Self-Labeling Methods for Unsupervised Transfer Ranking

Pengfei Li[a,*], Mark Sanderson[a,*], Mark Carman[b,*], Falk Scholer[a,*]

[a] *RMIT University, Melbourne, Australia*
[b] *Politecnico di Milano, Milan, Italy*

Abstract

A lack of reliable relevance labels for training ranking functions is a significant problem for many search applications. *Transfer ranking* is a technique aiming to transfer knowledge from an existing machine learning ranking task to a new ranking task. *Unsupervised transfer ranking* is a special case of transfer ranking where there aren't any relevance labels available for the new task, only queries and retrieved documents. One approach to tackling this problem is to impute relevance labels for (document-query) instances in the target collection. This is done by using knowledge from the source collection. We propose three self-labeling methods for unsupervised transfer ranking: 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 have compared the three algorithms on three large public test collections using LambdaMART as the base ranker and found that (i) all the proposed algorithms show improvements over the original source ranker in different transferring scenarios; (ii) RankPairwiseEM and RankSelfTrain significantly

⋆This manuscript version is made available under the CC-BY-NC-ND 4.0 license `http://creativecommons.org/licenses/by-nc-nd/4.0/`

⋆⋆The formal publication can be accessed via `https://doi.org/10.1016/j.ins.2019.12.067`

*Corresponding author

Email addresses: `vli@vinceli.org` (Pengfei Li), `mark.sanderson@rmit.edu.au` (Mark Sanderson), `mark.carman@polimi.it` (Mark Carman), `falk.scholer@rmit.edu.au` (Falk Scholer)

outperform the source rankers across all environments. We have also found that they 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.

Keywords: Learning to rank, Transfer learning, Ranking adaptation, Transfer Ranking, Information Retrieval, Domain Adaptation

1. Introduction

Ranking is one of the most important components of Information Retrieval (IR) systems (e.g. search engines). Given a search query expressing a particular information need, an IR system needs to rank the documents of a collection in a descending order of relevance to the query. The relevance score of the query and document is usually estimated through a scoring/ranking function that combines a set of features, which may include text match and other document quality features.

Conventional ranking functions for IR systems are outcomes of research that investigate ranking by lexical features that follow certain linguistic heuristics. However, refining such functions requires extensive human effort. Moreover, those ranking functions are usually not optimal for a particular document corpus. For example, several studies [44, 26] have shown that the effectiveness of ranking function vary under different document collections. Learning to rank (L2R) is an effective approach to train IR ranking functions via machine learning techniques. L2R trains a ranking function that can predict the ranking order of a set of retrieved documents for a query. The training is done using example search queries, retrieved answer documents, and corresponding relevance labels. L2R has been widely used in IR applications like Web Search, commerce search systems, and recommender systems.

Most L2R algorithms are supervised, which means they require a substantial number of labels, indicating the relevance for query-document pairs. Specifically, given a query and its retrieved documents, assessors will be asked to give a relevance label for each document to the query. The label could be binary or graded. Note that relevance labels are human-generated labels that reflect the degree of relevance. The optimal ranking order of documents to a query can be inferred from the relevance labels. A ranking algorithm predicts a real-value score (a relevance score). Relevance scores of query-document pairs will be used to rank the documents to approximate the optimal ranking. However, obtaining relevance labels for training L2R models requires expensive and time-consuming human assessment. For example, to build a new web search engine, one needs to obtain the relevance labels of a large volume of queries and retrieved documents. In some other cases, due to the highly personalized task, relevance assessments are not possible. For example, the relevance labels for email search is unlikely to be assessed by another person. A lack of labels has restricted the applicability of L2R in certain scenarios.

Generating cheap relevance judgments via crowd-sourcing [31] or actively selecting partial queries and documents for annotation [19, 33] have been considered as potential solutions for the lack of sufficient labels. However, quality control for relevance judgments can be challenging, and the cost nonetheless expensive.

An alternative approach is to reuse labels drawn from related collections. However, an L2R model trained in one collection may not generalize well to a different collection [41] as the distribution of data in the two collections is different. Transfer learning [40] is a technique that aims to train models for a *target* collection by transferring knowledge from related *source* collections. Transfer learning techniques can potentially be used to solve the lack of relevance label problem for L2R. A rank-focused application of transfer learning is called Transfer Ranking (TR) [32].

However, due to various reasons, conventional transfer learning techniques cannot be used for transfer ranking directly. 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 the consideration of efficiency, documents are pooled at a certain depth, which, however, makes it harder to formalize the data generating process. As a result, the data distribution of an L2R dataset 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 these factors have contributed to the challenge of implementing transfer ranking algorithms.

The transfer settings for TR can be different. If some labels are present in a target collection, then TR can be classified as supervised. Otherwise it is said to be unsupervised, which is the focus of this paper. Past research [22, 32] utilized instance-weighting to tackle unsupervised TR. Weights are assigned to training instances in the source collection to change the data distribution to be more like the distribution in the target. For a given search query from the collection, an L2R approach optimizes a ranking function over the documents. For each query, the ranking function predicts relevance scores for the documents retrieved. Ideally, the resulting rank order of the documents for each query should match the ground-truth ranking that results from ordering documents by their ground-truth relevance judgments. There are multiple ways to assign the weights for each query: to the documents (document-level); to document pairs (pair-level); or to queries, where all documents belonging to the same query will be assigned as the query weight (query weight). The objective of L2R algorithms is to maximize the ranking effectiveness of a ranking function for search queries in a collection. As a result, instance-weighting at query-level (assign instance weights to queries instead of documents) is a natural and more 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[1] for instance-weighting. Li et al. [32] demonstrated that the effectiveness of such algorithms varies substantially across different transfer scenarios.

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* [49] solution is related to self-training [36], co-training [15], and multi-view learning [47] methods, which have also been applied in transfer learning [15]. Co-training is a machine learning technique that trains a model using two different views/feature sets of the data, which usually involves a label imputation step.[2] Multi-view learning is a general case for co-training, where multiple views of the data were used to train the data. By gradually imputing new labels for unlabeled instances in the target collection, the self-labeled algorithm can bypass the difficult problem of density ratio estimation for the L2R collections. All of the mentioned methods are techniques to generate imputed labels for unlabeled data in the collection. A self-training algorithm imputes the labels by the output of the model trained on labeled data.

[1]The relative ratio of the density/frequency of an instance in one distribution compared with its density in another distribution.

[2]The terminology "imputation" usually refers to the technique to compensate for missing data in the machine learning community. In this paper, we introduce the terminology of "label imputation" to refer to the process of imputing missing relevance labels for L2R collections.

In this paper, we propose three different self-labeling techniques: an expectation maximization (EM) based transfer ranking algorithm (RankPairwiseEM), a "hard EM"-inspired transfer ranking algorithm (RankHardLabelEM), and a self-training for transfer ranking 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 using 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-probabilities) assignments of relevance labels to unlabeled training instances, while RankSelfTrain is an application of the self-training algorithm for TR.

While EM and self-training 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 specific document, as well as the confidence of the prediction. Finally, unlike conventional classification or regression algorithms that look to minimize the expected loss for each data point, the effectiveness of a ranking function will be measured on a query-level basis, i.e., the ranking effectiveness of the model on each query. The work in this paper was the first attempt to use this technique to solve unsupervised TR problems.

Notice that, although co-training/multi-view learning algorithms have been shown to be effective in semi-supervised learning tasks, they are not directly applicable to unsupervised TR tasks. By using distinctive feature sets to train different models for the same task, multi-view learning can use different models to fix the mistakes made by individual models. This will increase the quality and confidence of the prediction. However, one needs to make some assumptions regarding the feature sets. On the other hand, self-training algorithms use the model prediction as an approximation to the labels and iteratively improve the model using the approximated labels.

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

- How can one apply 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 transfer ranking 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. We also show that they are not significantly worse than the target model.

The rest of this article is organized as follows: Section 2 describes preliminaries about solutions for unsupervised TR problems and section 3 presents background and related work. In Section 4, we introduce our solution to use EM algorithms to tackle the problem and section 5 explains how self-training algorithms can be used to solve unsupervised TR problems. Section 6 describes our evaluation experiments. The results and further discussions on the answers to our research questions are presented in Sections 7 and 8. Finally, Section 9 summarizes our conclusions and future works.

2. Preliminaries

This section gives the formal definition of the unsupervised TR problem and some preliminary studies on existing solutions for the problem.

