs. For example, when $\Delta k\%$ is set to 2%, the algorithm gained the best effectiveness at the 2^{nd} run at the 20^{th} iteration, while it performed the worst at the 3^{rd} run.

Another challenge with this approach is knowing when to terminate the process. The algorithm can gradually label a certain amount of query-document pairs as relevant until all the pairs are labeled as relevant. It is not clear when the algorithm should add more relevant labels. Although we only plotted the first 20 iterations of the process in Figure 5.6, the five lines cross over at many iterations during training, which suggests that



(a) Sampled scenario 1 (b) Sampled scenario 2

(c) Sampled scenario 3 (d) Sampled scenario 4

FIGURE 5.6: Comparing the parameter settings for RankSelfTrain.

if the algorithm was halted at different iterations, the relative performance of different parameter settings would vary. Under the unsupervised TR scenario, it is difficult to determine which parameter to use and when to terminate.

Alternatively, the confidence-based approach does not require parameter setting except the confidence level, which can usually be set to a high value. The performance of the confidence-based approach is relatively stable compared with other settings, and it converges quickly. Although the performance may not be comparable to the best performance of other settings, it provides a more robust performance across different transferring settings.

5.8.3 Discussion

The results discussed above have illustrated that all the three proposed algorithms, RankPairwiseEM, RankHardLabelEM and RankSelfTrain can increase transferring effectiveness in most of the in-domain and cross-domain transferring scenarios. However, the improvements of the algorithms may not be consistent under different transferring settings. The RankPairwiseEM and RankSelfTrain algorithms tend to be more robust as they consistently outperform the source ranker across various test collections. RankSelfTrain showed slightly better consistency compared with the RankPairwiseEM and is easier to implement.

Parameter settings are critical for both RankHardLabelEM and RankSelfTrain algorithms. Setting the parameters for both algorithms based on some assumptions can gain acceptable results. However, the reliability and effectiveness could likely be improved if some supervision is provided.

Ideally we would hope to be able to monitor the cost of the algorithm during training to determine when to stop the iterations. However, this is not reliable for the unsupervised TR case due to two reasons: 1) relevance labels for the target data are imputed; 2) training data is gradually added to the training set. We would also hope that testing the variance of the performance on the training set would give us some indication of when to stop the iteration. We have tried the ideas, but it didn't work until we relaxed the condition to have a minimally supervised set, as later introduced in Chapter 6.

5.9 Conclusion

Aiming to improve learning-to-rank for scenarios where a ranker has to be transferred to a new collection with no available training data, we demonstrate three novel self-labeling unsupervised TR algorithms, RankPairwiseEM, RankHardLabelEM and RankSelfTrain. RankPairwiseEM is an application of an EM algorithm on unsupervised TR problems, which looks to achieve transfer effectiveness via maximizing the pairwise preference probabilities in the target collection. RankHardLabelEM is inspired by a hard EM approach, which applies an iterative process that predicts imputed relevance labels and updates models iteratively, while RankSelfTrain employs self-training (by gradually increasing the relevant label set) for semi-supervised learning.

Our algorithms can fit into a typical unsupervised TR scenario. These three novel algorithms do not rely on an instance-based density-ratio estimation for transferring knowledge from the source to the target. Therefore, they avoid the difficult problem of defining a representation for "query-space" and calculate density /similarity over it.

The three algorithms were tested on six transferring scenarios, with LambdaMART used as the base ranker. The results of the six scenarios show that with some simple parameter settings, all the algorithms can achieve improvements over the source ranking function, although in some cases the improvements are minimal.

Our experimental results showed that self-labeling methods are more effective than instance-weighting algorithms, with both new approaches outperforming a state-of-the-art instance-weighting algorithm across various test scenarios.

To confirm whether the effectiveness of self-labeling methods can perform consistently over different transferring collections, we demonstrated improvements via an average rank-based visualization method. The Nemenyi test on the results showed that both RankPairwiseEM and RankSelfTrain can significantly outperform λMART.source across different test collections.

We tested RankHardLabelEM and RankSelfTrain, under the "self-labeling paradigm", with different parameter settings to demonstrate how the parameters can impact effectiveness. For both algorithms, we have illustrated that the algorithms can achieve better results with an optimal parameter setting. However, it is difficult to estimate the parameters under the unsupervised TR setting. Instead, our confidence-based approach for RankSelfTrain has shown to be effective and stable.

The evaluation of self-labelling algorithms has shown their potential in tackling the unsupervised TR algorithms. However, the sensitivity analysis showed that some of the algorithms are sensitive to the parameter setting; in the next chapter, we will investigate proper ways to control the self-labelling algorithms with a limited number of relevance labels from the target collection.

Chapter 6

Minimally Supervised TR

In the previous chapter, we discussed the potential of using self-labeling methods for solving the unsupervised TR problem. However, one key issue that we found for self-labeling algorithms is that it takes a long time to converge, and the performance of the algorithms is sensitive to the parameter settings. In this chapter, we relax the unsupervised TR condition to allow limited amounts of relevance labels from the target collection, which will help calibrate the transferring. This is usually achievable in reality. For example, a search engine company may be able to run a few queries in the corpus from a new market and then obtain some editor judgements.

6.1 Introduction

It has been demonstrated in the previous chapter that TR with self-labeling methods can help transfer a ranking model from one collection to another. However, one bottleneck of unsupervised self-labeling methods is that the algorithms require parameter setting and some of the parameters are sensitive to the transfer setting. Self-labeling methods assume that there exists commonalities between the source and target collection. As a result, using a ranking function trained with a source collection can infer weak labels for a target collection. Starting from the model trained with source-only data, the algorithm can keep evolving with imputed weak labels from the target collection to better fit the target collection. However, the similarities between the source and target collection may vary, which may lead to differences in the label imputation accuracy. Without knowing any label information from the target collection, the transferring process may introduce too much noise during training. On the other hand, if the label imputation becomes too conservative, the impact of the target collection data with imputed labels becomes negligible. Unless some monitoring process is provided, one will find it hard to tune hyper-parameters.

In this chapter, we aim to solve the hyper-parameter problem by introducing a few relevance judgments for the target collection as a validation set for helping make those decisions. The labels from the target collection can help determine the hyper parameter for training and thus can improve the reliability of the models. We call this approach *Minimally Supervised TR (MSTR).*

Obtaining a small amount of training labels is usually achievable through crowd-sourcing or using less sensitive data, for example, annotating internally used queries. Different from supervised TR algorithms where the labeled target data is used during training [17, 19, 74], the labeled queries from the target collection are only used for calibrating the training. We argue this is a more reliable setup as the target labels provide some insights into the performance of the model rather than blindly using them for training. This setup is important especially when the labels are maintained at a minimal level. For example, relevance judgement for users' email queries is usually impossible due to privacy. Instead, a company may be able to use their staffs' work emails for relevance judgement. With the minimally supervised TR setting, this would give the algorithm more confidence of performance during the training.

We proposed a MSTR algorithm called *PairwiseRankSelfTrain*, which uses a small sample of the target collection data to calibrate the training of a variant of the RankSelfTrain algorithm that generates preference labels for pairs of documents. Extensive experiments on the Yahoo!L2R collection and Microsoft collections demonstrate that the novel algorithm, PairwiseRankSelfTrain, cannot only improve the effectiveness over a source model, our previous unsupervised TR algorithm, RankSelfTrain, but also the target model, which is the model that is directly trained on the target training set with labels.

