Ranking Function Adaptation with Boosting Trees

KEKE CHEN, Wright State University
JING BAI, Microsoft
Zhaohui Zheng, Yahoo! Labs

Machine learned ranking functions have shown successes in web search engines. With the increasing demands on developing effective ranking functions for different search domains, we have seen a big bottleneck, i.e., the problem of insufficient labeled training data, which has significantly slowed the development and deployment of machine learned ranking functions for different domains. There are two possible approaches to address this problem: (1) combining labeled training data from similar domains with the small target-domain labeled data for training or (2) using pairwise preference data extracted from user clickthrough log for the target domain for training. In this paper, we propose a new approach called tree based ranking function adaptation ("Trada") to effectively utilize these data sources for training cross-domain ranking functions. Tree adaptation assumes that ranking functions are trained with the Stochastic Gradient Boosting Trees method – a gradient boosting method on regression trees. It takes such a ranking function from one domain and tunes its tree based structure with a small amount of training data from the target domain. The unique features include (1) it can automatically identify the part of model that needs adjustment for the new domain, (2) it can appropriately weigh training examples considering both local and global distributions. Based on a novel pairwise loss function that we developed for pairwise learning, the basic tree adaptation algorithm is also extended ("Pairwise Trada") to utilize the pairwise preference data from the target domain to further improve the effectiveness of adaptation. Experiments are performed on real datasets to show that tree adaptation can provide better-quality ranking functions for a new domain than other methods.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Retrieval Models

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: web search ranking, boosting regression trees, domain adaptation, learning to rank, user feedback

ACM Reference Format:
Chen, K., Bai, J., and Zheng, Z. 2011. Ranking Function Adaptation with Boosting Trees. ACM Trans. Inf. Syst. 10, 1, Article 1 (October 2011), 30 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Learning to rank has been a promising method for continuously and efficiently improving relevance for web search. It applies novel machine learning algorithms [Zheng et al. 2007; Cao et al. 2007; Freund et al. 2003; Tsai et al. 2007; Xu and Li 2007; Joachims 2002; Burges et al. 2005] to a set of labeled relevance examples to learn a ranking function. Compared to the traditional ranking functions [Baeza-Yates and Ribeiro-Neto 1999] developed in the information retrieval community, learning to rank has several unique benefits: (1) it is convenient to incorporate new features to the rank-

Author's addresses: K. Chen, Department of Computer Science and Engineering, Wright State University, keke.chen@wright.edu; J. Bai, Microsoft, jbai@microsoft.com; Z. Zheng, Yahoo! Labs, zhaohui@yahoo-inc.com.

ing function without the need of manually tuning the function, which mainly relies on experts' experience and heuristics; (2) although depending on the specific learning algorithms, with sufficient training data it can usually give better performance over manually tuned functions. Currently, machine learned ranking functions have been successfully applied to several major search engines.

Learning to rank requires a sufficient amount of good-quality labeled training data. To obtain good-quality training data, we usually need trained editors (relevance experts) to judge the relevance of sampled web search results, i.e., (query, document) pairs, and cross-verify the judgments. Since this process has to be done manually, it is highly time-consuming and expensive. Although there are convenient methods to extract relevance judgments from implicit user feedbacks [Joachims 2002; Joachims et al. 2005], the quality is difficult to guarantee. Therefore, the expert-labeled data are still regarded as a more reliable source for training high-quality ranking functions[1]. Due to the increasing demands from different web search domains, e.g., different regions or countries or topics, it has been necessary to develop effective *domain-specific* ranking functions and continuously improve them. However, when applying learning to rank to a new domain, we usually do not have a sufficient amount of *labeled* training data.

One approach to addressing this problem is utilizing the training data from one major web search domain to help train a function for a new search domain. Apparently, the training data from one existing domain cannot be easily applied to another domain, due to different joint feature−relevance distributions. Specifically, for the web search ranking problem, there are usually tens or hundreds of features designed for learning a ranking function. Even small distribution differences in each feature will aggregate to significant differences in the multidimensional feature space. Although each search domain has its own characteristics, we observe that many of them share a certain level of commonality. In particular, we have seen that a ranking function developed in one domain, though not the best function for another domain, works reasonably well across different domains. We name the domain with sufficient training data as *the source domain*, and the domain with only a small amount of training data as *the target domain*. How to adapt a good ranking function from the source domain to a target domain and get a better ranking function is the major problem we are going to tackle.