Following the notations in Cao et al. [13], let $\mathrm{Q} = \{q_1, q_2, \cdots, q_m\}$ be a set of queries; $d_i{=}(d_{i1}, d_{i2}, \cdots, d_{in})$ be the list of documents associated with query q_i, where d_{ij} is the j^{th} document of query q_i. Furthermore, let $\vec{x}_{ij} = \Psi(q_i, d_{ij})$ 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 q_i to denote the list of document feature vectors corresponding to the query $q_i = \{\vec{x}_{ij}\}_{j=1}^{n}$ and let $\{r_{ij}\}_{j=1}^{n}$ be the list of relevance scores, where r_{ij} denotes the score of the j^{th} document for q_i.

A training example $t_{ij} = (\vec{x}_{ij}, r_{ij})$ consists of a feature vector and a relevance judgment. For ease of expression, we simplify the notation, denoting the set of training examples for each query, i.e., the ranked list, as: $l_i = \{(\vec{x}_{ij}, r_{ij})\}_{j=1}^{n}$. A training dataset consisting of multiple queries with associated relevance judgments is then denoted by $\mathbf{L} = \{l_i\}_{i=1}^{m} = (X, R)$.

Listwise algorithms have been shown to be more effective than the pointwise and pairwise approaches [48] because they directly optimize the query-level effectiveness on a collection:

$$\theta^{\star} = arg \min_{\theta} \; \mathbb{E}_{(q,\vec{r})\sim\mathbf{L}}[\ell(\{f(\vec{x}_j, \theta)\}_{j=1}^{n}, \vec{r})] \tag{1}$$

In Equation 1, $\mathbb{E}$ is the mathematical expectation, θ is the parameters for the ranking function f, $\vec{r}$ is a list of relevance labels, $(q, \vec{r})$ is a ranked list l drawn[3] from $\mathbf{L}$, and ℓ is the query-level loss. An equivalent objective function is to maximize the expected ranking metric scores, e.g., Normalized Discounted Cumulative Gain (NDCG) [27]. NDCG is a rank effectiveness metric that was designed to reflect a user's preferences of seeing more relevant documents at the top of the retrieved list. Cumulative gain (CG) aggregates gains in the number of relevant documents observed when iterating through the ranked list. A rank-based discount function is introduced to the cumulative gain so that the metric places more emphasis on top-ranked documents:

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

Here rel_i denotes the relevance judgment for the i^{th} document in the list and $2^{rel_i} - 1$ is an exponent gain formula used in Burges et al. [9]. The denominator $\frac{1}{\log_2(i+1)}$ is the discount function. There are other gain and discount functions for DCG, with a comparison of different methods discussed in Kanoulas and Aslam [29]. 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 the 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:

[3] $\sim$ denotes that a data is generated from a probability distribution.

$$NDCG = \frac{DCG}{IDCG} \quad (3)$$

where IDCG is the ideal DCG score for the returned documents, when the documents are ranked in descending order according to the relevance labels. A previous study has shown that users tend to be only interested in the top few pages of search results. As a result, a cut-off of the ranking list is commonly used to reflect such behavior. For example, in this paper, NDCG@10 is used to evaluate the NDCG score at cut-off at 10.

In this paper, we focus on transfer ranking algorithms that can work with the listwise L2R algorithms, because listwise algorithms can achieve better ranking effectiveness.

To distinguish between source and target collections in a particular transfer ranking problem, we use the superscripts '*so*' and '*ta*', respectively. Thus, for an *unsupervised* transfer ranking problem, we assume a training set $\mathbf{L}^{so}$, which is composed of a query set Q^{so}, the query-document pairs X^{so}, and corresponding relevance labels R^{so}, and we also assume a target dataset $\mathbf{L}^{ta}$, consisting of queries Q^{ta} and query-document pairs $\mathbf{L}^{ta}$ but for which the relevance labels R^{ta} are unknown. With such data, unsupervised TR aims to train a ranking function f^{ta} for $\mathbf{L}^{ta}$.

2.1. 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* [41], where the conditional probability 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 ($p^{so}(\mathbf{x}) \neq p^{ta}(\mathbf{x})$). A covariate shift can be addressed by re-weighting source samples in such a way that the source distribution approximates the target one. However, for listwise L2R algorithms, training is performed at the query-level (Equation 1). Consequently, instance-weighting is more meaningful and natural at the query-level rather than at 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[4] used during training tends to follow the desired loss function on the target collection.

In Li et al. [32], it was 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 other settings unchanged. The details of the datasets can be referred to the Section 6.1. Figure 1 shows the effectiveness of a query-weighted LambdaMART (wλMART)[5] based on the Kullback-Leibler Importance Estimation Procedure (KLIEP) [32], measured with NDCG@10, when it was trained with different sizes of source queries pooled from MSLR[6] and tested on LETOR4.0. The settings of the transfer are similar to Li et al. [32], except that the test set is used for density ratio estimation.

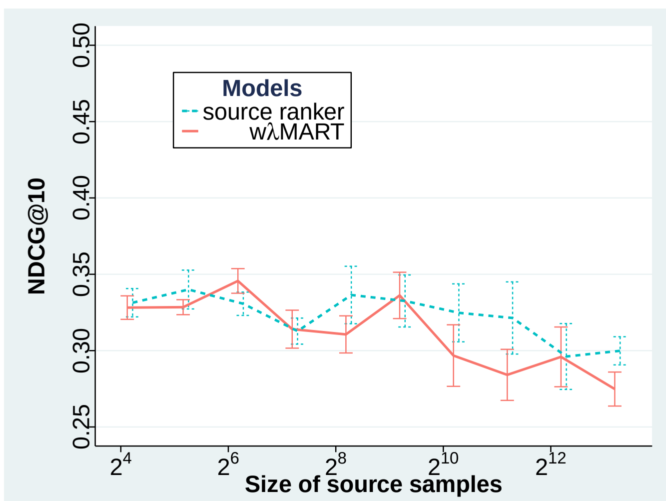


Figure 1: Effectiveness of wλMART versus source sample size

The results in Figure 1 show that the effectiveness of the source ranker on the target dataset varies across training samples and degrades with the

[4]A loss function is a function to quantify the difference between the ground-truth labels and the predictions of a model.

[5]The specific algorithm used here was document-level-weight-aggregation: kliep.doc.

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

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).

Thus far, we have seen that the performance of instance-weighting can be unreliable. Two factors can be the cause of the 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, in the standard learning-to-rank setup, the learned ranking function actually only *re-ranks the top-k documents* pooled by an initial base ranker. As a result, even if only a covariate shift is present, the *resulting* conditional distribution will likely be different across the source and target collections.

3. Related Work

In this section, we review the fundamentals of learning to rank, unsupervised TR algorithms, as well as related work on self-labeling.

3.1. Learning to Rank

For the sake of efficiency, modern IR systems usually first retrieve a pool of candidate documents that contain certain keywords in the search query from the document corpus, using an inverted index. A conventional retrieval model will be used as a base ranker to initialize the ranking order of the retrieved documents, and the models are usually light in computation. For example, BM25 [43] is a widely used IR retrieval model. One of the most famous variants of BM25 is the ATIRE BM25 [50], which can be computed as:

$$BM25(d, q, C) = \sum_{t \in q} \log \frac{N}{df(t)} \frac{(k_1 + 1) \cdot tf(t, d)}{k_1 \cdot (1 - b + b \cdot \frac{c(d)}{avg\ c(d)}) + td(t, f)} \tag{4}$$

where N is the number of documents, $df(t)$ is the number of documents containing the term t, $tf(t, d)$ is the frequency of term t in document d, $c(d)$ 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. However, the ranking order of documents by the base ranker is usually not optimal for a particular document collection. Past research [24] has shown that users tend to only look at the top-ranked search results. As a result, ranking optimization is required for IR systems.

An L2R system is a set of algorithms that use machine learning techniques to solve ranking problems. Given a set of queries and their corresponding documents retrieved by a conventional retrieval model, the objective of L2R is to optimize a ranking function that can predict the optimal permutations of the document lists according to their relevance to queries. The relevance of a document to a query is given by a human-generated relevance label assigned to the pair, typically on a binary or graded relevance scale [30]. L2R algorithms can be classified into one of three categories according to their optimisation level: *pointwise* algorithms [16] aim to minimize the loss at individual document level (i.e. the loss between the observed relevance label and the predicted label on each document), *pairwise* algorithms [25] aim to minimize the pairwise-preference loss at the document-pair level (i.e. the loss between the observed ordering and the predicted ordering on each pair of documents), and *listwise* algorithms [8, 13] aim to minimize the query-level loss over the predicted ranking as a whole (i.e. the loss, measured by IR ranking metrics, between the ground-truth ranking, and the predicted ranking of a set of retrieved documents for the query).

