On the Effectiveness of Query Weighting for Adapting Rank Learners to New Unlabelled Collections

Pengfei Li
RMIT University, Australia
li.pengfei@rmit.edu.au

Mark Sanderson
RMIT University, Australia
mark.sanderson@rmit.edu.au

Mark Carman
Monash University, Australia
mark.carman@monash.edu

Falk Scholer
RMIT University, Australia
falk.scholer@rmit.edu.au

ABSTRACT

Query-level instance weighting is a technique for unsupervised transfer ranking, which aims to train a ranker on a source collection so that it also performs effectively on a target collection, even if no judgement information exists for the latter. Past work has shown that this approach can be used to significantly improve effectiveness; in this work, the approach is re-examined on a wide set of publicly available L2R test collections with more advanced learning to rank algorithms. Different query-level weighting strategies are examined against two transfer ranking frameworks: AdaRank and a new weighted LambdaMART algorithm. Our experimental results show that the effectiveness of different weighting strategies, including those shown in past work, vary under different transferring environments. In particular, (i) Kullback-Leibler based density-ratio estimation tends to outperform a classification-based approach and (ii) aggregating document-level weights into query-level weights is likely superior to direct estimation using a query-level representation. The Nemenyi statistical test, applied across multiple datasets, indicates that most weighting transfer learning methods do not significantly outperform baselines, although there is potential for the further development of such techniques.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Information retrieval, Learning to rank, Ranking adaptation

1. INTRODUCTION

Learning To Rank (L2R) algorithms are trained to be effective on a particular collection. However, the ranker may not always generalize to other contexts. For example, a ranking function trained for Twitter might not be optimal for searching a newspaper collection. Indeed, even when searching smaller portions of an existing collection, there is evidence that some customization may be needed [19].

A bottleneck for L2R is that training and test data for new collections can be hard to obtain. Relevance judgements (*labels*) require some form of human input, which is often expensive and time-consuming to gather. When learning a ranker for a new document collection with its own set of queries, co-opting judgements from existing collections is a possible solution. Here, transfer learning can be exploited, as it is a technique that trains models for a new (*target*) collection using data from existing (*source*) collections.

In recent years, researchers have tried to apply transfer learning to L2R. The process has been called cross-collection ranking [30], ranking adaptation [4], and *Transfer Ranking* (TR) [23]. The latter is the name we adopt throughout the paper. TR can be divided into supervised and unsupervised approaches, depending on the availability of labels in the target collection. Supervised TR occurs if there are small amounts of labelled data (i.e. relevance judgements for a small set of queries) in the target collection. This type of TR is common [6, 7, 8, 10].

Unsupervised TR assumes that there are no relevance judgements available in the target collection. However, this does not exclude other knowledge, such as details of queries that have previously been submitted to search the target collection [12, 13, 30]. Our work fits into the unsupervised TR scenario.

Among the solutions to unsupervised TR, *instance weighting* is a common technique, which assigns weights to training examples in the source collection to adjust source data distributions to match those in the target collection. Instance weighting can be regarded as the first step for TR, and thus is independent of learning algorithms. The difficulty of applying instance weighting to ranking problems is the complexity of the training data. For conventional classification or ranking problems, the training data consists of sets of feature vectors, which form a multivariate distribution, and instance weighting can be applied to the distribution. However, L2R algorithms train and test models at a query-level, which is a list of feature vectors of documents. Therefore, instance weighting at the query-level is considered to be an appropriate approach. Ways to obtain query-level weights have been investigated, by aggregating document-level weights or directly estimating query weights with query

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983852

representation methods. Some attempts [13, 26] have also been made to apply these techniques to ranking problems, and showed improvements in some collections. However, a deeper understanding of the problem is still required, such as experimental analysis in a wider set of TR environments: test collections and L2R methods.

Previous research has shown that effectiveness of transfer learning varies in different environments [27], and it depends on the similarity of the source and target collection. TR appears to have the similar phenomena [9]. L2R datasets can be different form each other in many ways, and thus it is more difficult to evaluate the effectiveness of TR algorithms. To solve all these problems, one need to have a better understanding of the problem.

In this paper, we answer the following research questions.

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

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

With respect to previous work on unsupervised TR techniques, we explore a much wider range of transferring environments. We make use of two datasets from LETOR4.0, two datasets from the Yahoo! Learning to Rank Challenge, and also set up a transfer environment between the MSLR-Web10K dataset and the LETOR 4.0 dataset. More advanced L2R algorithms are studied in this paper, and we also introduce a visualization method to compare the effectiveness of different models across different datasets.