Another approach is utilizing the pairwise preference data mined from the user clickthrough log [Joachims 2002; Joachims et al. 2005; Agichtein et al. 2006; Chapelle and Zhang 2009; Ji et al. 2009] for the target domain. Assume we temporarily apply an acceptable ranking function that was developed for another domain to the target domain to collect pairwise preference data. An acceptable ranking function here means the top results, e.g., top 10, for most queries contain the most relevant results but the ranking quality is not ideal. The user clickthrough data are collected based on this initial ranking function. As recent research shows, pairwise preference data from clickthrough log might have a strong bias so that using them to train ranking functions might not be sufficient. Adaptation can utilize the benefit of the pairwise data while overcoming the bias brought by them. However, due to the different form of data, how to incorporate this pairwise dataset to the function adaptation process is the second challenging problem.

In this paper, we propose a tree-based ranking function adaptation approach (Trada) to address the problem of insufficient training data for target search domains. Although it can be applied to any regression-tree based ranking model, we will use rank-

[1]The data for modeling Yahoo! web search ranking functions: http://learningtorankchallenge.yahoo.com/; The data for modeling Microsoft Bing ranking functions: http://research.microsoft.com/en-us/projects/mslr/; and the LETOR data: http://research.microsoft.com/en-us/um/beijing/projects/letor/

ing functions trained with the gradient boosting trees (GBT) method [Friedman 2001] in this paper. Tree-based models have some known advantages over other kinds of models, such as the model interpretability. In our approach, we will utilize the unique structural benefit of regression tree. Based on the characteristics of regression tree and the mechanism of training a regression tree, we design a few algorithms to tune the base-model trees with the small target training dataset. The tree-based adaptation algorithm has a couple of unique features: (1) it can automatically identify the part of model that needs to adjust for the new domain, (2) it can appropriately weigh training examples considering both local and global distributions. By doing tree-adaptation, we can effectively tune a base model towards the domain-specific distributions indicated by the small dataset, and thus incorporate the knowledge learned by the base model into the target domain. Experiments have shown that this approach is more effective than other methods.

The basic tree adaptation approach is also extended to utilize pairwise training data - the “Pairwise Trada” algorithm. It is based on the idea of another our contribution, the GBRank method for handling pairwise data based on the regression framework [Zheng et al. 2007; Zheng et al. 2007]. Since the pairwise data might be biased due to the extraction methods, by combining the merits of GBRank and Trada, Pairwise Trada avoids the dataset bias and takes advantages of both sets of datasets. Experimental study shows that both Trada and Pairwise Trada can result in better models than other methods.

In Section 2, we will briefly review the related work, mainly, the representative learning to rank algorithms and the model adaptation work done in other areas. In Section 3, we describe some basic concepts and notations that will be used in tree adaptation. We will also analyze how a regression tree is generated, which helps understand the basic idea of tree adaptation. In Section 4, we first give intuitions on how tree adaptation works and then present a few tree adaptation algorithms. In Section 5, we present the Pairwise Trada algorithm based on the basic Trada algorithm and the GBRank loss function. Experimental results will be reported in Section 6 to validate the effectiveness of the proposed algorithms with different algorithmic settings and several real datasets from a major web search engine.

2. RELATED WORK

In recent years, several novel learning algorithms have been developed for the ranking problem. They can be roughly grouped into three categories. The first category works on training data labeled with absolute grades, typically two-level grades as “relevant” and “irrelevant”, or multilevel grades. Correspondingly, the learning problem is formalized as a classification problem [Nallapati 2004] or ordinal regression problem [Herbrich et al. 2000; Cao et al. 2006; Friedman 2001]. Among this type of approaches, regression tree based approaches [Li et al. 2007; Cossock and Zhang 2006; Zheng et al. 2007; Wu et al. 2010; Burges 2010], especially, those based on gradient boosting trees [Friedman 2001], have shown benefits in selecting useful features from many features and generating high quality ranking functions. McRank [Li et al. 2007] also proposes a method for converting multi-class classification result to ranking result based on the boosting tree method. Recent developments in Yahoo! Learning to Rank Challenge also show that tree-based boosting and bagging are among the most effective approaches [Burges 2010; Pavlov and Brunk 2010; Mohan et al. 2010; Sorokina 2010; Gulin and Kuralenok 2010; Geurts 2010]. That means our tree adaptation approach can be directly applied to many of these effective tree-based methods for domain adaptation.