3.2. Unsupervised TR

Training a reliable and robust learning to rank algorithm that can generalize to a ranking task, requires a massive number of relevance labels. However, obtaining the relevance labels for training L2R is expensive, techniques such as crowd-sourcing [4] have been used in the past. Active learning algorithms [33] is a machine learning technique that can selectively choose training instances for label annotation as well as for training. It has been shown that, by using a small set of training instances that are representative and informative for training, one can obtain an accurate prediction model with minimum cost. Active learning methods were also applied in attempts to solve the lack of labels problem for L2R. For example, Mehrotra and Yilmaz [38] proposed to select a subset of search queries following two criteria: informativeness and representativeness. Informativeness of the query selection process is instantiated by choosing the query with the lowest certainty scores, measured by a ranking probability computed using a committee of ranking functions trained with a random sample of already labeled queries. By doing so, the queries with the largest uncertainty will be selected for labelling. Representativeness of the query selection is instantiated by selecting queries that are topically similar to the large volumes of unlabeled data in the collection. The algorithm showed some performance uplift on a small L2R test collection compared with previous approaches. One challenge for active learning to rank is that one has to manage the selection of both the queries and the documents.

Additionally, semi-supervised L2R [20] looks to leverage unlabeled data in the collection using only a small number of already labeled target instances.

Transfer learning, including domain adaptation and multi-task learning, has successfully been applied to many classification and regression problems. Domain adaptation is a transfer learning technique that applies when the source and target datasets are from different domains. An example would be adapting a spam classifier for the IT domain to the medical domain. Multi-task learning is a series of techniques that simultaneously train multiple models for different tasks by sharing commonalities among those tasks. Most of these algorithms have investigated methods for modifying the source data (via some form of weighting) to make its distribution as similar to that of the target data as possible. Solutions for minimizing the difference between the source and target data distribution include *sample-based* methods, *feature-based* methods, and *miscellaneous* methods. Sample-based transfer learning algorithms train on weighted (or selected) training instances from the source collection, such that the weighted data approaches the data distribution in the target collection [46, 28], while feature-based methods conduct a similar task by training on subsets or weighted versions of the features (using latent feature spaces), such that the divergence in distribution between the modified source data and the target data is minimized. There are also numerous miscellaneous methods, such as self-labeling [15], which impute labels for unlabeled data from the target collection, and co-regularization [34], which optimizes the model by regularizing the similarity between the source and target tasks, which can be used to perform knowledge transfer in certain scenarios.

Most of the algorithms discussed above apply to TR problems. However, due to the difficulty of formalizing the concept of the "query space distribution" for L2R datasets, most of these algorithms have only been applied to pairwise algorithms, since the objective of L2R algorithms is to maximize the query-level ranking effectiveness. A more natural and effective mechanism is to attempt to minimize the query distribution[7] divergence, i.e., the differences in the probability distribution of the queries.

Increasingly, researchers study Transfer Ranking (TR) in both transfer learning unsupervised forms [11, 22, 32]. In unsupervised TR, instance-weighting has been used as the transferring process [11, 22, 32]. Most unsupervised TR solutions assume that the difference between the source and target collection only exist in the input feature space, and it is usually referred to as the *Covariate shift* problem. 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 covariate shift in ranking problems. Intuitively, instance weighting algorithms assign weights to the training samples in the source collection to make the data distribution more like the data distribution of the target collection. Through optimizing the cost function over the weighted samples, the algorithm can help improve the generalization on the target collection.

As discussed before, the training data for L2R are used in different ways. As a result, and therefore the instance weighting for L2R can be conducted at three different levels, i.e., document level, pair level, and query level. In Gao et al. [22], the authors generated instance weights at different levels for L2R datasets. 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[8]) on the six topic sets in LETOR3.0 and showed some significant improvements. Cai et al. [10] further improved the algorithm by classifying the queries directly. The algorithms were tested on a set of small datasets and showed only limited improvements in ranking effectiveness.

An importance-weighted AdaRank approach was proposed by Ren et al. [42]. The authors used the Kullback-Leibler Importance Estimation Procedure (KLIEP) [46] 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. Li et al.

[7]Query distribution refers to the probability distribution of search queries.

[8]AdaRank and LambdaMART are more effective [48].

[32] showed that the effectiveness of instance-weighting cannot be generalized to different transferring scenarios due to the inaccuracy of density-ratio estimates for queries. More details on why instance-weighting is problematic for unsupervised TR have been discussed in section 2.1.

3.3. Self-Labeling Algorithms

An alternative approach to unsupervised transfer learning is self-labeling [35]. 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. [49] 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 [17], self-training algorithms [37], and multi-view learning [47], which includes co-training [7]. All three solutions were originally utilized for semi-supervised learning, but have been extended to unsupervised transfer learning by Chen et al. [15]. In Chen et al. [15], the authors developed an algorithm called CODA (Co-training for Domain Adaptation) that uses a co-training method to adapt review sentiment classifiers across different domains. The objective of a sentiment classifier is to determine whether a review for a product is positive or negative. The CODA algorithm iteratively imputes sentiment labels for unlabeled reviews according to the current model's confidence score on the data. More specifically, at each iteration, CODA trains a classifier using labeled data, which includes data with imputed labels. As the algorithm was designed for domain adaptation, the model was initially trained with the source data only. Provided with predicted labels and the confidence of the model on the prediction, CODA then decides on which imputed labels to add to the training set. Moreover, a feature weighting process is applied during the iterations to ensure that the algorithm focuses on features that have commonalities among different domains. The performance of CODA was evaluated on the "Amazon reviews" benchmark data sets, which have four different domains for sentiment classification adaption. Results show that CODA can significantly outperform other domain adaption algorithms, even when there are no relevance labels from the target collection.

Preliminary works investigating self-training ideas in unsupervised transfer ranking scenarios 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). It is worth mentioning that most publicly available L2R collections do not have information on the original queries and documents. The only available resources are the extracted and normalized features for query-document pairs in the collection. 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 the of work of Goswami et al. [23]. In our solution, we only make use of the extracted features from the query-document pairs instead of the raw text features. One of the reason for this is that, in most publicly available test collections, the extracted features are the only provided information of a document. In some datasets, the details of the features are also unknown. Indeed they are not even directly comparable given that they are tackling different problems with different (and in their case more specific) assumptions.

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) [6]. The assumption in PRF is that the top-k retrieved documents for a query are relevant documents to the query, and they can be used to exploit more relevant documents from the corpus. 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.

Existing solutions to solve the unsupervised TR, which are mostly based on instance weight, have shown their weakness in their reliability under different transfer scenarios. The most important reason why these algorithms don't work well in practice is because of the difficulties of measuring the similarities between different L2R collections in order to quantify the changes in distribution. Self-labeling, on the other hand, does not require any process for estimating the changes in L2R data distribution, and has shown to be effective for solving other related problems. As a result, self-labeling methods for unsupervised TR constitute a promising approach that deserves further investigation.

4. EM for Unsupervised TR

A widely used self-labeling approach in the machine learning community is the Expectation-Maximization (EM) algorithm. The EM algorithm is a process used to estimate the parameters of a statistical model that is controlled by some hidden (i.e. unobserved) variables. It has therefore been widely studied and applied for training semi-supervised models when there is an absence of adequate labels [39]. The EM algorithm can potentially be used for solving TR problems because of its ability to leverage unlabeled training data.

The EM algorithm generates 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:

$$\theta^{(t)} = arg\, \underset{\theta}{max} \sum_{Y} p(Z|X, \theta^{(t-1)}) \log\ p(X, Z|\theta) \tag{5}$$

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

4.1. EM algorithm for TR with Pairwise Preferences

In this section, we apply a modified EM algorithm to tackle the TR problem. The implementation of the EM algorithm for TR (RankPairwiseEM) is present in Algorithm 1. 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 an 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 ($\gamma(\mathbf{x}, \theta) \mapsto r$). The mapping function can be decomposed into two functions: 1) a scoring function that estimates a similarity score[9] for a query-document pair; and 2) a (possibly random) assignment function that maps each query-similarity score to a relevance label.

Estimating $p(R|X, \theta)$ requires making strong assumptions about how scores map to relevance labels. We can avoid this issue by using the pairwise ranking preferences as the hidden values instead. The pairwise probability of a document pair $\{d_{ij}, d_{ik}\}$ can be estimated using a logistic function as in Burges et al. [9]:

$$p(r_{ij} > r_{ik}) = \frac{1}{1 + e^{-\sigma \ \Delta s_{ijk}}} \tag{6}$$

Here σ is a parameter controlling the shape of the logistic function[10], $\Delta s_{ijk} = s_{ij} - s_{ik}$ 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 $X^2 = \{(x_{ij}, x_{ik})\}_{i,j<k}$ $s.t.$ $r_{ij} \neq r_{ik}$, 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:

$$\omega_{ijk}^{(t-1)} = p(r_{ij} > r_{ik}|\theta^{(t-1)}) = \frac{1}{1 + e^{-\sigma \ \Delta s_{ijk}^{(t-1)}}} \tag{7}$$

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

[9]The output of a ranking function is a similarity score between a query and a document.