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

2. TRANSFER RANKING

Transfer learning is a technique to improve the learning of a target task, given source domain data and a task. The effectiveness of a source trained model cannot be guaranteed when tested on a target dataset, which may have a different data distribution. Training data for a learning system is typically made up of two parts, the feature (input) space X, and the label (output) space Y. Thus, changes of data distribution may happen in different ways. Covariate shift happens when the joint distribution of features change ($P^s(X) \neq P^t(X)$). It is also possible that for a given input, the predictions are different in two datasets, in which case the conditional distributions $P(Y|X)$ are different in two different datasets, which means the prediction function has changed over the two different datasets. This type of data distribution is referred to as domain shift [28] or functional change [10].

Different techniques have been tried to tackle these two problems separately or at the same time. For example, different instance weighting strategies [17, 28] have been developed to reduce the distribution divergence in the feature space. Feature mapping methods [1, 16] were developed to tackle the covariate shift problem. Here, even if the distribution of the original feature space is different, other subspaces or latent feature spaces could be found, in which the distributions are similar. Functional change is usually solved by sample selection techniques or through regularization [24].

Since L2R collections are represented by document–query vectors of features, when attempting TR from one collection to another, one can examine changes in the make up of queries, as well as variations in the documents composing the collections. Changes in documents will affect the distribution of features across the collection, as well as the actual features used. Changes in query sets happen in two ways: distribution and type. For distribution, some queries may appear more often in one dataset than another. For query type, e.g. ad-hoc and navigational queries, some types of query might appear more often. In effect, the definition of relevance has changed across the collections, which impacts on the ranker. Duh and Fujino [10] only considered the distribution of queries across collections.

3. RELATED WORK

We first introduce notation for L2R and TR. Training data for L2R consists of ranked lists of documents that were retrieved for a query set. Following Duh and Fujino [10], let:

1. l_i be a ranked list of documents returned by query q_i.
2. $(\vec{x}_i, y_i)$ be a document tuple; each l_j consists of multiple document tuples $(\vec{x}_{j,i}, y_{j,i})_{i=1}^{K}$, where $\vec{x}_i$ represents the vector space of features for document i in the list, while y_i is a relevance label expressed as a score.
3. t and s denote the target and source collections, respectively.

Training data is used differently depending on the type of L2R: point-wise algorithms use single documents; pair-wise

algorithms use the relative ranking preference of pairs of documents; and list-wise algorithms use a ranked list for training. List-wise algorithms typically outperform the other approaches because they train a model directly to optimize the evaluation metrics measured on the ranked list [29].

Most attempts at TR were supervised TR, transferring unsupervised environments is less well-studied. In Gao et al. [13], the authors generated instance weights at different levels for L2R datasets. Although not pointed out, their methods are similar to classification-based density-ratio estimation; they built a classifier hyperplane between source and target documents and used a sigmoid function of the distance of a target document to the hyperplane at the document-level. Since documents are independent of each other, the document-pair weights are the multiplication of the documents' weights. The query weights were generated by the average weights of document-pairs in the query. They tested their instance weights with RankSVM and RankNet (two pair-wise L2R algorithms[1]) on the six topic sets in LETOR3.0, and showed some significant improvements. Cai et al. [5] further improved the algorithm by classifying the queries directly. In our experiments, the algorithm was also implemented, but instead of using the probability transferred from the distance, we employed a logistic regression classifier which can output the probability directly.

An importance weighted AdaRank approach (wAda) was proposed by Ren et al. [26]. The authors used the Kullback-Leibler Importance Estimation Procedure (KLIEP) to estimate document weights, which were then incorporated into the AdaRank algorithm. However, the algorithm was not tested under an unsupervised TR scenario. 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.

There are other approaches to unsupervised TR. For example, instead of learning with data from the source collection, Goswami et al. [15] tried to predict relative relevance judgement for document pairs in the target collection and then use the judgements to train a ranking function for the target collection.

4. INSTANCE WEIGHTING

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

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

[1]AdaRank and LambdaMART are more effective [29].

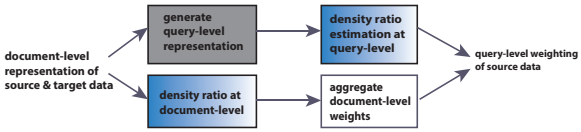


Figure 1: Pipelines for query-level instance weighting

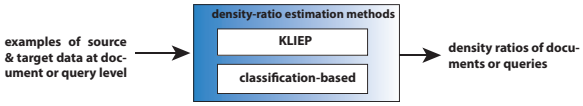


Figure 2: density-ratio estimation methods

Many techniques were developed to efficiently estimate the density-ratio $w(\vec{x})$ at the source datapoints. We examine two popular techniques, namely KLIEP[28] and a classification-based approach. In comparison with the density ratio estimation method in Huang et al. [17], KLIEP is considered state-of-the-art. The KLIEP technique aims to learn a function that minimises the divergence between the weighted source collection and the target collection:

$$\hat{w}(\vec{x}) = \underset{w(\vec{x})}{argmin}\ KL(w(\vec{x})p^s(\vec{x})||p^t(\vec{x})) \quad (2)$$