Since ranking cares only about the relative ordering between any pair of documents regarding to a specific query, rather than accurate prediction of grades, some algorithms try to model this ordering relationship. The second category of algorithms proposes to take pairwise data as training data and develops pairwise ranking functions. The representative algorithms include Ranking SVM [Joachims 2002], RankNet [Burges et al. 2005], RankBoost [Freund et al. 2003], and GBRank [Zheng et al. 2007], etc. The third category is the listwise approach, which tackles the ranking problem directly by adopting listwise loss functions, or directly optimizing information retrieval evaluation measures such as DCG, NDCG or MAP [Jarvelin and Kekalainen 2000]. The typical algorithms are LambdaRank[Burges et al. 2006], ListNet[Cao et al. 2007], and AdaRank[Xu and Li 2007].

Model adaptation has been of great interest in several areas, in particular, natural language processing, speech recognition and recently information retrieval, for addressing the problem of insufficient labeled training data. The classical approach is to treat the source domain data as “prior knowledge” and then to estimate maximum a posterior (MAP) values for the model parameters under this prior distribution for the target domain. This approach has been applied successfully to language modeling [Bacchiani and Roark 2003], parsing [Hwa 1999] and tagging [Blitzer et al. 2006]. In speech recognition, the maximum likelihood linear regression (MLLR) approach is also proposed for speaker adaptation [Leggetter and Woodland 1995]. The problem of distributional difference between the source domain and the target domain is formally addressed by the paper [Daumé III and Marcu 2006], which is further decomposed as 1) the difference between the prior distributions of feature vectors and 2) the difference between the label distributions [Jiang and Zhai 2007]. Jiang et al. [Jiang and Zhai 2007] also used a simple data combination technique by appropriately overweighting the target domain data in training. As we will show in this paper, overweighting the entire target-domain data may not give satisfactory results for ranking adaptation. The challenge is to appropriately assign different weights to different examples in terms of the distributional difference and sample importance. In contrast, our tree adaptation technique can automatically adapt to the fine-grained distribution difference between the source domain and the target domain [Chen et al. 2008]. There have been theoretical studies on the bounds of adaptation performance in terms of the distributional difference between the domains [Ben-david et al. 2007; Blitzer et al. 2008; Mansour et al. 2009]. Long et al. [Long et al. 2009] propose a general risk minimization framework to iteratively tune the individual sample weights for training data from the source domain. We will compare our approach to this risk minimization based weight tuning approach.

In addition to our work, there are some recent developments for adaptation in information retrieval. Gao and Wu et al. [Wu et al. 2008; Gao et al. 2009] comparatively studied the model interpolation method, i.e., linear combination of the source domain model and the target domain model, and a gradient boosting method similar to our additive model, i.e., appending trees to the source domain model to minimize the target domain prediction error only. Geng et al. [Geng et al. 2009] formulate the adaptation problem in web search under the framework of quadratic programming and thus a method is similar to SVM can be applied. Since the recent developments on learning to rank have shown that tree-based boosting and bagging are among the most popular and effective approaches [Burges 2010; Pavlov and Brunk 2010; Mohan et al. 2010; Sorokina 2010; Gulin and Kuralenok 2010; Geurts 2010], we believe it should be in high priority to develop adaptation methods for tree-based ranking functions.

Transfer learning [Pan and Yang 2010] considers a category of learning problems that involve two or more different but correlated tasks or domains in terms of data distribution, which certainly includes the domain adaptation problem. According to the type of data available for training and the learning methodology, transfer learning can be categorized into several categories: both source and target domains having labeled data (i.e., the same setting we used in this paper) [Dai et al. 2007; Long et al. 2009], only the source domain having labeled data [Liao et al. 2005; Wu and Dietterich 2004; Blitzer et al. 2006], both domains having unlabeled data only (self-taught learning) [Raina et al. 2007; Dai et al. 2008; Wang et al. 2008]. Transfer learning algorithms are also developed to find good feature representations to minimize domain divergence and model error [Argyriou et al. 2007; Ando and Zhang 2005], and to transfer knowledge of model parameters [Evgeniou and Pontil 2004]. In experiments, we will compare our algorithm to a transfer regression algorithm [Long et al. 2009] and a SVM based transfer learning algorithm [Geng et al. 2009] on the ranking problem.