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

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 [8], which learns a boosted regression tree model for ranking and has been shown to be highly effective [48].

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

$$f(\vec{x}) = \sum_{l=1}^{L} \alpha_l \ h_l(\vec{x}; \theta_l) \tag{8}$$

where $f(.)$ is the trained ranking function, h_l is the l^{th} regression tree, θ_l are the parameters for the regression tree, α is the weight for the regression tree.

On 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 6. 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:

$$C_{ijk} = |\Delta Z_{ijk}|(I_{[r_{ij} > r_{ik}]} \log(1 + e^{-\sigma \ \Delta s_{ijk}^{(l-1)}}) + I_{[r_{ij} < r_{ik}]} \log(1 + e^{\sigma \ \Delta s_{ijk}^{(l-1)}})) \tag{9}$$

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 $\vec{x}_{ij}$ 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:

$$\lambda_{ij} = \frac{\partial C_{ij}}{\partial s_{ij}^{(l-1)}} = \sum_{k:k \neq j} |\Delta Z_{ijk}|(I_{[r_{ij} > r_{ik}]} \frac{-\sigma}{1 + e^{\sigma \ \Delta s_{ijk}^{(l-1)}}} - I_{[r_{ij} < r_{ik}]} \frac{-\sigma}{1 + e^{-\sigma \ \Delta s_{ijk}^{(l-1)}}})) \tag{10}$$

According to Burges [8], the value of the k^{th} leaf in the l^{th} tree is then updated using a second-order approximation:

$$\gamma_{km} = \frac{\sum_{d_{ij} \in R_{km}} \frac{\partial C_{ij}}{\partial s_{ij}^{l-1}}}{\sum_{d_{ij} \in R_{km}} \frac{\partial^2 C_{ij}}{\partial (s_{ij}^{l-1})^2}} = \frac{\sum_{d_{ij} \in R_{km}} \lambda_{ij}}{\sum_{d_{ij} \in R_{km}} \frac{\partial \lambda_{ij}}{\partial s_{ij}^{l-1}}} \tag{11}$$

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:

$$\mathbb{E}[C_{ij}] = \sum_{k:k \neq j} |\Delta Z_{ijk}| (\omega_{ijk}^{(t-1)} \log(1 + e^{-\sigma \ \Delta s_{ijk}^{(t,l-1)}}) + \omega_{ikj}^{(t-1)} \log(1 + e^{\sigma \ \Delta s_{ijk}^{(t,l-1)}})) \tag{12}$$

where ω_{ijk} and ω_{ikj} are probabilities computed using Equation 7, 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:

$$\mathbb{E}[\lambda_{ij}] = \sum_{k:k \neq j} \mathbb{E}[|\Delta Z_{ijk}|] (\frac{-\omega_{ijk}\sigma}{1 + e^{\sigma \ \Delta s_{ijk}^{(t,l-1)}}} - \frac{-\omega_{ikj}\sigma}{1 + e^{-\sigma \ \Delta s_{ijk}^{(t,l-1)}}}) \tag{13}$$

In this paper, we use NDCG@10 as the training metric for LambdaMART (i.e. $Z = NDCG@10$). Later in the paper, NDCG at cut-off 10 is also used as the evaluation metric for the experiments. For other optimization objectives, Z can be replaced by the expected metric for optimization. Because the relevance labels and the ranking orders of documents are unknown, we need to compute the expected $|\Delta NDCG@10|$[11] 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: $\mathbb{E}[r_{ij}] \approx s_{ij}^{(t-1)} = f(\vec{x}_{ij}; \theta^{(t-1)})$.

$$\mathbb{E}[|\Delta NDCG@10_{ijk}|] = \frac{2^{\mathbb{E}[r_{ik}]} - 2^{\mathbb{E}[r_{ij}]}}{IDCG} \times (\frac{1}{\log_2(\pi_{ij}^{(t,l-1)} + 1)} - \frac{1}{\log_2(\pi_{ik}^{(t,l-1)} + 1)}) \tag{14}$$

where $\pi_{ij}^{(t,l-1)}$ denotes the rank of the j^{th} document for query i, according to the scoring function $f(x_{ij}; \theta^{(t,l-1)})$. The ground truth labels for the docu-

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

ments for the queries are unknown, and therefore we use the similarity score predicted in the last iteration as the label for estimating IDCG. As a result, IDCG is calculated as:

$$IDCG = \sum_{g=1}^{10} \frac{2^{s^{(t-1)}_{i\pi^{-1}(g)}} - 1}{\log_2(g+1)} \tag{15}$$

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:

$$\mathbb{E}[\gamma_{km}] = \frac{\sum_{d_{ij} \in R_{km}} \mathbb{E}[\lambda_{ij}]}{\sum_{d_{ij} \in R_{km}} \frac{\partial \mathbb{E}[\lambda_{ij}]}{\partial s_{ij}^{(t,l-1)}}} \tag{16}$$

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

In line 2 of Algorithm 1, the parameters are initialized by training a LambdaMART with source data:

$$\hat{\theta}^{(0)} = \arg\min_{\theta} \sum_{q_i \in Q^{so}} \sum_{d_{ij} \in q_i} C_{ij} \tag{17}$$

In the E-step (line 4 to 9), 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 7.

In the M-step (line 10 to 14), the parameters are re-estimated with the expected LambdaMART together with the labeled source data:

$$\hat{\theta}^{(t+1)} = arg\ \underset{\theta}{min} \sum_{q_i \in Q^{so}} \sum_{d_{ij} \in q_i} C_{ij} + \sum_{q_i \in Q^{ta}} \sum_{d_{ij} \in q_i} \mathbb{E}[C_{ij}] \tag{18}$$

The algorithm repeats the E-step and M-step until the parameters converge, or until the maximum iteration Γ is met. In practice, we have found that the performance of the algorithm reaches its peak after a few iterations and then it fluctuates within a small region. The parameter Γ is used to terminate the process early for efficiency consideration.

Input: Source queries Q^{so} and judgements R^{so}, target queries Q^{ta}, max iterations Γ, τ threshold ϵ
Output: Ranking function f

RankPairwiseEM(Q^{so},R^{so},Q^{ta},Γ)
  Train ranker $f^{(0)}$ using (Q^{so},R^{so}) with Eq. 17;
  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. 7;
    end
    `/* M-step */`
    Train $f(\mathbf{x}; \theta^{(t)})$ using pairwise probs, Eq. 18;
    if $\theta^{(t)} == \theta^{(t-1)}$ **then**
      return $f^{(t-1)}$;
    end
  end
  return $f^{(t)}$;

Algorithm 1: Label-imputation via RankPairwiseEM

4.2. EM for TR with "Hard" Assignment

It has been shown that, in certain cases, an EM algorithm with a hard deterministic label assignment can be more efficient and effective than the original EM algorithm for particular tasks [45]. 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. The RankHardLabelEM algorithm is given in Algorithm 2.

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 the 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

Input: Source queries Q^{so} and judgements R^{so}, target queries Q^{ta}, stopping threshold ϵ, max iteration Γ
Output: Ranking function f

1 **RankHardLabelEM**(Q^{so},R^{so},Q^{ta},ϵ,Γ)
2   Train ranker $f^{(0)}$ using (Q^{so},R^{so}) with Eq. 17;
3   **for** $t \in \{1, ..., \Gamma\}$ **do**
        `/* E-step */`
4   Calculate scores for all query-doc pairs;
5   Sort query-doc pairs by decreasing score;
6   Label top $k\%$ as relevant, remainder irrelevant;
        `/* M-step */`
7   Train $f(\mathbf{x}; \theta^{(t)})$ using Eq. 19;
8   **if** $\theta^{(t)} == \theta^{(t-1)}$ **then**
9   **return** $f^{(t-1)}$;
10   **end**
11   **end**
12   **return** $f^{(t)}$;

Algorithm 2: Self-labeling via RankHardLabelEM

documents, the ranker is most confidential and it tends to be better for model transferring. 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 with both the labeled source data and unlabeled target data, together with the imputed relevance labels:

$$\hat{\theta}^{(t+1)} = arg\,\min_{\theta} \sum_{q_i \in Q^{so}} \sum_{d_{ij} \in q_i} C_{ij} + \sum_{q_i \in Q^{ta}} \sum_{d_{ij} \in q_i} \hat{C}_{ij}(\hat{R}^{(t)}) \qquad (19)$$

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.