In this chapter, the following research questions are addressed to gain better understanding of the properties of minimal supervision for TR:

1. How can the validation set be configured to maximize its reliability in measuring the performance of transferred rankers?
2. Does MSTR gain better effectiveness performance on the target collection than other TR settings?

6.2 Minimally Self-Training for TR

Self-labeling methods have shown to be an effective approach for TR problems in the previous chapter. However, most of those algorithms require a proper parameter setting. The RankSelfTrain is demonstrated to be effective across different settings.The algorithm is done through the following steps:

1. Initializing the parameters for a ranker with source data.
2. Computing imputed relevance labels for all unlabeled query-document pairs from an unlabeled target training set.
3. Updating ranker via training with both source and target training data with imputed labels.

A more detailed explanation of RankSelfTrain is given in Algorithm 4 in Chapter 5. At each iteration of the RankSelfTrain, the algorithm needs to determine the confidence of the label imputation. As the LambdaMART algorithm does not generate the confidence or probability scores for the degree of relevance, the estimation of confidence of label imputation becomes difficult. In the previous chapter, we studied two methods to obtain the confidence scores:

1. Use the model trained in the last iteration to predict the relevance scores for all remaining documents, then take the top-k documents with the highest relevance scores as relevant documents and the rest as irrelevant documents.
2. Similarly, first predict the relevance score for every document in the remaining collection. Then the algorithm estimates a relevance probability using kernel estimation with equation 5.21.

The problem with the first approach is that it requires selecting the amount of imputed relevance labels to be added to the training set, and the percentage $\Delta k\%$ is sensitive to different transferring settings. Moreover, the algorithm does not always converge to an optimal ranker for the target collection. This is because too many inaccurate labels are added to the training set. The second approach was shown to be more robust under different transferring settings. However, the results in Figure 5.6 suggest that the approach does not always yield the optimal transfer effectiveness compared with a proper $\Delta k\%$ setting.

To improve the effectiveness and robustness of unsupervised TR algorithms, we propose to introduce a small number of relevance labels from the target collection as calibration data. Different from supervised TR settings, where the labeled data from the target collection is used for training, under this MSTR setting, the labeled data will only be used for tuning the hyper-parameters for the unsupervised TR algorithms.

There are several reasons why the labeled target data is not used for training: 1) under the MSTR setting, we only require a minimal number of relevance judgements for the target collection, fewer than the number needed for training a supervised TR algorithm; 2) the labeled training data is too small to split into training and validation sets; 3) it is hard to build the confidence of the transferred ranking function, without any monitoring of the performance during the transferring; 4) Having the training data as calibration data means there is a risk of overfitting.

Two hyper-parameters need to be tuned to enable better effectiveness and reliability of the RankSelfTrain algorithm, which will be explained and discussed in the following two sections. The methodologies of tuning those parameters with validation data are also discussed.

6.2.1 Pairwise Label Imputation

At the start of each iteration, RankSelfTrain needs to predict the relevance labels for all the unlabeled query-document pairs for training from the target collection, given the ranker parameters estimated from the previous iteration. This is done by using the updated ranker $f^{(t-1)}$ to predict a similarity score s_{ij}^{t-1} for every unlabeled query-document pair. As each pair x_{ij}, the algorithm needs to determine the most likely label (r_{ij}) for the pair with the confidence level, $p(r_{ij}|s_{ij}^{t-1})$, given the similarity score.

In Chapter 5, the confidence score is estimated using Bayes' rules:

$$p(r_{ij} = 1|s_{ij} = \alpha) = \frac{p(r_{ij} = 1)p(s_{ij} = \alpha|r_{ij} = 1)}{\sum_{v=0,1} p(s_{ij} = \alpha|r = v)p(r = v)} \tag{6.1}$$

The conditional distribution $p(s_{ij}|r = 1)$ and $(s_{ij}|r = 0)$ are estimated using Kernel Density Estimation by combining the source data together with already labeled data from the target collection. Similarly, the prior probabilities of $p(r_{ij} = 1)$ and $p(r_{ij} = 0)$ are estimated using the source data, adjusted by the imputed relevance labels. At each iteration, we only impute relevant labels - all the remaining documents will be labeled as irrelevant documents and be used as unlabeled documents in the next iteration. However, using the source data prior probability to approximate the prior probability for the target collection is very inaccurate. Moreover, Kernel Density Estimation is more computationally expensive.

Since the LambdaMART algorithm indirectly optimizes the pairwise loss (as part of a list-wise loss, which is measured by swapping pairs of documents in the ranking), we can directly use the modeled preference probabilities $p(r_{ij} > r_{ik})$ to predict the confidence of pairwise preferences using Equation 5.3 from Chapter 5. The RankPairwiseEM algorithm used the probabilities as weights to calculate the expected cost of the current model. Instead, we now use the preference probabilities as confidence scores for label predictions. At each iteration of the self-training process, if an unlabeled pair of documents has a confidence score higher than the threshold, it will be added to the training set and the label will be kept for the rest of the iterations. To choose the best threshold, we train several candidate ranking models with different candidates and pick the best one by measuring the ranking effectiveness of the current ranker on the validation set.

6.2.2 Early Stopping with the Validation Set

It can be difficult to develop a single unsupervised strategy to determine when to stop the RankSelfTrain processes when the algorithms are applied to a new collection with little relevance information.

Allowing the algorithms to determine which ranker is the optimal over the iterations, it may require some relevance judgments for query-document pairs from the target collection as the validation set. A ranker trained on different iterations can be tested and compared on the validation set to give an image of how well the rankers perform.

However, in a real TR scenario, the objective is to optimize the effectiveness of the ranking functions on a new collection while minimizing the cost. So there will always be constraints on how many relevance judgments can be obtained. If there is sufficient budget to generate relevance labels for the target collection, TR may not be helpful as it introduces noise from another collection. So we only consider the case where a few labeled target query-document pairs are available for validating the performance. However, measuring the performance of the rankers on a small validation set may not represent their real performance on the entire collection. As a result, there may be variations in terms of the measurements of system performance reflected on the judged queries. One could use some active learning techniques to select a subset of queries that are more representative for evaluation, or randomly select a small number of queries and then use a statistical tool to make decisions. In reality, one may not have the flexibility to choose the queries they want. As a result, in this chapter, we chose the second approach to demonstrate the effectiveness of the algorithm.

To decide when to stop the process, we conduct a paired t-test between the performance of two consecutive iterations. If the performance has dropped significantly for a certain number of iterations, we terminate the process and pick the best model. As a small sample size was used, there is generally a risk that Type II Error can arise where there t-test cannot measure the significance of the difference when there is one. It puts a strict condition for the algorithm to update. If it happens in the first iterations, one needs to increase the sample size.