3. PRELIMINARY

Tree adaptation follows the general GBT training framework, while the major algorithms are more related to the mechanism of generating regression trees. In order to design effective adaptation algorithms on trees, we need to understand the structure of a regression tree and how the boosted trees are generated. In this section, we will first give the definition of training data for the ranking problem, and then briefly describe how a regression tree is generated, which is the main component of GBT. This section will also setup the notations used later in this paper.

3.1. Training Data for Learning to Rank

Multi-grade Labeled Data: In learning to rank approaches, the expert judged results (query, document, grade) are transformed to training examples $\{(\mathbf{x}_i, y_i)\}$. Here, $\mathbf{x}_i$ represents a feature vector describing the features associated with the (query, document) pair. y_i is the target value from a set of grades, e.g., a five-grade scheme, which represents different levels of relevance between the query and the document. The grade is determined by the relevance expert for each (query, docuemnt) pair. The task of learning to rank is thus transformed to learning a function from the training examples $\{(\mathbf{x}_i, y_i)\}$, so that the learned function can predict the target value for any (query, document) pair if its feature vector is provided. Since such a ranking function outputs a score for each (query, document) pair, we can simply sort the scores for a set of (query, document) pairs and display the sorted list of documents for the given query.

Pairwise Data: A pairwise training example is defined as (query, document1, document2). Let's use (q_i, d_i^1, d_i^2) to represent it and also define the preference relationship as $(q_i, d_i^1) \succ (q_i, d_i^2)$, i.e., for query q_i, d_i^1 is more relevant than than d_i^2. Although we can use pairwise data for training, it is inappropriate to generate pairwise ranking results. Since it would need to make $O(n^2)$ prediction for n documents and then sort the $O(n^2)$ pairwise results, the cost would be much higher than the score-based sorting. Therefore, normally we still prefer to use the pairwise data to train a scoring based ranking function.

Feature Extraction: We briefly describe some of the typical features available for ranking. For each query-document pair, there are three categories of features:

— Features modeling the user query, q. They do not change over different documents in the document set $\mathcal{D}$. This type of features may include the number of terms in the query, the frequency of a term in the corpus, and query classification, e.g., name query, adult query, or navigational query. In total, over ten query features are used in training.
— Features modeling the web document, d. They are constant crossing all the queries q in the query set $\mathcal{Q}$. This type of features may include, the number of inbound links to the document, the number of distinct anchor-texts for the document, the language/region identity of the document, and the classification of the document, etc. About tens of such features are used in training.

— Features modeling the query-document relationship. They describe the matching between the query q and the document d. Such features may include, the frequency of each query term in the title of the document d, the frequency of each term in the anchor-texts of the document d, etc. Since the matching can happen in different sections of a document, hundreds of such features can be defined and used in training.

3.2. Learning a Regression Tree

With multi-grade labeled training examples, one straightforward learning method is order regression, which can be learned with many algorithms. We will use gradient boosting trees in this paper for its superb modeling quality and flexible structure. The basic component of GBT is regression tree [Hastie et al. 2001]. For better understanding of the tree adaptation algorithms, we will give sufficient details of learning a regression tree in this section. Figure 1 shows a sample regression tree, which is a binary tree with one predicate at each internal node. The predicate consists of a variable (feature) and a splitting value, typically in form of $F < \tau$?. In such a tree, an internal tree node partitions the training data that reach the node into two parts, with the corresponding predicate defined in the node. The tree is grown with a top-down manner, i.e., starting from the root and terminating with certain satisfied condition, e.g., the fixed number of leaf nodes. In the following, we describe how the training algorithm decides which feature and splitting value are used for growing child nodes.