For RankHardLabelEM (Algorithm 2), the algorithm first trains a source ranker with the labeled query document pairs from the source collection together (line 2). In the E step (line 4 to 6), the algorithm will compute the similarity scores for all query-document pairs and label the top $k\%$ pairs as relevant documents, and the remaining documents as irrelevant. The ranking function will be updated in the M step (line 7 to 9) by training a new ranking function with the labeled source data and the target data together with their imputed labels. The process runs iteratively until the imputed labels stop changing or until the maximum iteration count is reached.

5. Self-training for unsupervised TR

Apart from EM algorithms, self-training [1] is another approach to generate imputed labels for unlabeled data in the collection to improve the training process. Self-training is a form of semi-supervised learning [1, 36], with applications in natural language processing [36, 37] and transfer learning [15]. 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 retained from the previous iteration. In each subsequent iteration, the algorithms simply add the next documents to the relevant set on which it is most confident. The implementation of the self-training algorithm (RankSelfTrain) is shown in Algorithm 3.

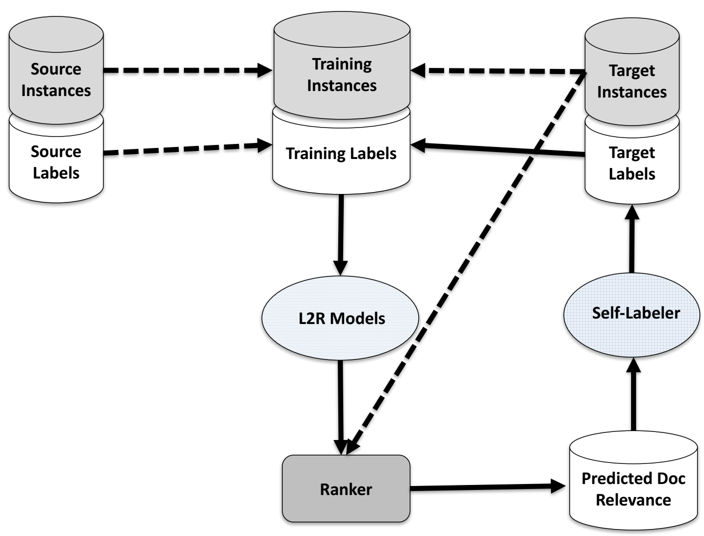


Figure 2: RankHardLabelEM & Self-labeling Paradigm

So the self-training algorithm (RankSelfTrain) gradually increases the number of imputed relevant documents via an iterative process. Both RankHard-

LabelEM and RankSelfTrain follow the self-labeling paradigm demonstrated in Figure 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 assign labels for all the unlabeled target instances. With the newly updated labels, the algorithm updates the ranker and conducts the self-labelling again. The process runs iteratively until convergence is reached.

The difference between the RankHardLabelEM and the RankSelfTrain algorithms lies in the fact that, once a document has been added to the imputed relevant set, the label will not change in the next iteration.

Unlike the RankHardLabelEM algorithm, which updates imputed labels iteratively, the RankSelfTrain gradually adds confident labels to the training set. By gradually adding a small number of accurate predictions, it is expected that the self-trained ranker will update itself toward a ranking function that can generalize to the target collection.

A confidence score is needed to allow label prediction. It is possible for some classification algorithms to produce such scores; for example, logistic regressions can output a probability for a class label. However, it is not straightforward for ranking algorithms to produce such probabilities. [12] We therefore developed 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 of the labels.

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

$$p(r=1|s=\alpha) = \frac{p(r=1)p(s=\alpha|r=1)}{\sum_{v\in\{0,1\}} p(s=\alpha|r=v)p(r=v)} \tag{20}$$

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)[5] on a collection, while the prior probability $p(r=1)$ is estimated by the percentage of relevant documents in the collection:

[12] RankSVM [25] 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.

$$p(r=1) = \frac{|relevant\ documents|}{|documents|} \tag{21}$$

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 labels in some source collections are multi-graded, we regard all the documents as relevant if their relevance labels are larger than zero. 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:

$$p^{ta}(s=\alpha|r=1) \approx p^{ta}(s=\alpha|\hat{r}=1) \tag{22}$$
$$p^{ta}(s=\alpha|r=0) \approx p^{ta}(s=\alpha|\hat{r}=0) \tag{23}$$

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

As the imputed labels are gradually added, directly estimating the prior probability $p(r=1)$ with the imputed labels is unreliable. On the other hand, the prior probability of the target collection can be different from the source collection. Instead, we propose a Dirichlet smoothed estimation that can balance the impact of the source and the imputed labels from the target adaptively:

$$\begin{aligned} p^{ta}(r=1) &\approx \frac{\sum_i \mathbb{I}(\hat{r}=1) + \mu p^{so}(r=1)}{|\hat{r}|+\mu} \\ p^{ta}(r=0) &\approx \frac{\sum_i \mathbb{I}(\hat{r}=0) + \mu(1-p^{so}(r=1))}{|\hat{r}|+\mu} \end{aligned} \tag{24}$$

where μ is set to be half of the number of training instances in the target collection. As a result, probability can be estimated:

$$p^{ta}(r=1|s=\alpha) = \frac{p^{ta}(r=1)p^{ta}(s=\alpha|\hat{r}=1)}{\sum_{v\in\{0,1\}} p^{ta}(s=\alpha|\hat{r}=v)p^{ta}(r=v)} \tag{25}$$

In line 2 of the Algorithm 3, a source ranker f^0 is initially 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 (line 7 to 8), the algorithm calculates the relevance probability for each query-document pair, the probability that a document is relevant to a query, via Equation 20 with probabilities in the source data. If the probability of a relevance label for a given pair is larger than the threshold, the query-document pair will be added to the labeled document set (line 12 to 15). 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 19 (line 20). In the following iterations, the algorithm will continue to compute the probabilities via the imputed labels from the target collection using Equation 25, conduct the labeling and update the ranker iteratively until no more confident labels can be added, or until the maximum iteration is met, where maximum iteration is a pre-set parameter. At the end of the iterations, if only a small number of relevance labels remain, the algorithm will continue updating and train very similar models while only introducing a very small number of target data to the training set. At the same time, training LambdaMART algorithm is very expensive. As a result, the maximum iteration threshold is applied to reduce the computational cost.

6. Data and Methods

6.1. Datasets

Three public L2R test collections are used in our experiments: MSLR, LETOR4.0, and the Yahoo! Learning to Rank (Yahoo! L2R) dataset. Details of these collections are presented in Table 1.

LETOR4.0[13] was built using the million query tracks [2, 3] 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.

The Microsoft learning-to-rank dataset (MSLR)[14] is a large L2R test collection developed based on Bing's retired collections. MSLR contains two collections, namely MSLR-30K and MSLR-10K. MSLR-10K is composed of

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

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

Input: Source queries Q^{so} and judgements R^{so}, target queries Q^{ta}, confidence threshold η

Output: Ranking function f

SelfTrain(Q^{so},R^{so},Q^{ta},η)
  Initialize set of *labeled docs* to be empty: $\Omega^{(0)} = \emptyset$;
  Train ranker $f^{(0)}$ using (Q^{so}, R^{so}) with Eq. 17;
  for $t \in \{1, ...\}$ **do**
    Calculate similarities for all query-doc pairs;
    foreach *unlabeled pair* $x_{ij} \notin \Omega^{(t-1)}$ **do**
      if t==*1* **then**
        Compute $p(r_{ij}|s_{ij})$ following Eq. 20;
      else
        Compute $p(r_{ij}|s_{ij})$ following Eq. 25;
      end
      if $p(r_{ij} = 1|s_{ij}) > \eta$ **then**
        Add $(x_{ij}, 1)$ to $\Omega^{(t)}$;
      else if $p(r_{ij} = 0|s_{ij}) > \eta$ **then**
        Add $(x_{ij}, 0)$ to $\Omega^{(t)}$;
    end
    if $(|\Omega^{(t)}| - |\Omega^{(t-1)}|) == 0$ **then**
      return $f^{(t-1)}$;
    end
    Train ranker $f^{(t)}$ using Eq. 19;
  end

Algorithm 3: Self-training for Ranking

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)[15] [14] 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

[15] `https://webscope.sandbox.yahoo.com/catalog.php?datatype=c`

Table 1: Statistics of public L2R datasets

Collection	Corpus	Query Set	#Queries	#Features
LETOR 4.0	Gov2	MQ2007	1,692	46
		MQ2008	784	46
MSLR	Web	10k	10k	136
	Web	30k	30k	136
Yahoo	Web	Set 1	20k	700
		Set 2	6k	700

market. Set 1 has 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[16] features respectively in the two collections, with some overlap. All the features are rank-normalized as:

$$\tilde{x}_i := \frac{1}{n-1}|\{j, x_j < x_i\}| \tag{26}$$

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

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 we 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.[17]

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

[17] The features in LETOR 4.0 were normalized via a **query-level feature normalization method** [12]. For all the documents belonging to the same query, a min-max normalization is applied to every feature. In this work, we conducted normalization for all the test collections. It turned out that conducting feature normalization, in the same

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 2. All reported results are averages over the five-fold cross-validation.