6.2.3 PairwiseRankSelfTrain

Algorithm 6 describes the proposed PairwiseRankSelfTrain algorithm, which starts by training a source ranker $f^{(0)}$ with all the labeled data from the source collection. It initializes an empty set Ω to store the pairwise label imputation. Lines 4-26 explains the label imputation and ranker updating for each iteration. At each iteration, with each

Algorithm 6: PAIRWISE SELF-TRAINING FOR RANKING

```
Input: Source queries $Q^{so}$ and judgments $R^{so}$, target queries $Q^{ta}$, confidence
       thresholds $\eta$s and a validate set $Q_v^{ta}$, $R_v^{ta}$
Output: Ranking function $f$
1  PairwiseRankSelfTrain($Q^{so}$,$R^{so}$,$Q^{ta}$,$\eta$)
2      Initialize set of labeled docs to be empty: $\Omega^{(0)} = \emptyset$;
3      Train ranker $f^{(0)}$ using $(Q^{so}, R^{so})$ with Eq. 5.13;
4      for $t \in \{1, ...\}$ do
5          Calculate similarities for all query-doc pairs;
6          foreach unlabeled pair using $f(\mathbf{x}_{ij}, \mathbf{x}_{ik}) \notin \Omega^{(t-1)}$ do
7              Calculate pairwise probabilities for all unlabeled doc pairs according to
               Equation 5.2;
8              best_validation_score=0;
9              best_model=$f^{(t-1)}$;
10             foreach $\eta \in \eta s$ do
11                 $\hat{\Omega}_\eta^{(t)} = \Omega^{(t-1)}$;
12                 if $p(r_{ij} > \mathbf{x}_{ik})|s_{ij}^{(t-1)}, s_{ik}^{(t-1)}) > \eta$ then
13                     Add $(\mathbf{x}_{ij}, 1), (\mathbf{x}_{ik}, 0)$ to $\hat{\Omega}^{(t)}$;
14                 else if $p(r_{ij} < \mathbf{x}_{ik})|s_{ij}^{(t-1)}, s_{ik}^{(t-1)}) > \eta$ then
15                     Add $(\mathbf{x}_{ij}, 0), (\mathbf{x}_{ik}, 1)$ to $\hat{\Omega}^{(t)}$;
16             end
17             Train ranker $\hat{f}^{(t)}$ using Eq. 5.15;
18             Estimate performance $s(\theta^t)$, by measuring the performance of $\hat{f}^{(t)}$ with
               validation set,$M(\hat{f}^{(t)}, Q_v^{ta}, R_v^{ta})$;
19             if $s(\theta) >$ best_validation_score then
20                 best_model=$\hat{f}^{(t)}$;
21                 $\Omega^{(t)} = \hat{\Omega}_\eta^{(t)}$;
22             end
23         end
24         if $(|\Omega^{(t)}| - |\Omega^{(t-1)}|) == 0$ then
25             return $f^{(t-1)}$;
26         end
27         if Model has not been improved for certain number of iterations, measured by
           paired t-test then
28             return Best model in the last few iterations
29         end
30     end
```

candidate threshold for the confidence score, different candidate parameters are evaluated to get the best performing model (lines 6-22). During each run with a parameter setting, the algorithm first computes the pairwise probabilities for all unlabeled document pairs from the unlabeled set. Notice that once a document pair has been labelled, its label will not be changed in the following iterations. The updated document pairs will be added to the training set together with their relevance labels to a temperate set

$\hat{\Omega}_{\eta}^{(t)}$. With all the imputed relevance labels, an updated ranker is trained together with the source data via:

$$\hat{\theta}^{(t+1)} = arg\ \min_{\theta} \sum_{q_i \in Q^{so}} \sum_{d_{ij} \in q_i} C_{ij} + \sum_{\varrho \in \hat{\Omega}_{\eta}^{(t)}} \sum_{\mathbf{x}_{ij} \in \varrho} \hat{C}_{ij}(\hat{R}^{(t)}) \tag{6.2}$$

where ϱ is an imputed document pair, $\hat{R}^{(t)}$ is the imputed label. We measure the performance of the current model, $s(\theta^t)$, according to their performance on the validation set. We choose the best performed model to update the current best model and then update the best training set accordingly. Finally, if there are no more labels to be added to the set, the last ranking model will be returned. Or if the model has not been improved for a certain number of iterations according to paired t-test, the best performed algorithm will be returned.

6.3 Data and Methods

As it has been shown in the previous chapters, the performances of TR algorithms depends on the similarity of the source and target collections. However, under the unsupervised TR scenario, there is no established ways to measure the similarity of L2R collections. As a result, the criteria for an effective unsupervised and minimally supervised TR algorithm is that it can robustly improve or at least retain the effectiveness of the source model under all circumstances.

To evaluate the performance of the proposed PairwiseRankSelfTrain algorithm, we run a series of experiments to simulate different transfer settings where the similarity between the source and target collection vary.

6.3.1 Datasets

Notice that experiment setting is a bit different from previous chapters due to minimally supervision setting, which requires a random sample from the target collection. To obtain different similarities between the source and target collection, we randomly select a subset from the source collection as the source dataset, and select a subset from the target collection as the target dataset.

Table 6.1: Experiment Setting for Minimally Supervised TR

	Source Dataset	Unlabeled Target Dataset for Training	Target Dataset for Testing	Target Dataset for Calibration
MSLR to LETOR 4.0	3k MSLR queries	1,213 LETOR 4.0 queries	1,213 LETOR 4.0 queries	50 LETOR 4.0 queries
Yahoo!L2R Set 1 to Set2	3k Set 1queries	3140 Set 2 queries	3140 Set 2 queries	50 Set 2 queries

We choose two transferring settings from previous chapters as the testing environment: 1) transferring from Set 1 to Set 2 in Yahoo!L2R; 2) transferring from MSLR to LETOR 4.0. Both settings are close to real scenarios when one wants to transfer their search algorithms between different markets. Moreover, the size of queries in both cases are larger than others for randomization. The detailed description of the datasets are shown in Table 2.1.

In this chapter, we create multiple synthetic data for TR via randomly selecting 3k queries from the the source and target dataset separately. The data pooled from Set 1/MSLR is used as the source collection data and the data from Set 2/LETOR 4.0 is used as the target collection data. The randomization process will create variations in the similarities. We randomly selected 50 queries from the target collection as the target validation set, the remaining queries of the target collection data are randomly split into two equal sized query sets, one for training and the other for testing. The ground-truth labels for the training data from the target collection are removed, and the data will be used for label imputation and training during the transfer. For data split, training and testing have been run 10 times to measure variation. The detailed experiment setting is shown in Table 6.1.

6.3.2 Setup and Measurements

The XGBoost library implementation of LambdaMART was used as the base ranker.[1] We use "pairwise:ndcg" as the objective function for XGBoost, which means that the updating gradients are the λs where the swap change $|\Delta Z|$ was measured using the NDCG metric. The tree size was set to 1000, the maximum number of leaves was set to 10, and the subsample size was set at 0.5 to reduce the variance in the sample. During the transfer, the alpha level for the t-test used to determine whether to update models was set at 0.1[2].

[1] https://github.com/dmlc/xgboost version 0.81

[2] Since the sample size is small, and the variation in the sample is larger, so we set the alpha level to a larger value to allow for more variation.