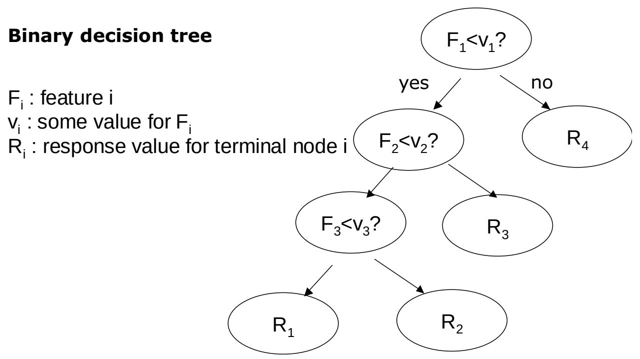


Fig. 1. A sample regression tree

First, splitting a leaf node to grow a tree should give some kind of "gain", namely, optimizing the goal of regression, i.e., minimizing the square error between the predicted value and the target value. We assume that there are n_i training records reaching the node i, each of which, $\mathbf{x}_j$, has a target value r_{ij} to fit at node i. $r_{ij} = y_j$ if the current node is the root, otherwise, r_{ij} is the residual by fitting the parent node. It represents how well this example is fit so far from the existing part of tree. The best-effort predictor for all records falling onto the current node is the mean of all r_{ij}, i.e., $\hat{r}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} r_{ij}$ [Hastie et al. 2001]. With $\hat{r}_i$, the square error E for the current node is

$$E = \sum_{j=1}^{n_i}(r_{ij} - \hat{r}_i)^2$$

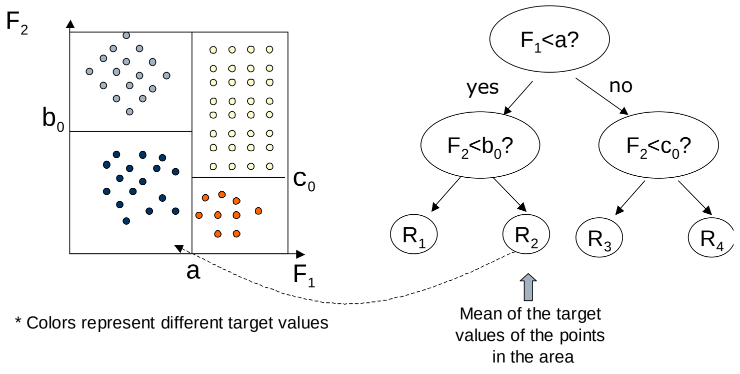
Finding the Best Split for a Node. Let F_p denote the feature and $v_{p,q}$ is a feasible value for F_p. $(F_p, v_{p,q})$ partitions the data into two parts: those records with $F_p < v_{p,q}$ go to the left subtree and the rest records go to the right subtree. After performing this partition, similarly, we can get the square error E_L for the left subtree, and E_R for the right subtree. We define the gain by splitting this node as $gain = E - E_L - E_R$. By scanning through all possible features and feasible splitting values for each feature, we can find the best splitting condition, which satisfies

$$argmin_{(F_p, v_{p,q})}\{E_L + E_R\}, \text{for all possible } p, q.$$

Finding the Best Node for Splitting. With the above criterion, a greedy search procedure can be applied to determine the leaf node that will bring the highest gain among all existing leaf nodes for splitting.

$$\text{node } i \text{ for splitting} = argmax_i\{gain_i\}, \text{for all leaf nodes.}$$

This is a hill-climbing procedure, which does not guarantee to get a globally optimal tree. Certainly, there are other strategies, but this one is very efficient especially when we have many features in the dataset, as the cost is linear in the number of features. We will use this tree growing strategy by default. Figure 2 shows a perfectly fitted tree to the underlying target value distribution. To extend it to general multidimensional cases, we can understand that each node represents a "multidimensional bounding box" defined by the disjointed partitioning conditions along the path from the root to that node. For example, the leaf node labeled with R_2 in Figure 2 is defined by the bounding box $F_1 < a \wedge F_2 < b_0$.