Table 2: Transfer Settings for testing different algorithms

	LETOR 4.0			MSLR-LETOR4.0			Yahoo! L2R		
	Collection	Queries	Features	Collection	Queries	Features	Collection	Queries	Features
Source	MQ2007	1,692	46	LETOR 4.0	2,476	45	Set 1	19,944	415
Target training	MQ2008	627	46	MSLR	8k	45	Set 2	5,064	415
Target testing	MQ2008	157	46	MSLR	2k	45	Set 2	1,266	415
Source	MQ2008	784	46	MSLR	10k	45	Set 2	6,330	415
Target training	MQ2007	1,353	46	LETOR 4.0	1,980	45	Set 1	15,955	415
Target testing	MQ2007	339	46	LETOR 4.0	496	45	Set 1	3,989	415

6.2. Setup and Measurements

The RankLib 2.1. implementation of LambdaMART was used as the base ranker.[18] 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.[19]

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 RankHard-LabelEM algorithm. For the RankSelfTrain algorithm, the threshold on con-

way, can lead to a better generalization for another collection.

[18]http://sourceforge.net/p/lemur/wiki/RankLib/

[19]http://www.ms.k.u- tokyo.ac.jp/software.html

fidence was set at 95%. The σ for pairwise probability was set as 1 in the RankPairwiseEM algorithm.

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. [32]. 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 [46].
- **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.
- **RankSelfTrain:** Self-training-based algorithm, using LambdaMART as the base ranker.

All models were evaluated using normalized discounted cumulative gain (NDCG) [27], with a rank cut-off of 10. Statistical significance was tested using a two-tailed paired t-test, with a threshold of 0.05.

7. Results and Discussion

The experimental results are presented and discussed below.

7.1. Effectiveness of Self-Labeling Methods

We compared 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 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 3 and 4. 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.

Table 3: 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%) ↑†**

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 larger sample, MQ2007, to the smaller sample, MQ2008, most of the unsupervised TR methods, including wλMART, did not show significant improvements, except 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 to 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 seen 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.

Table 4: 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.Set1-Yahoo.Set2	Yahoo.Set2-Yahoo.Set1
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%)** ↑†

Cross-domain transfers Transferring between MSLR and LETOR4.0 is the first cross-domain transfer scenario. As explained earlier, 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 obtained by Li et al. [32], 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 because 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 on 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. The assumption TR makes, is that the source collection has abundant training data with relevance labels, which can train a well-generalized ranking function for the source collection.

Transferring between Yahoo! L2R Set 1 and Set 2 is more difficult, because of the cross-language setting. Moreover, because Set 1 has a larger query set than Set 2, the generalization of the source model is relatively good compared with others. As a result, TR can be challenging to even compete with the source model. On the other hand, Set 2 is too small compared with Set 1, and therefore transfer from a smaller set to a large set can be difficult too. When transferring from Set 1 to Set 2, the effectiveness of 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 Set1), all the algorithms can significantly outperform λMART.source. For this particular task, the generalization gap between the source and large collection is smaller compared with other scenarios, even though the document corpus of Set 1 and Set 2 are from different countries with different languages. For instance: 1) the features for Yahoo! L2R datasets have been normalized using rank-normalization, and therefore the differences in the data distribution of the input feature spaces are smaller; 2) the sample size of both the source and target collection are larger than other cases, which in effect, reduces the variations in the query distribution; 3) although the tasks are cross-lingual, the features used for ranking are independent of languages. For example, term frequency is only the counts of a query term appearing in a document, which will, in most cases, not be affected by language.

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

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 was.

We compare the effectiveness of unsupervised TR algorithms using average-rank-based visualization [32]. The average rank of all the systems over all the folds in the different collections is computed, and shown in Figure 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 [18], which is used to determine whether there is a significant difference between the average rank of any two systems. The Nemenyi test is used to determine whether there is a significant difference between the average rank of any two systems. It can be performed after first checking with the Friedman test [21] (a non-parametric alternative to repeated measures ANOVA) that the systems are not independent of rank (across the datasets).

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 CD is computed as:

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{27}$$

where k is the number of the algorithms, N is the number of datasets, and q_α is the confidence level of the test, which can be computed with a Studentised range statistics, divided by $\sqrt{2}$.

The results of the tests are displayed in Figure 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 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 there was no significant difference from λMART.target. RankSelfTrain is also the most effective algorithm compared to all the other self-labeling methods.

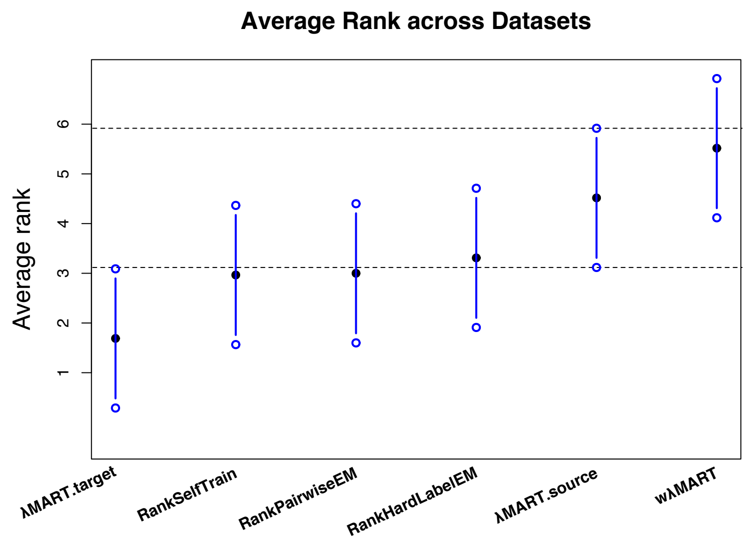


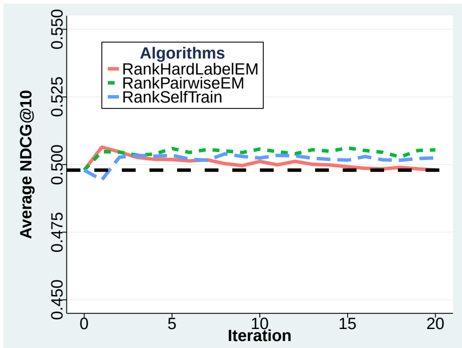
Figure 3: Plots of average rank across the 6 test environments for the 6 different Transfer Learning techniques and the λMART.source and baseline λMART.target system (where no TR was applied). The lower the rank the better performance of the approach. The critical distance (CD) for the Nemenyi test (at the 5% confidence level)

Interestingly, wλMART appears less effective than the λMART.source, which differs from what was found in previous research. The reason for this is possibly that the difference between the feature distributions has been reduced by performing query-level feature normalization on the MSLR dataset. 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.

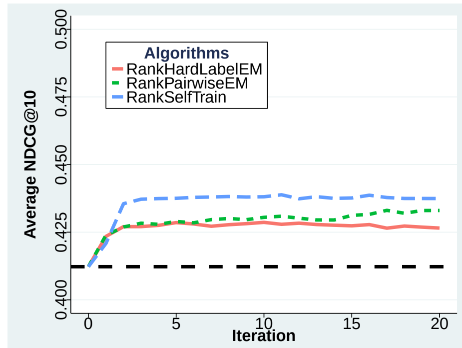
7.3. Analysis of Self-Labeling Methods

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 4. The learning curves presented are averaged over the five runs.