As suggested by Kanamori [20], the density ratio can also be estimated using a classification-based approach. The technique involves combining a sample of data points from the source and target domains, and then training a probabilistic classifier to classify the domain of each instance. Having trained such a model, the density ratio can be estimated as:

$$\hat{w}(\vec{x}) = \frac{N^s}{N^t} \frac{p(y = target|\vec{x})}{1 - p(y = target|\vec{x})} \quad (3)$$

where N^s and N^t are the number of instances in the source and target collection respectively, and $P(y = target|\vec{x})$ is the classifier's estimate of the class probability. For example, a logistic regression classifier (LR) can be employed to generate a probabilistic model, $p(y|\vec{x}, \theta^*)$.

4.1 Instance Weighting for TR

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

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

Estimation of the density-ratio of the documents is straightforward, while it is more difficult to estimate the density-ratio for queries, since queries are lists of documents instead of single data points. Representing queries in a feature vector space is possible, but there is a danger that the repre-

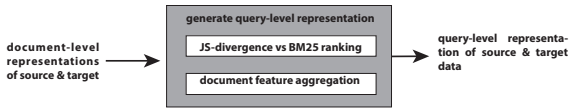


Figure 3: Query representation methods

sentation could lose the structural information in queries. Similar ideas have been applied for query-dependent learning algorithms, where the difference of queries has been considered for ranking [14, 25]. We explore two approaches to representing queries, as shown in Figure 3. The two methods have been successfully used by query-clustering-based L2R algorithms.

Document feature aggregation.

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

$$\vec{q}_j =< \frac{1}{n}\sum_i x_{j,i,1}, \cdots, \frac{1}{n}\sum_i x_{j,i,m} > \quad (4)$$

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

Feature Divergence.

A simple term representation of queries is unlikely to be effective as term overlap is likely to be low. Equally, comparing document-query features alone in each of the collections is unlikely to be effective, as the value of document-query features is likely to be different between the two collections. We therefore apply a weighting method, inspired by an approach proposed by Peng et al. [25], who used a baseline ranker to act as a normalizing pivot against which document-query features in the collections were measured and compared.

Peng et al. [25] suggested that the effect of a feature on a query can be represented by its divergence from a baseline ranker, such as BM25. The divergence represents how much a document ranking has been changed by a particular feature. This provides a way to normalize measurement of features across collections. Both Kullback-Leibler [21] and Jensen-Shannon (JS) Divergence [22] were used.[2] Since it is symmetric, we use the JS Divergence to calculate a query-level feature vector $\vec{JS}(q_j) = \langle JS_k(q_j), \ldots JS_n(q_j)\rangle$ as follows:

$$JS_k(q_j) = JS(\vec{x}_{j,\cdot,1}||\vec{x}_{j,\cdot,b}) = \frac{1}{2}(KL(\vec{x}_{j,\cdot,k}||\overline{x}_j)+KL(\vec{x}_{j,\cdot,b}||\overline{x}_j)) \quad (5)$$

where $\vec{x}_{j,\cdot,b}$ is the vector of values for the baseline feature (assumed to be BM25) for query j, and $\vec{x}_{j,\cdot,k}$ denotes the vector of values for the k^{th} feature. $KL(.)$ denotes the Kullback-Leibler Divergence [21] between two distributions, calculated as:

$$KL(\vec{x}_{j,\cdot,k}||\vec{x}_{j,\cdot,k'}) = \sum_{i=1}^{N} x_{j,i,k}\log_2\frac{x_{j,i,k}}{x_{j,i,k'}} \quad (6)$$

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

Here, $x_{j,i,k}$ is the document score of d_i assigned by the k^{th} feature and N is the number of documents in a ranked list. This query representation is the same as that used by Peng et al., which to the best of our knowledge has never been applied to TR before.

5. UNSUPERVISED TR FRAMEWORKS

We describe two TR frameworks that can incorporate the query weights into training: an existing framework modified from AdaRank for importance weighting [26]; and a new weighted LambdaMART approach that we developed.

5.1 Weighted AdaRank (wAda)

We integrate query weights into AdaRank, a list-wise L2R model [31].[3] AdaRank learns an ensemble model F, which is a linear combination of weak rankers:

$$F(\vec{x};\beta,\alpha) = \sum_{i=1}^{t}\beta_i h_i(\vec{x};\alpha_i) \quad (7)$$

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

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

Thus adding density-ratio weights to AdaRank is straightforward and involves modifying the initial weighting of queries in the AdaRank algorithm. Following query importance weighted AdaRank (wAda) [26], density-ratio weights are assigned to queries at the initial stage. When updating query weighting at the end of iteration, density-ratio weights will also be applied to queries together with their performance weights.