For all the algorithms, we set the maximum iteration Γ to 20. For the RankSelfTrain algorithm, the threshold on confidence was set at 95%. The σ for pair-wise probability was set as 1 in the PairwiseRankSelfTrain algorithm and the candidate thresholds were set at $\{0.9, 0.92, 0.94, 0.96, 0.98\}$.

The following baselines were considered:

- **λMART.source:** LambdaMART trained with all the data from the source collection.
- **λMART.target:** LambdaMART trained with data from the target collection via cross-validation.

The following label imputation algorithms were tested:

- **RankSelfTrain:** Self-training-based algorithm for relevance labels from last chapter, using LambdaMART as the base ranker.
- **PairwiseRankSelfTrain:** Self-training-based algorithm for document pair preferences, using LambdaMART as the base ranker. The algorthms use a target validation set to calibrate the hyper-parameters for training.

All models were evaluated using NDCG [106], with a rank cut-off of 10. Statistical significance was determined using a two-tailed paired t-test, with a threshold of 0.05.

6.4 Result and Discussion

Table 6.2 shows the results when transferring from the subset of MSLR to LETOR 4.0. Due to the fact that the query size and the document depth in LETOR 4.0 is small, the model trained on the source dataset, MSLR, which has larger training data, has similar or even better performance on the target collection than the λMART.target. This simulate the situation when the source model is more effective than the target collection. In such situations, an effective TR would at least have similar performance with the λMART.source. The existing model RankSelfTrain has shown its reliability in such situations that it did not harm the performance of the source model. The proposed

Table 6.2: Effectiveness (NDCG@10 score) for 10 runs of minimally supervised TR when transferring from MSLR to LETOR 4.0. Bold text indicates the best scores of each row, † denotes the figure is significantly better than λMART.source, ‡ denotes the figure is significantly better than RankSelfTrain, ⋆ denotes the figure is significantly better than λMART.target. $p < 0.05$

Run	λMART.source	RankSelfTrain	PairwiseRankSelfTrain	λMART.target
R1	0.569	0.569	**0.608** † ‡ ⋆	0.564
R2	0.561 ⋆	0.562 ⋆	**0.602** † ‡ ⋆	0.553
R3	0.572 ⋆	0.576 † ⋆	**0.606** † ‡ ⋆	0.563
R4	0.561	0.562	**0.602** † ‡ ⋆	0.557
R5	0.563	0.567	**0.598** † ‡ ⋆	0.560 ⋆
R6	0.573 ⋆	0.571 ⋆	**0.610** † ‡ ⋆	0.565
R7	0.565	0.565	**0.606** † ‡ ⋆	0.565
R8	0.567	0.569	**0.611** † ‡ ⋆	0.562
R9	0.559	0.561 ⋆	**0.601** † ‡ ⋆	0.552
R10	0.565	0.564	**0.605** † ‡ ⋆	0.565

Table 6.3: Effectiveness (NDCG@10 score) for 10 runs of minimally supervised TR when transferring from Yahoo!L2R Set 1 to Set 2. Bold text indicates the best scores of each row, † denotes the figure is significantly better than λMART.source, ‡ denotes the figure is significantly better than RankSelfTrain, ⋆ denotes the figure is significantly better than λMART.target. $p < 0.05$

Run	λMART.source	RankSelfTrain	PairwiseRankSelfTrain	λMART.target
R1	0.740	0.743†	**0.759**† ‡ ⋆	0.750† ‡
R2	0.743	0.746†	**0.758**† ‡ ⋆	0.748†
R3	0.736	0.739†	**0.757**† ‡ ⋆	0.746† ‡
R4	0.740	0.742	**0.757**† ‡ ⋆	0.749† ‡
R5	0.735	0.735	**0.751**† ‡ ⋆	0.743† ‡
R6	0.741	0.741	**0.758**† ‡ ⋆	0.747† ‡
R7	0.742	0.743	**0.759**† ‡ ⋆	0.746 †
R8	0.734	0.734	**0.754**† ‡ ⋆	0.745† ‡
R9	0.739	0.742†	**0.756**† ‡ ⋆	0.745† ‡
R10	0.734	0.738†	**0.754**† ‡ ⋆	0.743 †

algorithm under the minimally supervised TR scenario, turns out to be the most effective model among all. The PairwiseRankSelfTrain algorithm has shown significantly better performance than all the other algorithms in all runs.

The results for transferring from Yahoo!L2R are shown in Table 6.3. This is the scenario when the source and target collection come from different markets with different languages. Similar to what we found before, performance degradation was observed when LambdaMART is trained and tested on two different datasets from different data distribution. For all the 10 random sets of experiments, λMART.target is significantly better than λMART.source. With the label-imputation method, the RankSelfTrain model we developed from the previous chapter shows some improvement over the λMART.source in most cases, although the improvements are not significant. Furthermore, all of these algorithms are significantly worse than the λMART.target. The proposed algorithm, PairwiseRankSelfTrain, appears to be the best among all the other algorithms. In all the 10 sets of experiments, the PairwiseRankSelfTrain algorithm outperformed the λMART.source and RankSelfTrain algorithm significantly. Moreover, the PairwiseRankSelfTrain algorithm outperformed the λMART.targets which are ranking functions that were directly trained on the data from the same collection.

6.4.1 Discussion

Overall, the performance of the proposed algorithm Pairwise has shown its advantages of using a target validation set to tune the hyper-parameters in order to optimize the performance. The result shows that with a proper TR algorithm and setup, the performance of a transferred ranker can be better than directly training with a relatively small number of training data.

In this experiment, we have used only 50 queries as the target validation set and demonstrated a significant performance increase. In real applications, the variations in the query set may be so large that the t-test will not detect the difference between algorithms. In such cases, one would need to make a decision whether to increase the alpha level for the t-test, or increase the number of relevance judgements to expand the reliability of the judgement.

Apart from the benefits of using a validation set for tuning hyper-parameters, the new pairwise label imputation is the other reason why the new algorithm was performing much better than the previous one. The RankSelfTrain only assumes that relevance labels are binary from the target collection. However, two relevant documents may have a different granularity of relevance, which means one document could be more relevant than the other, which will not be reflected in the RankSelfTrain label imputation process. Differently, with the PairwiseRankSelfTrain algorithm, each pair of documents in the target collection will have a pairwise preference probability for label imputation, which can obtain a more accurate label process for training.

6.5 Conclusion

This chapter aims to improve the effectiveness and reliability of the unsupervised TR by introducing a minimal number of relevance judgments from the target collection. We proposed a PairwiseRankSelfTrain algorithm that directly imputes the relevance preference labels for each pair of documents in the target collection, determined by the preference probability inferred from the model trained in the previous iterations. The threshold of when to add the preference labels and when to terminate the transfer is calibrated by a minimal number of validation sets from the target collection.

The algorithm, together with our previous algorithm, RankSelfTrain, were tested on the two transferring sets by randomly generating source and target collections. The results on the Yahoo!L2R and Microsoft collections show that the new proposed algorithm can outperform the baseline, which is a model trained on the source collection and directly applied to the target. The new algorithm is also significantly better than our previous approach on all the ten sets of experiments. Moreover, the algorithm demonstrates to be significantly better than models that were directly trained on a set of data from the same collection.