F2
b0
c0
a
F1
F1<a?
yes
no
F2<b0?
F2<c0?
R1
R2
R3
R4
* Colors represent different target values
Mean of the target values of the points in the area

Fig. 2. A perfectly fitted tree

Calculating Leaf Node Response. In the above algorithm, during the growing phase, the predicted value $\hat{r}_i$ for the node i is recorded, and the residuals $r_{ij} - \hat{r}_i$ are used as the new target values for its child nodes. Let $\{\hat{r}_i^{(\omega)}$, node i in the path from the root to any node $\omega\}$ denote the predicted values along the path from the root to the leaf node t. Since each $\hat{r}_i^{(\omega)}$ fits the residual from its parent, the predicted response R_ω for the node ω should be defined as

$$R_\omega = \sum_i r_i^{(\omega)} \tag{1}$$

An equivalent way to computing the response is to simply find the mean of the target values for all points falling onto the leaf node. However, these two methods will result in quite different adaptation strategies, which will be discussed in "response value adaptation".

3.3. Learning Gradient Boosting Trees

Gradient boosting trees can be used to model both classification and regression problems. The boosted trees are a series of regression trees, denoted by $h_i(\mathbf{x})$. The final function is based on these regression trees.

$$H(\mathbf{x}) = \sum_{i=1}^{k} \gamma_i h_i(\mathbf{x})$$

where γ_i is the learning rate, which is often small, e.g., 0.05. A formal description of the training algorithm can be found in the literature [Friedman 2001]. The GBT learning method trains the k-th tree, based on the previous trees h_j, $1 \leq j < k$, with a set of random samples from the training dataset (*Stochastic Gradient Boosting*). The steps can be briefly described as follows.

(1) randomly sample the training data to get a subset of training examples $\mathcal{S}_k$;
(2) set the target r_i of the example in $\mathcal{S}_k$ to the original target y_i for k=1, or to the residual of the previous trees h_j, for $k > 1$, $1 \leq j < k$, $r_i = y_i - \sum_{j=1}^{k-1} \gamma_j h_j(\mathbf{x}_i)$.
(3) train the regression tree h_k with the examples $\{(\mathbf{x}_i, r_i)\}$, $\mathbf{x}_i \in \mathcal{S}_k$.

4. RANKING FUNCTION ADAPTATION BASED ON GRADIENT BOOSTING TREES

In this section, we first describe the basic challenge that the tree adaptation approach will address and justify why this approach will work. Then, we will present several tree adaptation algorithms in details.

4.1. Domain Adaptation for Tree-based Models

In this section, we describe the rationale behind the tree model adaptation and the special challenges for domain adaptation on the ranking problem. The idea of tree-based adaptation is a mix of discriminative and generative modeling.

Given a set of training examples, $\{\mathbf{x}_i, y_i\}$, the goal of training an effective model is to find a function approximating the conditional label distribution $p(y|\mathbf{x})$. $p(y|\mathbf{x})$ can be directly learned (discriminative modeling) or learned based on the Bayes rule $p(y|\mathbf{x}) = p(\mathbf{x}, y)/p(\mathbf{x})$ (generative modeling), where $p(\mathbf{x})$ is the underlying feature vector distribution and $p(\mathbf{x}, y)$ is the joint distribution of labels and feature vectors. Let s denote the source domain and t the target domain. The similarity between the source feature vector distribution $p^s(\mathbf{x})$ and the target distribution $p^t(\mathbf{x})$ is an important intuition behind the tree-based adaptation. In the following, we use feature vector distributions, sampling bias, and noisy labels to explain the situations that domain adaptation may work for the ranking problem.

— In the ranking problem, a set of common features are defined for different domains. Normally, the relevant training examples are far less than the irrelevant examples in the distribution $p(\mathbf{x})$. Therefore, in constructing the training data, some biases have already existed in collecting a sufficient amount of *relevant* examples. When the size of target data is small, the sample distribution $\tilde{p}^t(\mathbf{x})$ may significantly deviate from the real distribution $p^t(\mathbf{x})$. In this case, we hope that the source domain data can patch the missing part of the distribution, and thus adaptation may work. (Figure 3 illustrates this situation - the specific description on the illustration comes later).
— At the local areas that have similar sample distribution, i.e., $\tilde{p}^s(\mathbf{x}) \approx \tilde{p}^t(\mathbf{x})$, the label distribution may still be different (Figure 4). This is especially true when there are a significant amount of noisy labels in the target domain. In this situation, the source domain data may help correct the bias from the noisy labels.