5.2 Weighted LambdaMART (wλMART)

LambdaMART [3] is an L2R algorithm which uses Gradient Boosted Regression Trees to optimize a listwise objective function, which depends on the chosen evaluation metric. Thus, similar to AdaRank, LambdaMART also relies a boosting-based technique that outputs an ensemble of weak learners as in Equation (7), except that the weak learners $h(\vec{x};\alpha_i)$ are regression trees rather than single feature based predictors as was the case for AdaRank. The reason for introducing regression trees is that they have proven effective for training ranking models. At each iteration, LambdaMART fits a new regression tree to the gradient of the objective function of the current ensemble.

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

For a regression tree, the prediction at each leaf or the tree is simply the average value of the training examples assigned to the leaf. (The average value is chosen because it

[3]Our weighting strategy can be used for any list-wise L2R algorithm.

minimises the squared error of the prediction at the node). Trees are grown by recursively splitting the data present at each leaf node. For each leaf, the feature k and split-point s is chosen that results in the minimum sum of squared errors across the resulting branches:

$$(k,s)^* = \underset{k,s}{argmin} \sum_{i:x_{ik}\leq s} (\lambda_i - \bar{\lambda}_L)^2 + \sum_{i:x_{ik}>s} (\lambda_i - \bar{\lambda}_R)^2 \quad (8)$$

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

$$(k,s)^* = \underset{k,s}{argmin} \sum_{i:x_{ik}\leq s} w_i(\lambda_i - \bar{\lambda}_L)^2 + \sum_{i:x_{ik}>s} w_i(\lambda_i - \bar{\lambda}_R)^2 \quad (9)$$

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

$$(k,s)^* = \underset{k,s}{argmin} \sum_i w_i\lambda_i^2 - (V_L\bar{\lambda}_L^2 + V_R\bar{\lambda}_R^2) \quad (10)$$

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

$$(k,s)^* = \underset{k,s}{argmax} \frac{(S_L)^2}{V_L} + \frac{(S_L)^2}{V_R} \quad (11)$$

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

Input: $\{\vec{x}_i, \lambda_i, w_i\}_{i=1}^N$, document feature vectors, corresponding λ-gradient values, and instance weights for instances

$leafCount = 1$;

WeightedTree$(\{\vec{x}_i, \lambda_i, w_i\}_{i=1}^N)$

  if $leafCount \geq 10$ **then**

    return leaf with prediction: $\bar{\lambda} = \frac{\sum_i w_i\lambda_i}{\sum_i w_i}$;

  else

    `/* find best feature and split point */`

    $bestgain = 0$;

    $k = -1; s = -1$;

    for $feature \in \{1, \ldots, m\}$ **do**

      $sort(\{\vec{x}_i\}_{i=1}^N, feature)$

      $S_L = 0;\ S_R = \sum_i w_i\lambda_i$;

      $V_L = 0;\ V_R = \sum_i w_i$;

      $previous = -\infty$;

      for $split \in possible\_splits(X, feature)$ **do**

        $\Delta S = \sum_{i:previous<x_{i,feature}\leq split} w_i\lambda_i$;

        $\Delta V = \sum_{i:previous<x_{i,feature}\leq split} w_i$;

        $S_L = S_L + \Delta S;\ S_R = S_R - \Delta S$;

        $V_L = V_L + \Delta V;\ V_R = V_R - \Delta V$;

        $gain = (S_L)^2/V_L + (S_R)^2/V_R$;

        if $gain > bestgain$ **then**

          $bestgain = gain$;

          $k = feature;\ s = split$;

        end

        $previous = split$;

      end

    end

    if $bestgain > 0$ **then**

      `/* recurse to child nodes */`

      $leafCount = leafCount + 1$;

      $left$ =WeightedTree$(\{\vec{x}_i, \lambda_i, w_i\}_{i:x_ik\leq s})$;

      $right$ =WeightedTree$(\{\vec{x}_i, \lambda_i, w_i\}_{i:x_ik>s})$;

    else

      return leaf with prediction: $\bar{\lambda} = \frac{\sum_i w_i\lambda_i}{\sum_i w_i}$;

    end

  end

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

6. DATA AND METHODS

The set up for the testing is now described.

6.1 Collections

To validate the TR techniques, three existing widely used L2R collections were used: the LETOR 4.0 dataset, the Microsoft Learning to Rank datasets (MSLR[5]), and the Yahoo! Learning to Rank (Yahoo!L2R) datasets, see Table 1. Each collection was set up to contain a pair of datasets to simulate transfer from source to target. In LETOR 4.0 we use the same document collection, but different query sets so we can examine how different weighting methods perform when the source and target collection are drawn from the same distribution. The Yahoo!L2R collection is composed of two datasets, which were created to test TR. The documents and queries of Set 1 are pooled from a non-English search engine, and the size of query size is smaller than that of Set 2. The two datasets share 415 features in common, and we used the 415 features in our experiment.