The minimality of the calibration set has not been explored in this chapter. The calibration set from the target collection is not used for training, it will not directly impact the training process of the algorithm. However, the labels will be used for calibrating the hyper-parameters during the training process, the calibration set may have some impact on the performance of the algorithm. There are different ways to explore the effectiveness of an IR test collection, for example, some [129] have studied minimal test collections. In this chapter, we simply applied a random procedure to select the queries to demonstrate the effectiveness of the algorithm. However, better methodologies can be investigated in the future.

Chapter 7

Conclusion and Future Work

Due to the difficulties of obtaining relevance labels for a new collection, training an effective and reliable ranking function for an information retrieval system with limited relevance labels has attracted much research. In many cases, because of privacy issues in assessing documents, or because of the highly personalized tasks like job search or hotel search, there are many search domains where it is difficult or impossible to obtain relevance labels to build offline test collections for L2R. Because of the ability to share knowledge between different tasks, transfer learning has been considered as one possible solution for such ranking tasks, especially when there exists a related test collection that provides plenty of relevance labels. For example, a search engine company may also have an offline test collection for an old market, but not the new one. Existing studies have been mainly focused on transfer learning when there are already some relevance labels from the target collection, and not cases when there are no any explicit relevance judgments. Moreover, several fundamental research questions have not been answered which would help tackle the TR problem: What are the generalization abilities of different L2R models? How does changing the document collection affect the data distribution for L2R tasks? How can one determine the "transferabilities" for L2R tasks? Which are the best-performing TR techniques? How can one measure the performance of different TR algorithms? This thesis has contributed to answering the above research questions by thorough examination and development of different TR techniques on large publicly-available L2R test collections.

7.1 Thesis Contribution

Although some attempts have been made to apply transfer learning techniques to L2R tasks, few researchers have investigated the causes of performance degradation. In Chapter 3, we formalized the data generating process for L2R collections and showed that the data distribution of an L2R task is controlled by many factors. As a result, any changes in the document set, query set, and relevance judgment process may cause a distribution change. The impact of different types of changes in the data distribution differs with individual L2R algorithms. As a result, for a particular TR scenario, understanding what caused the differences in the distribution and what performance change is expected is the first step in conducting a successful knowledge transfer or determining whether transfer learning is needed.

As shown in Chapter 3, unlike with other natural language processing (NLP) tasks, the ranking effectiveness of a rank learning algorithm may not change even when applied to a different domain. Part of the reason is that the features used for defining an L2R task are not specific the words in the documents, but the statistics resulting from term-matching across the query-document pairs. At the same time, the domain adaption techniques used for NLP tasks that incorporate lexicon correlations cannot be used for L2R tasks. The data distribution change in the input feature space seems to be the major challenge. One common solution for tackling the so-called covariate shift issue is through a technique called "instance weighting" that computes importance weights for each training instance in the source collection to move the source data distribution closer to the target data distribution. The algorithms are trained by optimizing the weighted loss according to their importance. Since most of state-of-the-art L2R algorithms optimize a ranking function at the pair or query level, one difficulty of applying instance weighting techniques to L2R is instance weighting at pair level or query level. Chapter 4 answers the challenge by examining both existing and new query-level instance weighting techniques on various publicly available L2R collections. The thorough comparison shows that none of the techniques consistently outperforms the others including the source model trained without any weights. Our new proposed model seems to be more reliable, however, there are still cases when the new approach does not work properly. The underlying reason was that the high-level representation of queries may lose structural information in the data.

Instance weighting for L2R datasets can be difficult and inaccurate. Instead, in Chapter 5, we propose to directly impute relevance labels for training data from the target collection and then use these imputed labels for training. As the data from the target collection is directly used for training, the problem of covariate shift can be tackled. The main challenge of the label imputation process is to generate accurate predictions for each query-document pair. This could be done by different variants of the Expectation-Maximization processes. Experiments over various test collections show that label-imputation algorithms are more effective and robust across different transfer environments.

The label-imputation methods appears to be an effective solutionfor unsupervised TR tasks. However, the performance of some of those algorithms are sensitive to parameter settings for particular transfer tasks. In Chapter 6, we looked to solve the problem by obtaining a minimal number of relevance labels from the target collection and use them as a validation set to calibrate the hyper-parameters during training. This technique demonstrated consistent improvements in performance, and indeed outperformed models trained on the target data alone.

7.2 Other Contributions

This thesis has also made a number of other contributions to TR research as well as broader research on transfer learning and L2R.

Transfer learning for ranking is a relatively new task and thus there have been limited resources available for testing different algorithms. This thesis has established various test environments for testing both supervised and unsupervised TR algorithms, which could potentially be used for testing other TR algorithms. Moreover, the effectiveness of a TR algorithm could vary across different transfer environments, and thus measuring the consistency of a TR algorithm is an important metric to test the robustness of a TR algorithm. In this thesis, we propose to perform posthoc analysis using the Friedman and Nemenyi tests to determine significant differences between algorithms across different experimental settings. .

The thesis has made several attempts to minimize the differences in the data distribution between the source and target collection when the objective function of an algorithm is not minimized at the instance level. The study of this thesis can inspire other work on unconventional transfer learning tasks like preference learning [130]. Moreover, Chapter 4 discussed various ways to measure the similarities between different queries. The study of the similarities of queries can be used to understand the importance of query selection for active learning as well as proper ways to establish effective offline L2R test collections.

7.3 Limitation of the Study

The study of this thesis is largely limited by the available test environment. As has been mentioned previously, apart from the Yahoo!L2R, there are no other public datasets that can be used for TR settings. As a result, we are not explicitly testing the distribution change when a particular controlling factor has changed. Moreover, as for those large test collections like MSLR and Yahoo!L2R, there is no available information for the queries, which has limited us to explore other possibilities to estimate importance weights for source queries via the information of the queries.

7.4 The Future

The lack of reliable relevance labels for building offline L2R test collections is still one of the biggest challenges for developing a sound ranking model for commercial information retrieval systems. Thus, the study of TR has continuously attracted the attention of researchers as well as industry. With a better understanding of the importance of query and document distributions for the training of L2R models, better approaches for query-level instance weighting can be developed. Apart from investigating the data distribution change of the feature space, the meta information of the queries can also be helpful in solving the TR problem.

Many other researches have turned to user engagements (e.g. user dwell-time on search results) to build offline [131, 132] or online L2R models [133–136] instead of explicit relevance judgments. TR is still a useful technique for these approaches as the implicit relevance labels could be biased and the interpretation of those engagement signals can be challenging. The study of TR algorithms for those tasks would help transfer knowledge from an existing successful task.

Recently, the advances in deep learning [137] techniques and deep learning solutions for retrieval and ranking [138–142] have attracted a large amount of attention. Training neural IR models requires massive amounts of training data. Moreover, most neural IR models use raw text features to represent queries and documents. The generalization ability of deep neural models for ranking may not be as great as it is for conventional L2R algorithms. For such learning algorithms, the learned high-level feature representation may be domain-specific. As a result, using TR techniques to solve the lack of label bottlenecks for neural IR models has much potential. In the field of deep learning, transfer learning has been a hot topic and many studies [65, 143] have been conducted to determine how best to transfer knowledge between two or more tasks. As a result, we would expect to see more studies on TR for neural IR models in the near future.