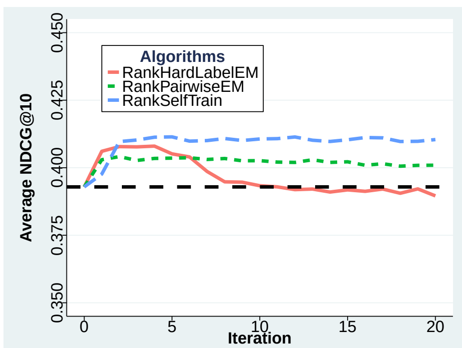
The x-axis in the figure represents the number of the iterations, starting from the 0^{th} iteration (where the source ranker was applied). The y-axis is the average performance of the rankers tested on the target training set, which is the unlabeled target set used for training, together with their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



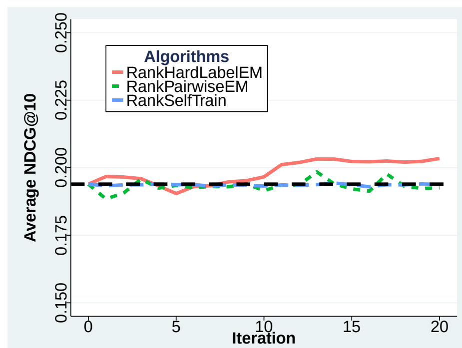
(a) LETOR4.0 MQ2007 to MQ2008



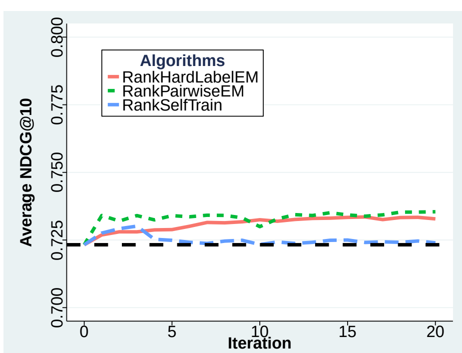
(b) LETOR4.0 MQ2008 to MQ2007



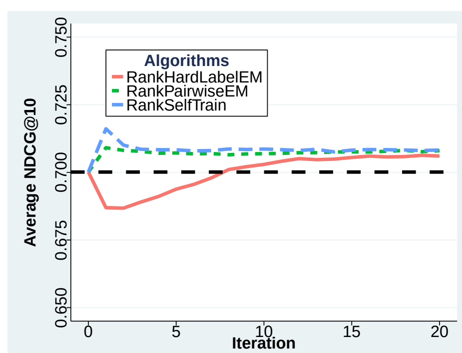
(c) MSLR to LETOR4.0



(d) LETOR4.0 to MSLR



(e) Yahoo Set 1 to Set 2



(f) Yahoo Set 2 to Set 1

Figure 4: Performance vs iteration curve of different self-labeling methods under various settings.

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).

When transferring from LETOR4.0 to MSLR, none of the algorithms have performed as expected. We argue that 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 similarity between the source and target collections, as well as the quality of the source collection have an impact on the effectiveness of an unsupervised TR algorithm. When the source collection is similar to the target collection, TR is not required. Under those circumstances, a good unsupervised TR algorithm should at least not harm the performance of the original source model. As a result, RankPairwiseEM and RankSelfTrain tend to be more reliable than the RankHardLabelEM algorithm.

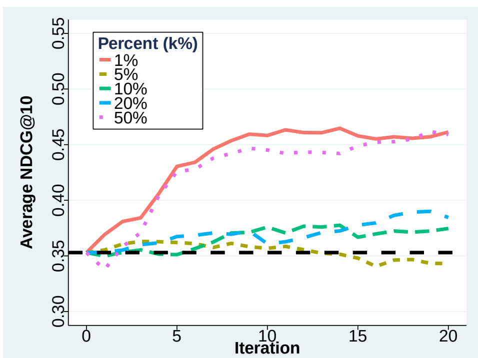
However, the performance of different algorithms is limited by the parameter selection. In the following section, the impact of the parameters on the performance of the algorithms will be analyzed.

8. Sensitivity of Parameter Settings

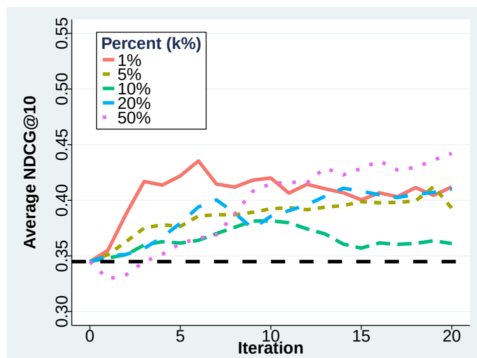
The sensitivity of the parameter settings for different transfer algorithms will be discussed in this section. The RankPairwiseEM algorithms do not require any other parameter setting except for the σ parameter of the sigmoid function, which is usually set as 1 for the LambdaMART algorithm. The RankHardLabelEM 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, which could be set as a constant as 95%. Alternatively, the percentage can be set manually as it is for the RankHardLabelEM, both the manually setting and confidence score based methods will be compared in the following section.

8.1. Threshold setting for RankHardLabelEM

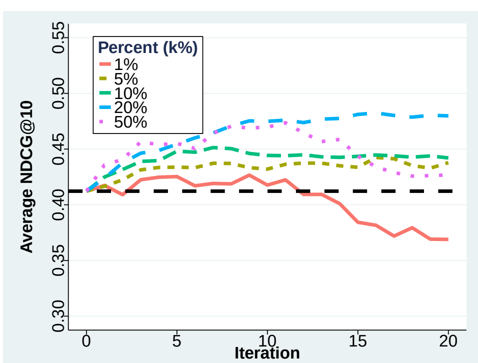
In the RankHardLabelEM algorithm, the percentage of documents being labeled as a relevant document is manually defined. In this section, we compare the performance of the RankHardLabelEM algorithm with different parameter settings. As the source collection we randomly sample 1,000 queries from the MSLR dataset, and as the target collection we sample 1,000 queries from the LETOR4.0 dataset. 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.



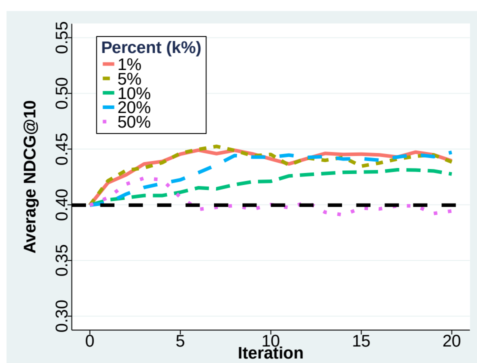
(a) Sampled scenario 1



(b) Sampled scenario 2



(c) Sampled scenario 3



(d) Sampled scenario 4

Figure 5: Comparing the parameter settings for RankHardLabelEM.

The x-axis in Figure 5 is the number of the iterations, the y-axis is the NDCG@10 scores measured on the unlabeled target set, and 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 different at different runs with a different setting of $k\%$. For example, when $k\%$ was set as 1%, its performance increased gradually over the iterations at the first run (Figure 5a), while in the other cases, the performance kept dropping (Figure 5c), indicating a significant amount of variance in performance. Moreover, in some cases, we have seen that the performance of the algorithm will start to decrease after a certain point (50% in Figure 5c), so it is also important to determine when to stop the iterations. Notice that, although 20% seems to be optimal for this particular transfer setting, it may not be the best threshold for other transfer settings.

In RankSelfTrain, there are two parameters, which are the σ for the pairwise preference probability in Equation 7 and the confidence threshold η. The setting of σ is usually determined by the implementation of the LambdaMART algorithm, and is usually set as 1. The confidence threshold η is set as 95%, following the probability convention.

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.

8.2. Confidence Versus Fixed-Increments for RankSelfTrain

In the RankSelfTrain algorithm, we have determined to set a threshold for confidence 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 4. The main difference between the fixed-increments-based RankSelfTrain and confidence-based RankSelfTrain is that the number of relevant labels is fixed, and all the unlabeled documents will also be labeled as irrelevant.

The main challenge with this algorithm is how to set a proper parameter of $\Delta k\%$ for a particular transfer setting. To compare the algorithms, we used the same sampling and testing strategy utilized in the last section. The learning curves of different runs are plotted in Figure 6.

A glance at the figure above illustrates the effectiveness of RankSelfTrain with different parameter settings. Most of the algorithms tested so far have

Input: Source queries Q^{so} and judgements R^{so}, target queries Q^{ta}, maximum number of iterations

Output: Ranking function f

SelfTrain(Q^{so},R^{so},Q^{ta}, Γ)
    Initialize set of *relevant docs* to be empty: $\Omega^{(0)} = \emptyset$;
    Initialize set of *irrelevant docs* to be empty: $\mho^{(0)} = \emptyset$;
    Train ranker $f^{(0)}$ using (Q^{so}, R^{so}) with Eq. 17;
    for $t \in \{1, ..., \Gamma\}$ **do**
        `/* E-step */`
        Calculate scores for all query-doc pairs;
        Sort *unlabeled* pairs $(i, j) \notin \Omega^{(t-1)}$ by score;
        Label top $\Delta k\%$ pairs as *newly* relevant: $\Omega^{(t)} = \Omega^{(t-1)} \cup \{topk\}$;
        Set remaining query-doc pairs as irrelevant: $\mho^{(t)} = X^{ta} - \Omega^{(t)}$;
        `/* M-step */`
        Train ranker $f^{(t)}$ using Eq. 19;
    end
    Return $f^{(t)}$;