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

In each group, we select source and target collections, and then randomly split the target collection into five folds for cross-validation evaluation. In each experiment, four folds were taken as training data, and we removed the labels of the training set to simulate an unsupervised TR experiment. On both test collections, we assigned one of the collections

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

[5]We used the subset of MSLR: MSLR-WEB10K. To keep denotation simple, we refer to the subset as MSLR.

Dataset	Docs	Topics	Queries
LETOR 4.0	GOV2	MQ2007	1,700
		MQ2008	800
MSLR	web documents	-	10k
Yahoo!L2R	Yahoo	-	36k

Table 1: Test collections used in the experiments

as the source and the other as the target The TR algorithms are tested in both "directions", first with an initial assignment and then with the assignments reversed. The transfer settings are shown in Table 2.

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

6.2 Measures

The effectiveness measure and training objective function was Normalized Discounted Cumulative Gain (NDCG) [18] with the standard rank-plus-one discount function and exponential gain [2]. We measure NDCG at rank cutoff 10. We conducted two-tailed t-tests between results with p set at 0.05, also the value set for the later Friedman's test.

6.3 Setup

The implementation of the algorithms used the open source L2R library Ranklib2.1.[6] For all AdaRank based algorithms, we set the iteration number to 500. For all LambdaMART based algorithms, we trained 1,000 trees with ten leaves with jForests-0.5 library [11][7]. The learning rate was set at 0.1. Sugiyama-Sato's KLIEP code[8] was used to estimate density-ratio at the document or query level.

6.4 Comparison Models

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

- **BM25** was used as a baseline to examine whether a TR algorithm can exceed the performance of a static ranker.
- **Ada.source** was the model trained by AdaRank with all the data from the source collection. This is a simple TR algorithm with the model is transferred to the target collection, but with no adjustment.
- **λMART.source** was the model trained by LambdaMART with all the data from the source collection. This is a simple TR algorithm with the model is transferred to the target collection, but with no adjustment.
- **Ada.target** was the model trained by AdaRank with data from the target collection, the results were measured by 5-Fold cross validation. One can view this as an upper bound to which the designer of the TR algorithms aspires to achieve.
- **λMART.target** was the model trained by LambdaMART with data from the target collection, the results were measured by 5-Fold cross validation. One can view this as an upper bound to which the designer of the TR algorithms aspires to achieve.

The following instance weighting algorithms were tested in the experiments:

- **kliep.doc**: density-ratio estimation at document-level using KLIEP, aggregating document weights of queries to query-level weights. kliep.doc with wAda (wAda-kliep.doc) was proposed in Ren et al. [26].
- **kliep.avg**: density-ratio estimation at query-level using KLIEP, with feature aggregating representation method.
- **kliep.js**: density-ratio estimation at query level using KLIEP, with a JS divergence representation.
- **class.doc**: density-ratio estimation at document level using a classification-based method, aggregating document weights of queries to query level weights. This approach was most close to the weighting strategy proposed by Gao et al. [13].
- **class.avg**: density-ratio estimation at query level using a classification based method, with a feature aggregating representation method.
- **class.js**: density-ratio estimation at query level using a classification based method, with a JS divergence representation.

7. RESULTS

The effectiveness of the instance weighting algorithms with wAda and wλMART are now described.

7.1 Comparing AdaRank & LambdaMART

LambdaMART was found to be more effective than AdaRank in all the six datasets (as shown the last lines of Table 3 and 4). However, when we use models trained with all the data from a source collection to a target collection (the source models, i.e. Ada.source and λMART.source), the effectiveness varies. We represent the effectiveness of AdaRank and LambdaMART models on different datasets and to examine how the trained model can be generalized to another collection.

The document collection is in common to MQ2007 and MQ2008. While MQ2007 has more queries, both AdaRank and LambdaMART trained on MQ2007 was more effective than the models trained on MQ2008 when testing on both collections.

The difference between Set1 and Set2 in Yahoo!L2R are bigger than the difference of the two datasets in LETOR 4.0. When transferring from Set 1 to Set 2, both Ada.source and λMART.source saw a 6% performance decrease compared with target models (Ada.target and λMART.target). However, in the opposite direction, *sourceAda* is 1% worse than *targetAda* while LambdaMART suffers a greater performance decrease (6%). However, source models trained with LambdaMART are still better than AdaRank.