7.5 Overall Conclusion

The findings of this thesis suggest that the generalization abilities of L2R algorithms can vary across different situations. Instance weighting at the query level over the feature distribution appears an ineffectivesolution for unsupervised TR while label imputation could be an effective and reliable solution. Moreover, obtaining a few target relevance labels for validation purposes to tune the hyper-parameters for unsupervised TR can dramatically improve the effectiveness and reliability of the model.

Bibliography

[1] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and others. Okapi at TREC-3. In *Overview of the Third Text REtrieval Conference (TREC3)*, volume 109, page 109. Gaithersburg, MD: NIST, 1995.

[2] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, Melbourne, Australia, 1998. ACM Press.

[3] Fei Song and W. Bruce Croft. A General Language Model for Information Retrieval. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, pages 316–321, New York, NY, USA, 1999. ACM.

[4] Mark Sanderson, Andrew Turpin, Ying Zhang, and Falk Scholer. Differences in effectiveness across sub-collections. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 1965–1969, Maui, Hawaii, USA, 2012. ACM Press.

[5] Timothy Jones, Andrew Turpin, Stefano Mizzaro, Falk Scholer, and Mark Sanderson. Size and source matter: Understanding inconsistencies in test collection-based evaluation. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1843–1846, Shanghai, China, 2014. ACM Press.

[6] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to BM25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, page 58, Melbourne, VIC, Australia, 2014. ACM Press.

[7] Tie-Yan Liu and others. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[8] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset Shift in Machine Learning.* The MIT Press, Cambridge, MA, 2009.

[9] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[10] Amos Storkey. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, pages 3–28, 2009.

[11] Rong Yan and Jian Zhang. Transfer Learning Using Task-Level Features with Application to Information Retrieval. In *Twenty-First International Joint Conference on Artificial Intelligence*, pages 1315–1320, 2009.

[12] Sandeepkumar Satpal and Sunita Sarawagi. Domain adaptation of conditional probability models via feature subsetting. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 224–235, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[14] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics, 2006.

[15] Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1189–1198, Washington, DC, USA, 2010. ACM Press.

[16] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 20*, pages 41–48, Vancouver, Canada, 2007. Neural Information Processing Systems Foundation.

[17] Depin Chen, Yan Xiong, Jun Yan, Gui-Rong Xue, Gang Wang, and Zheng Chen. Knowledge transfer for cross domain learning to rank. *Information Retrieval*, 13 (3):236–253, 2010.

[18] Wei Gao, Peng Cai, Kam-Fai Wong, and Aoying Zhou. Learning to rank only using training data from related domain. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 162–169, Geneva, Switzerland, 2010. ACM Press.

[19] Kevin Duh and Akinori Fujino. Flexible sample selection strategies for transfer learning in ranking. *Information Processing & Management*, 48(3):502–512, 2012.

[20] Bo Geng, Linjun Yang, Chao Xu, and Xian-Sheng Hua. Ranking model adaptation for domain-specific search. In *IEEE Transactions on Knowledge and Data Engineering*, volume 24, pages 745–758, Hong Kong, China, 2009. ACM Press.

[21] Keke Chen, Jing Bai, Srihari Reddy, and Belle Tseng. On domain similarity and effectiveness of adapting-to-rank. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1601–1604, Hong Kong, China, 2009. ACM Press.

[22] Olivier Chapelle, Yi Chang, and T-Y Liu. Future directions in learning to rank. In *Proceedings of the Learning to Rank Challenge*, pages 91–100, 2011.

[23] Parantapa Goswami, Massih R. Amini, and Eric Gaussier. Transferring knowledge with source selection to learn IR functions on unlabeled collections. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, pages 2315–2320, Geneva, Switzerland, November 2013. ACM Press.

[24] Yanyan Lan, Tie-Yan Liu, Tao Qin, Zhiming Ma, and Hang Li. Query-level stability and generalization in learning to rank. In *Proceedings of the 25th International Conference on Machine Learning*, pages 512–519, Helsinki, Finland, 2008. ACM Press.

[25] Yanyan Lan, Tie-Yan Liu, Zhiming Ma, and Hang Li. Generalization analysis of listwise learning-to-rank algorithms. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 577–584, Montreal, Quebec, Canada, 2009. ACM Press.

[26] Jingbo Shang, Tianqi Chen, Hang Li, Zhengdong Lu, and Yong Yu. A parallel and efficient algorithm for learning to match. In *2014 IEEE International Conference on Data Mining*, pages 971–976. IEEE, 2014.

[27] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, and others. *Modern Information Retrieval*, volume 463. ACM press, New York, NY, USA, 1999.

[28] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.

[29] Thomas Roelleke and Jun Wang. TF-IDF uncovered: a study of theories and probabilities. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 435–442, Singapore, Singapore, 2008. ACM Press.

[30] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241, Dublin, Ireland, 1994. Springer-Verlag New York, Inc.

[31] ChengXiang Zhai. Statistical language models for information retrieval: a critical review. *Foundations and Trends in Information Retrieval*, 2(3):137–213, 2008.

[32] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. *SIGIR Forum*, 51(2):268–276, 2017. ISSN 0163-5840.

[33] D. Frank Hsu and Isak Taksa. Comparing rank and score combination methods for data fusion in information retrieval. *Information Retrieval*, 8(3):449–480, 2005.

[34] Matthew Lease and Emine Yilmaz. Crowdsourcing for information retrieval. In *ACM SIGIR Forum*, volume 45, pages 66–75. ACM Press, 2011.

[35] Craig Macdonald, Rodrygo L.T. Santos, Iadh Ounis, and Ben He. About learning models with multiple query-dependent features. *ACM Transactions on Information Systems*, 31(3):11:1–11:39, 2013.

[36] Niek Tax, Sander Bockting, and Djoerd Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51(6): 757–772, 2015.

[37] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[38] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, Edmonton, Alberta, Canada, 2002. ACM Press.

[39] Jaana Keklinen and Kalervo Jrvelin. Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002.

[40] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136, Corvalis, Oregon, USA, 2007. ACM Press.

[41] Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in WWW search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 478–479, Sheffield, United Kingdom, 2004. ACM. event-place: Sheffield, United Kingdom.

[42] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, Bonn, Germany, 2005. ACM Press.

[43] Evangelos Kanoulas and Javed A. Aslam. Empirical justification of the gain and discount function for nDCG. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 611–620, Hong Kong, China, 2009. ACM Press.

[44] Ramesh Nallapati. Discriminative models for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 64–71, Sheffield, United Kingdom, 2004. ACM Press.

[45] Ping Li, Qiang Wu, and Christopher J. Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in neural information processing systems 20*, pages 897–904, Red Hook, NY, USA, 2007. Curran Associates, Inc.

[46] Koby Crammer and Yoram Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*, pages 641–647. MIT Press, 2002.

[47] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Advances in Neural Information Processing Systems 15*, pages 961–968, Cambridge, MA, USA, 2003. MIT Press.

[48] Jason DM Rennie and Nathan Srebro. Loss functions for preference levels: Regression with discrete ordered labels. In *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*, pages 180–186. Kluwer Norwell, MA, 2005.