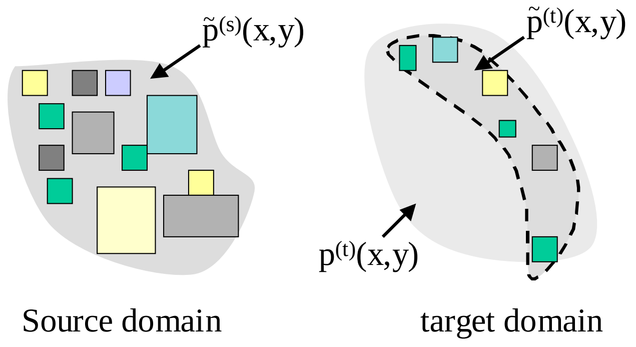


Fig. 3. Source domain data may be able to patch the missing part of target domain.

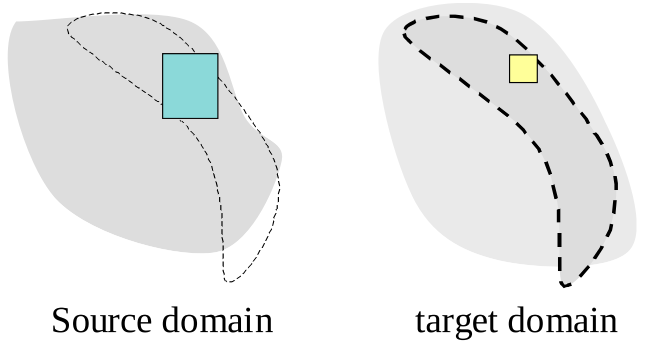


Fig. 4. In overlapped part of distribution, the label distribution may still differ between the source and the target.

— In addition to the intersecting part of distribution, an effective adaptation model also has to incorporate the target domain's unique part of the distribution (Figure 6).

Next, we explain how tree models fit in these three situations. A tree model (for both regression and classification) partitions the sample feature space and can be used to roughly describe $\tilde{p}(\mathrm{x})$, the accuracy of which depends on the sampling method and the amount of samples. Figure 3 and 4 can be used to illustrate how a multi-grade training data space is partitioned by a regression tree. The small blocks in the figures represent the local areas in the multidimensional space ("the multidimensional bounding boxes") that are covered by the training data. Different colors represent different average response values for the blocks. In regression tree modeling, each leaf node tries to approximately model one of these blocks, and each internal node groups several nearby blocks to minimize the prediction error.

In the following, we define the concept of distribution intersection to describe the domains's relationship in terms of $\tilde{p}(\mathrm{x})$, which is a necessary condition for effective tree-based adaptation.

Definition 4.1. Let x^s and x^t be sample data from the source domain and target domain, respectively. $p^s(\mathrm{x})$ and $p^t(\mathrm{x})$ are not intersecting, if $p^s(\mathrm{x}^t) = 0$ and $p^t(\mathrm{x}^s) = 0$ for all x^s and x^t; Otherwise, $p^s(\mathrm{x})$ and $p^t(\mathrm{x})$ are intersecting.

Figure 5 shows an example of distributions having no intersection in the one dimension situation.

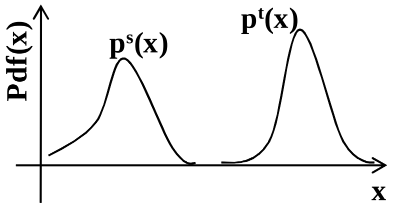


Fig. 5. Distributions are not intersecting.

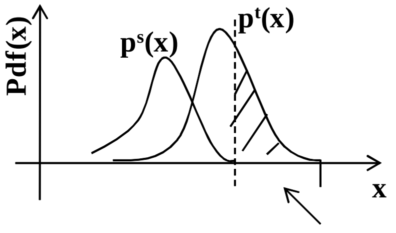


Fig. 6. Distributions are intersecting. The shaded part is unique to the target domain.

An effective tree adaptation algorithm depends on the following assumptions: (1) $p^s(\mathrm{x})$ and $p^t(\mathrm{x})$ are intersecting; (2) the source data can be helpful in terms of patching the target distribution and demoting the noise labels.