In the last group of transferring settings, where the dissimilarity is also the largest, the decrease of model performance is even greater. Compared with Ada.target, Ada.source saw

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

	Yahoo			LETOR 4.0			LETOR 4.0-MSLR		
	Collection	Queries	Feature	Collection	Queries	Feature	Collection	Queries	Feature
Source	Set 1	19,944	415	MQ2007	1,700	46	LETOR 4.0	2,340	45
Target training	Set 2	5064	415	MQ2008	640	46	MSLR-WEB10K	8k	45
Target testing	Set 2	1266	415	MQ2008	160	46	MSLR-WEB10K	2k	45
Source	Set 2	6330	415	MQ2008	800	46	MSLR-WEB10K	10k	45
Target training	Set 1	15955	415	MQ2007	1440	46	LETOR 4.0	1,872	45
Target testing	Set 1	3989	415	MQ2007	360	46	LETOR 4.0	468	45

Table 2: Transfer Settings for testing different algorithms

	MQ2007-MQ2008	MQ2008-MQ2007	Yahoo.set1-Yahoo.set2	Yahoo.set2-Yahoo.set1	MSLR-LETOR 4.0	LETOR 4.0-MSLR
BM25	0.335	0.249	0.540	0.507	0.276	0.180
Ada.source	0.495	0.353	0.658	0.701	0.367	0.251
wAda.kliep.doc	0.329 ↓	**0.431** ↑	**0.708** ↑	0.704 ↑	0.286 ↓	0.196 ↓
wAda.kliep.avg	0.379 ↓	0.383 ↑	0.684 ↑	0.695 ↓	0.370 ↑	0.281 ↑
wAda.kliep.js	0.493	0.384 ↑	0.694 ↑	0.705 ↑	0.402 ↑	0.274 ↑
wAda.class.doc	0.497	0.424 ↑	0.690 ↑	0.688 ↓	0.362 ↓	0.303 ↑
wAda.class.avg	**0.501**	0.265 ↓	0.566 ↓	0.605 ↓	0.362 ↓	0.140 ↓
wAda.class.js	0.363 ↓	0.383 ↑	0.561	0.667	0.370 ↑	0.281 ↑
Ada.target	0.494	0.417 ↑	0.698	**0.710** ↑	**0.447** ↑	**0.304** ↑

Table 3: Effectiveness (NDCG@10 score) on different transfer settings with wAda. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than Ada.source, ↓ denotes the figure is significantly worse than Ada.source. $p < 0.05$

a 17.9% and 17.4% decrease in system effectiveness when transferring from MSLR to LETOR 4.0 and from LETOR 4.0 to MSLR respectively. LambdaMART suffers from an even greater drop; the λMART.sources are 49% and 54.1% worse than the λMART.targets in the two transferring scenarios. During the training, LambdaMART looks into the feature space to find best splits to minimize the loss function, while AdaRank is just looking to find best features that gain the best performance on the query sets. Thus, LambdaMART is more sensitive to the distribution of the feature space. In the last group of collections, the feature spaces of LETOR 4.0 and MSLR are too different between each other. Moreover, the features of LETOR 4.0 have been normalized while MSLR datasets kept the original features.

7.2 Transferring with Different Algorithms

wAda and wλMART perform differently even with the same weighting strategies, and on the same datasets. As shown in the third columns of Tables 3 and 4, when transferring from Set 1 to Set 2 of Yahoo!L2R dataset,the kliep.doc, kliep.avg, kliep.js and class.doc methods are significantly better the Ada.source when they are used in the wAda framework. However, with the same set of weighting algorithms, wλMART decreases the effectiveness of λMART.source. Similarly, all the wλMART algorithms outperform the LambdaMART without weights (λMART.source), when transferring from MSLR to LETOR 4.0, while some of the wAda algorithms reduce the performance under the same environment, as shown in the last columns of the two tables.

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

7.3 Transferring Across Different Datasets

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

LETOR 4.0.

As we mentioned, MQ2007 and MQ2008 are two samples from the same distribution, while MQ2007 has a larger query set size than MQ2008. Transferring from MQ2007 to MQ2008 appears not to work at all with all weighting methods under either TR framework. However, when transferring is executed in the opposite direction, some improvements occur with different weighting approaches. Most of the figures from third column of Table 3 are significantly better than Ada.source, except the wAda.class.avg. We speculate that since MQ2007 has a larger dataset, this tends to be more unbiased than MQ2008, and since MQ2007 and MQ2008 are two samples from the same distribution, estimating the density-ratio with respect to an unbiased sample would improve the accuracy. Moreover, since the size of MQ2008 is small, the testing set would have a small portion for testing, which could cause the variation in test results.

Yahoo!L2R.