[49] Robin L Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975.

[50] R Duncan Luce. *Individual choice behavior: A theoretical analysis.* Courier Corporation, North Chelmsford, MA, 2005.

[51] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, Amsterdam, The Netherlands, 2007. ACM Press.

[52] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37, Orlando, FL, USA, 1997. Academic Press, Inc.

[53] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[54] Nick Craswell, David Hawking, Ross Wilkinson, and Mingfang Wu. Overview of the TREC 2003 Web track. In *TREC*, pages 78–92, 2003.

[55] Craswell, Nick and Hawking, David. Overview of the TREC-2004 web track. In *13th Text REtrieval Conference 2004 (TREC 2004)*, Gaithersburg, Md., 2004.

[56] William Hersh, Chris Buckley, TJ Leone, and David Hickam. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 192–201, Dublin, Ireland, 1994. Springer-Verlag New York, Inc.

[57] James Allan, Ben Carterette, Javed A Aslam, Virgil Pavlu, Blagovest Dachev, and Evangelos Kanoulas. Million query track 2007 overview. Technical report, University of Massachusetts, Amherst, MA, 2007.

[58] James Allan, Javed A Aslam, Ben Carterette, Virgil Pavlu, and Evangelos Kanoulas. Million query track 2008 overview. Technical report, Massachusetts University, Amherst, MA, 2008.

[59] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge*, volume 14, pages 1–24, Haifa, Israel, 2011.

[60] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on*

Information Systems, 35(2):15, 2016.

[61] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 949–952, Pisa, Italy, 2016. ACM Press.

[62] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

[63] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[64] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning.* Cambridge University Press, 2012.

[65] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, pages 2200–2207, Washington, DC, USA, December 2013. IEEE Computer Society.

[66] Yin Zhu, Yuqiang Chen, Zhongqi Lu, Sinno Jialin Pan, Gui-Rong Xue, Yong Yu, and Qiang Yang. Heterogeneous Transfer Learning for Image Classification. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, San Francisco, California, 2011. AAAI Press.

[67] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NIPS 2005 Workshop on Transfer Learning*, volume 898, pages 1–4, Vancouver, BC, Canada, 2005. MIT Press.

[68] Massih Reza Amini, Tuong Vinh Truong, and Cyril Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 99–106, Singapore, Singapore, 2008. ACM Press.

[69] Kevin Duh and Katrin Kirchhoff. Learning to rank with partially-labeled data. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 251–258, Singapore, Singapore, 2008. ACM Press.

[70] Kevin Duh and Katrin Kirchhoff. Semi-supervised ranking for document retrieval. *Computer Speech & Language*, 25(2):261–281, 2011.

[71] Ming Li, Hang Li, and Zhi-Hua Zhou. Semi-supervised document retrieval. *Information Processing & Management*, 45(3):341–355, 2009.

[72] Shangkun Ren, Yuexian Hou, Peng Zhang, and Xueru Liang. Importance Weighted AdaRank. In *Proceedings of the 7th International Conference on Advanced Intelligent Computing*, pages 448–455, Zhengzhou, China, 2011. Springer.

[73] Peng Cai, Wei Gao, Kam-Fai Wong, and Aoying Zhou. Weight-based boosting model for cross-domain relevance ranking adaptation. In *Proceedings of the 33rd European Conference on Advances in Information Retrieval*, pages 562–567, Dublin, Ireland, 2011. Springer.

[74] Depin Chen, Jun Yan, Gang Wang, Yan Xiong, Weiguo Fan, and Zheng Chen. Transrank: A novel algorithm for transfer of rank learning. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, pages 106–115, Washington, DC, USA, 2008. IEEE Computer Society.

[75] Peng Cai, Wei Gao, Aoying Zhou, and Kam-Fai Wong. Relevant knowledge helps in choosing right teacher: active query selection for ranking adaptation. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, Beijing, China, 2011. ACM Press.

[76] Wei Gao and Pei Yang. Democracy is Good for Ranking: Towards Multi-view Rank Learning and Adaptation in Web Search. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, pages 63–72, New York, NY, USA, 2014. ACM Press.

[77] Keke Chen, Rongqing Lu, CK Wong, Gordon Sun, Larry Heck, and Belle Tseng. Trada: tree based ranking function adaptation. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1143–1152, Napa Valley, California, USA, 2008. ACM Press.

[78] Qiang Wu, Chris JC Burges, Krysta M Svore, and Jianfeng Gao. Ranking, Boosting, and Model Adaptation. Technical report, Microsoft Research, 2008.

[79] Jianfeng Gao, Qiang Wu, Chris Burges, Krysta Svore, Yi Su, Nazan Khan, Shalin Shah, and Hongyan Zhou. Model adaptation via model interpolation and boosting for web search ranking. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 505–513,

Singapore, 2009. Association for Computational Linguistics.

[80] Bo Wang, Jie Tang, Wei Fan, Songcan Chen, Zi Yang, and Yanzhu Liu. Heterogeneous cross domain ranking in latent space. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 987–996, Hong Kong, China, 2009. ACM Press.

[81] Ke Zhou, Jing Bai, Hongyuan Zha, and Gui-Rong Xue. Leveraging Auxiliary Data for Learning to Rank. *ACM Transactions on Intelligent Systems and Technology*, 3(2):1–21, 2012.

[82] Bo Long, Yi Chang, Anlei Dong, and Jianzhang He. Pairwise cross-domain factor model for heterogeneous transfer ranking. In *Proceedings of the fifth ACM International Conference on Web Search and Data Mining*, pages 113–122, Seattle, Washington, USA, 2012. ACM Press.

[83] Mohammad Taha Bahadori, Yi Chang, Bo Long, and Yan Liu. Scalable heterogeneous transfer ranking. In *Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications-Volume 36*, pages 214–228, New York, NY, 2014. JMLR. org.

[84] Pierre Geurts and Gilles Louppe. Learning to rank with extremely randomized trees. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge*, volume 14, pages 49–61, Haifa, Israel, 2011. JMLR.org.

[85] Craig Macdonald, B. Taner Diner, and Iadh Ounis. Transferring Learning To Rank Models for Web Search. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 41–50, New York, NY, USA, 2015. ACM Press.

[86] Jing Bai, Ke Zhou, Guirong Xue, Hongyuan Zha, Gordon Sun, Belle Tseng, Zhaohui Zheng, and Yi Chang. Multi-task Learning for Learning to Rank in Web Search. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1549–1552, Hong Kong, China, 2009. ACM Press.

[87] Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Boosted multi-task learning. *Machine Learning*, 85 (1-2):149–173, 2011.

[88] Jing Bai, Fernando Diaz, Yi Chang, Zhaohui Zheng, and Keke Chen. Cross-market model adaptation with pairwise preference data for web search ranking. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*,

pages 18–26, Beijing, China, 2010. Association for Computational Linguistics.

[89] Hongning Wang, Xiaodong He, Ming-Wei Chang, Yang Song, Ryen W White, and Wei Chu. Personalized ranking model adaptation for web search. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 323–332, Dublin, Ireland, 2013. ACM Press.

[90] Jie Tang and Wendy Hall. Cross-domain ranking via latent space learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 2618–2624, San Francisco, USA, 2017. AAAI Press.