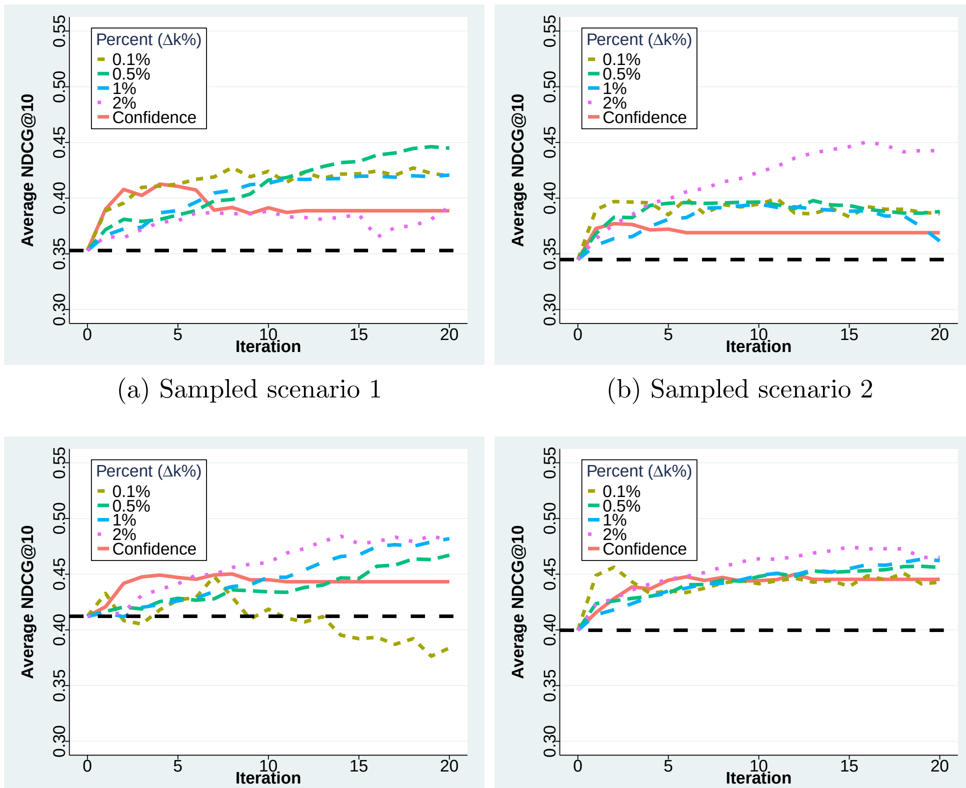
Algorithm 4: RankSelfTrain with top Δ percentage

shown a gradual increase in the effectiveness of the ranker with each iteration, starting from the source ranker (0^{th} iteration).

The performance of the algorithm is different with different parameter settings across different 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 performs 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 plot the first 20 iterations of the process in Figure 6, the five lines cross over at many iterations during the training. This suggests that, 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 provided a parameter-free



(a) Sampled scenario 1

(b) Sampled scenario 2

(c) Sampled scenario 3

(d) Sampled scenario 4

Figure 6: Comparing the parameter settings for RankSelfTrain.

setting except for the confidence threshold. It is arguable that the threshold can always be set as a high value constant so that it is "parameter-free". 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.

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, improvements of the algorithms may not be consistent under different transferring settings (i.e. the dataset). By "consistent", we mean the improvements of ranking effectiveness on the target collection with an unsupervised TR across all circumstances, namely, the transferred model performs no worse than the source model in various transfer 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, reliability and effectiveness could likely be improved if some supervision is provided.

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 transfer ranking (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.

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. In some cases, however, the improvements are minimal. Self-labeling methods are showed to be more effective than instance-weighting algorithms.

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.

For RankHardLabelEM and RankSelfTrain, we have illustrated that both algorithms can achieve better results with 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.

Further research is needed to understand how to use common or latent features to better exploit the labeling process. Apart from the proposed self-labelling approaches, there are other related algorithms, such as multi-viewing learning, that could be explored for unsupervised TR problems. Moreover, the similarity of the source and target collection has been shown to be correlated with transfer effectiveness. Thus, investigations are needed to identify the impact of collection similarity on the performance of unsupervised TR algorithms. In particular, finding the best method for measuring the similarities between different L2R collections could help to avoid some negative transfer effects.

References

References

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

[2] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, B. Dachev, and E. Kanoulas. Million query track 2007 overview. Technical report, University of Massachusetts, Amherst, MA, 2007.

[3] J. Allan, J. A. Aslam, B. Carterette, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. Technical report, Massachusetts University, Amherst, MA, 2008.

[4] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. In *ACM SigIR Forum*, volume 42, pages 9–15. ACM, 2008.

[5] Y. Anzai. Kernel densityestimators. In *Pattern recognition and machine learning*, pages 122–123. Elsevier, 2012.

[6] R. Baeza-Yates, B. Ribeiro-Neto, and others. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[7] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[8] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23):81, 2010.

[9] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, pages 193–200, 2007.

[10] P. Cai, W. Gao, K.-F. Wong, and A. 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.

[11] P. Cai, W. Gao, A. Zhou, and K.-F. 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*, HLT '11, pages 112–122. Association for Computational Linguistics, 2011.

[12] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. 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. ACM, 2006.

[13] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136. ACM, 2007.

[14] O. Chapelle and Y. 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.

[15] M. Chen, K. Q. Weinberger, and J. Blitzer. Co-training for domain adaptation. In *Advances in neural information processing systems*, pages 2456–2464, 2011.

[16] D. Cossock and T. Zhang. Subset ranking using regression. In *International Conference on Computational Learning Theory*, pages 605–619. Springer, 2006.

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[18] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30, 2006.

[19] P. Donmez and J. G. Carbonell. Active sampling for rank learning via optimizing the area under the ROC curve. In *European Conference on Information Retrieval*, pages 78–89. Springer, 2009.

[20] K. Duh and K. 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. ACM, 2008.

[21] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[22] W. Gao, P. Cai, K.-F. Wong, and A. 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*, SIGIR '10, pages 162–169. ACM, 2010.

[23] P. Goswami, M. R. Amini, and E. 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*, CIKM '13, pages 2315–2320. ACM, 2013.

[24] L. A. Granka, T. Joachims, and G. 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*, SIGIR '04, pages 478–479, New York, NY, USA, 2004. ACM.

[25] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142. ACM, 2002.

[26] T. Jones, A. Turpin, S. Mizzaro, F. Scholer, and M. 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*, CIKM '14, pages 1843–1846, New York, NY, USA, 2014. ACM.

[27] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4): 422–446, 2002.

[28] T. Kanamori, S. Hido, and M. Sugiyama. A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10:1391–1445, 2009.

[29] E. Kanoulas and J. 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.

[30] J. Kekäläinen and K. Järvelin. Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002-11.

[31] A. Kumar and M. Lease. Learning to rank from a noisy crowd. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 1221–1222. ACM, 2011.

[32] P. Li, M. Sanderson, M. Carman, and F. 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*, CIKM '16, pages 1413–1422. ACM, 2016.

[33] B. Long, O. Chapelle, Y. Zhang, Y. Chang, Z. Zheng, and B. Tseng. Active learning for ranking through expected loss optimization. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 267–274. ACM, 2010.

[34] M. Long, J. Wang, G. Ding, S. J. Pan, and S. Y. Philip. Adaptation regularization: A general framework for transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1076–1089, 2014.

[35] A. Margolis. A literature review of domain adaptation with unlabeled data. *Tec. Report*, pages 1–42, 2011.

[36] D. McClosky, E. Charniak, and M. 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*, HLT-NAACL '06, pages 152–159. Association for Computational Linguistics, 2006.

[37] D. McClosky, E. Charniak, and M. 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. Association for Computational Linguistics, 2006.

[38] R. Mehrotra and E. Yilmaz. Representative & Informative Query Selection for Learning to Rank Using Submodular Functions. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 545–554, New York, NY, USA, 2015. ACM.

[39] K. Nigam, A. McCallum, and T. Mitchell. Semi-supervised text classification using EM. *Semi-Supervised Learning*, pages 33–56, 2006.

[40] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[41] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[42] S. Ren, Y. Hou, P. Zhang, and X. Liang. Importance Weighted AdaRank. In *Proceedings of the 7th International Conference on Advanced Intelligent Computing*, pages 448–455, Zhengzhou, China, 2011. Springer.

[43] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, and others. Okapi at TREC-3. *Nist Special Publication Sp*, 109: 109, 1995.

[44] M. Sanderson, A. Turpin, Y. Zhang, and F. Scholer. Differences in effectiveness across sub-collections. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1965–1969. ACM, 2012.

[45] V. I. Spitkovsky, H. Alshawi, D. Jurafsky, and C. D. Manning. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 9–17. Association for Computational Linguistics, 2010.

[46] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Buenau, and M. 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.

[47] S. Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7):2031–2038, 2013.

[48] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51(6):757–772, 2015-11.

[49] I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2015.

[50] A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, pages 58–65. ACM, 2014.