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

	MQ2007-MQ2008	MQ2008-MQ2007	Yahoo.set1-Yahoo.set2	Yahoo.set2-Yahoo.set1	MSLR-LETOR 4.0	LETOR 4.0-MSLR
BM25	0.335	0.249	0.540	0.507	0.276	0.180
λMART.source	**0.505**	0.407	0.718	0.702	*0.236*	0.197
wλMART.kliep.doc	0.499 ↓	0.412 ↑	0.712 ↓	0.703	*0.273* ↑	0.200 ↑
wλMART.kliep.avg	0.498 ↓	0.384 ↓	0.705 ↓	0.697	0.271 ↑	0.180 ↓
wλMART.kliep.js	0.473 ↓	0.395	0.710 ↓	0.700 ↓	0.295 ↑	0.222 ↑
wλMART.class.doc	0.496 ↓	0.413 ↑	0.710 ↓	0.697 ↓	0.289 ↑	0.202 ↑
wλMART.class.avg	0.495 ↓	0.408	0.693 ↓	0.690 ↓	0.289 ↑	0.213 ↑
wλMART.class.js	0.466 ↓	0.392 ↓	0.698 ↓	0.686 ↓	*0.273* ↑	0.226 ↑
λMART.target	0.501	**0.455** ↑	**0.763** ↑	**0.742** ↑	**0.463** ↑	**0.429** ↑

Table 4: Effectiveness (NDCG@10 score) on different transfer settings with wλMART. Bold text indicates the best scores of each column, ↑ denotes the figure is significantly better than λMART.source, ↓ denotes the figure is significantly worse than λMART.source. $p < 0.05$

MSLR-LETOR 4.0.

Transferring between MSLR and LETOR 4.0 is the most challenging task among all the transferring settings, since datasets are dissimilar. Although the source models suffer from a substantial effectiveness drop compared with target models, many of the examined weighting algorithms appear to work. Consistent improvements are observed from kliep.js and class.js with both frameworks. When transferring from MSLR to LETOR 4.0, the wAda.kliep.js algorithm in Table 3 gains 9.5% effectiveness compared with the Ada.source. When the same weighting approach was used in wλMART, a 25% performance boost is achieved compared with λMART.source under the same transferring setting. Transferring in the other direction, from LETOR 4.0 to MSLR, also shows improvements, from the weighting algorithms. For example, wAda.class.doc outperforms Ada.source by 5.2%, while wλMART.class.js increased effectiveness by 25% from λMART.source.

7.4 Does Query-Level Instance Weighting Work?

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

It is very difficult to distinguish the better algorithms from the poorer ones based on inspection of the results tables. Thus we visualise the results in Figure 4 by computing the average rank [9] for each approach across all datasets (and all folds). The Nemenyi test [10] is used to determine whether there is significant difference between the average rank of any two systems.

The differences between models is compared against the critical distance (CD)[11], two models are not significantly different if the average ranks $\overline{r}$ of two models are within the CD. We examine CD on the average rank graph to determine whether one model is more effective than another. The results of the tests are displayed in Figure 4, where the y-axis shows the average ranks of the models. The black dots show the average ranks of particular models, with line connecting a black and an empty dot the CD. If a model's mean rank has no overlap with another model's line, then they are significantly different.

The tests on AdaRank (Figure 4a) show that some instance weighting methods are more effective than the non-weighted AdaRank. However, the differences are not significant. The document-level classification based algorithm (*class.doc*) and the KLIEP method with JS query representation method (*kliep.js*) show some improvements over Ada.source (AdaRank-source in the figure). KLIEP based weighting methods are better than classification based algorithms, as most KLIEP based algorithms are ranked above or around the Ada.source models. Two algorithms, *class.js* and *class.avg*, are most likely to be useless for wAda since they are significantly worse than the non-weighted model.

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

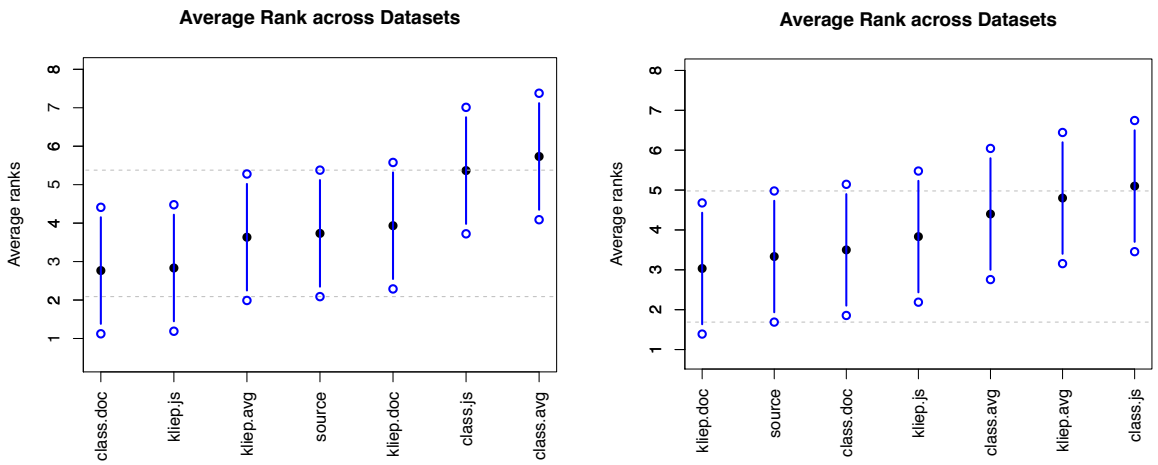
Query representation methods are an attempt to represent queries at a high level. If the method cannot represent the properties of queries, for example, averaging document features just ignore the ranking preferences of documents in the query, the density of queries can be estimated incorrectly. Instead, density-ratio estimation at document-level is looking at the distributions of the feature input space, which is also the direct inputs to an algorithm, the density-ratio density estimation is then meaningful. But how to use the document-level weights for ranking is another issue.