[91] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical report, Computer Science Department, Trinity College Dublin, 2004.

[92] Donna Harman. Overview of the first TREC conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 36–47, Pittsburgh, Pennsylvania, USA, 1993. ACM Press.

[93] Hal Daum III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[94] Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7):2031–2038, 2013.

[95] Stphane Clinchant and Eric Gaussier. Information-based models for ad hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 234–241, Geneva, Switzerland, 2010. ACM Press.

[96] Ambuj Tewari and Sougata Chaudhuri. Generalization error bounds for learning to rank: Does the length of document lists matter? *arXiv preprint arXiv:1603.01860*, 2016.

[97] Craig Macdonald, Rodrygo L. T. Santos, and Iadh Ounis. The whens and hows of learning to rank for web search. *Information Retrieval*, 16(5):584–628, 2013.

[98] Emine Yilmaz and Stephen Robertson. Deep versus shallow judgments in learning to rank. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 662–663, Boston, MA, USA, 2009. ACM Press.

[99] Wei Chen, Tie-Yan Liu, and Zhi-Ming Ma. Two-layer generalization analysis for ranking using rademacher average. In *Advances in Neural Information Processing Systems*, volume 1, pages 370–378, Vancouver, British Columbia, Canada, 2010.

Curran Associates Inc.

[100] Nicola Ferro and Mark Sanderson. Sub-corpora impact on system effectiveness. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 901–904, Shinjuku, Tokyo, Japan, 2017. ACM Press.

[101] Ingmar Weber and Carlos Castillo. The demographics of web search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 523–530, Geneva, Switzerland, 2010. ACM Press.

[102] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789, Cambridge, United Kingdom, 2017. ACM Press.

[103] Javed A. Aslam, Evangelos Kanoulas, Virgil Pavlu, Stefan Savev, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 468–475, Boston, MA, USA, 2009. ACM Press.

[104] Jose G. Moreno-Torres, Troy Raeder, Roco Alaiz-Rodrguez, Nitesh V. Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521 – 530, 2012.

[105] Jerome H Friedman and Jacqueline J Meulman. Multiple additive regression trees with application in epidemiology. *Statistics in Medicine*, 22(9):1365–1381, 2003.

[106] Kalervo Jrvelin and Jaana Keklinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[107] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*, volume 2. Lawrence Earlbaum Associates, 1988. ISBN 978-0-12-179060-8.

[108] Muhammad Ibrahim and Mark Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Trans. Inf. Syst.*, 34(4):20:1–20:38, August 2016. ISSN 1046-8188.

[109] Peng Cai, Wei Gao, Aoying Zhou, and Kam-Fai Wong. Query weighting for ranking model adaptation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 112–122, Portland, Oregon, June 2011. Association for Computational Linguistics.

[110] Jiayuan Huang, Alexander J Smola, Arthur Gretton, Karsten M Borgwardt, Bernhard Schlkopf, and others. Correcting sample selection bias by unlabeled data. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, volume 19, page 601, Canada, 2006. MIT Press Cambridge.

[111] Jie Peng, Craig Macdonald, and Iadh Ounis. Learning to select a ranking function. In *European Conference on Information Retrieval*, pages 114–126, Milton Keynes, UK, 2010. Springer-Verlag Berlin.

[112] Solomon Kullback. *Information Theory and Statistics.* Courier Corporation, 1997.

[113] Jianhua Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[114] Yasser Ganjisaffar, Rich Caruana, and Cristina Videira Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 85–94, Beijing, China, 2011. ACM Press.

[115] Janez Demar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7(Jan):1–30, 2006.

[116] Isaac Triguero, Salvador Garca, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2015.

[117] Minmin Chen, Kilian Q Weinberger, and John Blitzer. Co-training for domain adaptation. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 2456–2464, Granada, Spain, 2011. Curran Associates Inc.

[118] Anna Margolis. A literature review of domain adaptation with unlabeled data. *Rapport Technique, University of Washington*, pages 1–42, 2011.

[119] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

[120] David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 337–344, Sydney, Australia, 2006. Association for Computational Linguistics.

[121] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100, Madison, Wisconsin, USA, 1998. ACM Press.

[122] Pengfei Li, Mark Sanderson, Mark Carman, and Falk Scholer. On the effectiveness of query weighting for adapting rank learners to new unlabelled collections. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1413–1422, Indianapolis, Indiana, USA, 2016. ACM Press.

[123] Kamal Nigam, Andrew McCallum, and Tom Mitchell. Semi-supervised text classification using EM. *Semi-Supervised Learning*, pages 33–56, 2006.

[124] Valentin I Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D Manning. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 9–17, Uppsala, Sweden, 2010. Association for Computational Linguistics.

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

[126] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[127] Yuichiro Anzai. Kernel density estimators. In *Pattern Recognition and Machine Learning*, pages 122–123. Springer, 2012.

[128] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193, Seattle, Washington, USA, 2006. ACM Press.

[129] Ben Carterette, James Allan, and Ramesh Sitaraman. Minimal test collections for retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 268–275, New York, NY, USA, 2006. ACM.

[130] Johannes Frnkranz and Eyke Hllermeier. Preference Learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 789–795. Springer US, Boston, MA, 2010.

[131] Zhicheng Dou, Ruihua Song, Xiaojie Yuan, and Ji-Rong Wen. Are click-through data adequate for learning web search rankings? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 73–82, Napa Valley, California, USA, 2008. ACM Press.

[132] J. Yu, D. Tao, M. Wang, and Y. Rui. Learning to rank using user clicks and visual features for image retrieval. *IEEE Transactions on Cybernetics*, 45(4):767–779, 2015.

[133] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Balancing exploration and exploitation in learning to rank online. In *European Conference on Information Retrieval*, volume 6611, pages 251–263, Dublin, Ireland, 2011. Springer.

[134] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, Pisa, Italy, 2016. ACM Press.

[135] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. Multileave gradient descent for fast online learning to rank. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 457–466, San Francisco, California, USA, 2016. ACM Press.

[136] Artem Grotov and Maarten de Rijke. Online learning to rank for information retrieval: SIGIR 2016 Tutorial. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1215–1218, Pisa, Italy, 2016. ACM Press.

[137] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015.

[138] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382, Santiago, Chile, 2015. ACM Press.

[139] Hang Li and Zhengdong Lu. Deep learning for information retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1203–1206, Pisa, Italy, 2016. ACM Press.

[140] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International*

ACM SIGIR Conference on Research and Development in Information Retrieval, pages 515–524, Shinjuku, Tokyo, Japan, 2017. ACM Press.

[141] Bhaskar Mitra and Nick Craswell. Neural Models for Information Retrieval. *arXiv preprint arXiv:1705.01509*, 2017.

[142] Tom Kenter, Alexey Borisov, Christophe Van Gysel, Mostafa Dehghani, Maarten de Rijke, and Bhaskar Mitra. Neural networks for information retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1403–1406, Shinjuku, Tokyo, Japan, 2017. ACM Press.

[143] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A Survey on Deep Transfer Learning. In *27th International Conference on Artificial Neural Networks*, Rhodes, Greece, 2018. Springer.