8. CONCLUSIONS

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

[10]The Nemenyi test is used to determine whether there is significant difference between the average rank of any two systems. It can be performed after first checking with the Friedman test (a non-parametric alternative to repeated measures ANOVA) that the systems are not independent of rank (across the datasets).

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



(a) Compare all Adarank-based models

(b) Compare all LambdaMART-based models

Figure 4: Plots of average rank across the 6 test environments for the 6 different Transfer Learning techniques and the "source" baseline system (where no TR was applied), the critical distance (CD) for the Nemenyi test (at the 5% confidence level). The lower the rank the better performance of the approach.

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

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

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

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

As the results show some inconsistency of the effectiveness of algorithms across different datasets, further work could analyse the correlation between the TR settings and the improvements gained by unsupervised TR algorithms. Furthermore, the experiment could be extended to larger datasets and different TR algorithms, as well as studying better document-level weight aggregating strategies for query-level density-ratio estimation.

9. ACKNOWLEDGMENTS

This work was supported in part by the Australian Research Council (DP130104007) a scholarship from RMIT University, and also by a Google Faculty Research Award.

References

[1] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 120–128, 2006. ISBN 1-932432-73-6.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the*

22Nd International Conference on Machine Learning, ICML '05, pages 89–96, 2005.

[3] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[4] P. Cai, W. Gao, K.-F. Wong, and A. Zhou. Weight-based boosting model for cross-domain relevance ranking adaptation. In *Advances in Information Retrieval*, pages 562–567. Springer, 2011.

[5] 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*, pages 112–122. Association for Computational Linguistics, 2011.

[6] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1189–1198. ACM, 2010.

[7] D. Chen, Y. Xiong, J. Yan, G.-R. Xue, G. Wang, and Z. Chen. Knowledge transfer for cross domain learning to rank. *Information Retrieval*, 13(3):236–253, 2010.

[8] K. Chen, R. Lu, C. Wong, G. Sun, L. Heck, and B. Tseng. Trada: tree based ranking function adaptation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1143–1152. ACM, 2008.

[9] K. Chen, J. Bai, S. Reddy, and B. Tseng. On domain similarity and effectiveness of adapting-to-rank. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1601–1604. ACM, 2009.

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

[11] Y. Ganjisaffar, R. Caruana, and C. V. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 85–94. ACM, 2011.

[12] W. Gao and P. Yang. Democracy is good for ranking: Towards multi-view rank learning and adaptation in web search. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 63–72. ACM, 2014.

[13] 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*, pages 162–169. ACM, 2010.

[14] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122. ACM, 2008.

[15] P. Goswami, M. R. Amini, and É. Gaussier. Transferring knowledge with source selection to learn ir functions on unlabeled collections. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2315–2320. ACM, 2013.

[16] H. D. Hal. Frustratingly Easy Domain Adaptation. *Proceedings of the 45th Association of Computational Linguistics*, pages 256–263, 2007.

[17] J. Huang, A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.

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

[19] 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*, pages 1843–1846. ACM, 2014.

[20] T. Kanamori. Density ratio estimation: A comprehensive review. *Workshop on Statistical Experiment and its Related Topics*, 1703:10–31, 2010.

[21] S. Kullback. *Information theory and statistics*. Courier Corporation, 1968.

[22] J. Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.

[23] B. Long, Y. Chang, A. Dong, and J. He. Pairwise cross-domain factor model for heterogeneous transfer ranking. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 113–122. ACM, 2012.

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

[25] J. Peng, C. Macdonald, and I. Ounis. Learning to select a ranking function. In *Advances in Information Retrieval*, pages 114–126. Springer, 2010.

[26] 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. Springer, 2011.

[27] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *NIPS 2005 Workshop on Transfer Learning*, volume 898, 2005.

[28] 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 *Proceedings of Neural Information Processing Systems (NIPS)*, pages 1–8, 2008.

[29] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Inf. Process. Manage.*, 51(6):757–772, Nov. 2015.

[30] B. Wang, J. Tang, W. Fan, S. Chen, C. Tan, and Z. Yang. Query-dependent cross-domain ranking in heterogeneous network. *Knowledge and information systems*, 34(1):109–145, 2013.

[31] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM, 2007.