The basic challenge to adaptation is to appropriately combine the two sets of data or two models. A simple global weighting scheme (i.e., overweighting one set of data over another set with only one global weight) may not work well for the following reason: it ignores the detailed differences between the sample distribution $\tilde{p}^t(\mathrm{x})$ and $\tilde{p}^s(\mathrm{x})$, which results in either overfitting or over-demoting the domain-specific part of the model. A feasible solution should be able to (1) locate the distributional differences, and (2) appropriately weight and combine two sets of data or models. Tree adaptation provides a convenient way to locate the part of the model that needs tuning, and to automatically weight different part of target data according to both source and target data distributions. As a result, tree adaptation models are more robust and less possible to overfit the small data from the target domain.

4.2. Trada Tree Adaptation Algorithms

The basic components of tree adaptation include 1) using the base model to partition the new dataset, i.e., approximating $\tilde{p}^t(\mathrm{x})$ with $\tilde{p}^s(\mathrm{x})$; 2) properly weighting the samples based on locality; 3) and finely tuning the partition based on both source and target data distributions.

The tree adaptation algorithms are closely related to the mechanism of regression tree modeling that we have discussed. We can understand tree adaptation from the perspective of multidimensional partitioning of the sample space $\tilde{p}(\mathrm{x})$. In a regression tree, each path from root to any node represents a multidimensional bounding box, which is a subspace of $\tilde{p}(\mathrm{x})$. In particular, it is worth noting that from top down the parent bounding box also encloses the children bounding boxes, and the records falling to the same box will get the same predicted value (and response). By inheriting the tree structure from the base model, we try to tune the target data distribution $\tilde{p}^t(\mathrm{x})$ based on the source data distribution $\tilde{p}^s(\mathrm{x})$.

In tree adaptation, we will slightly tune the response (and also the boundary of the bounding box) based on the local distributions of the source and target data. This process will be done node by node, from the root to leaves. By doing so, we not only incorporate the distribution learned by the base model to the new model, but also take into consideration the subtle distributional differences represented by the target domain. Due to the complexity of the tree structure, there are probably numerous strategies for tuning the base model. According to the intensity level of changing the base model, we choose to present a few representative algorithms.

Algorithm 1. Tuning Responses Only. The first strategy is fixing the bounding boxes and tuning responses only. This strategy and some of the later ones employ the similar local-distribution-based tuning algorithm. Namely, we assume there are a certain number of records, n_0, from the source domain D_0, i.e., the training data for the base model, and n_1 from D_1, the small training data for the target domain, falling onto the same bounding box, respectively. We allow the two populations to vote for the the final decision about the response. By appropriately aligning up the size difference between the two sets of data, we generate the weight for each vote and then calculate the tuned response.

Concretely, we calculate the weights as follows. First, let a leaf node at the base model be associated with response R_0, and there are n_0 records from the source training data falling onto this node. Next, we apply the target domain data to the tree, by fixing the splitting condition, to get the response value R_1. Similarly, we know n_1 target domain records falling onto that node. We assume the corresponding probabilities of a record belonging to the source and target domains, are are p_0 and $(1-p_0)$, respectively. A balanced estimate of the combined value is calculated by

$$f(R_0, R_1, p_0) = p_0 \times R_0 + (1 - p_0) \times R_1 \tag{2}$$

$f(R_0, R_1, p_0)$ is used as the tuned response value for this leaf node. Now, we should estimate p_0 based on the two local sample populations on this node. Since these two original datasets have unequal size, we may need to scale up the small data with an appropriate factor β. Based on the sample populations and β, we estimate p_0 with

$$\hat{p}_0 = \frac{n_0}{n_0 + \beta \times n_1} \tag{3}$$

The appropriate β can be determined through cross-validation. This distribution-based estimation will also applied to boundary tuning later. Plugging 3 into Formula 2, we expand the parameters to $f(R_0, R_1, n_0, n_1, \beta)$. Formula 3 says that, when $n_1 \ll n_0$, the original response is almost used as the response in the new model.

As we have mentioned, each node has a predicted value trying to fit the residual from its parent and Formula 1 calculates the leaf node response based on the series of residual prediction $\{\hat{r}_i\}$ on the path from root to the node. Alternatively, we can adapt $\hat{r}_i$ on each node to get $\hat{r}'_i$. Let $r_{0,i}$ and $r_{1,i}$ be the predicted values at the node i by applying the source data and the target data, respectively. Also, let $n_{0,i}$, $n_{1,i}$ be the number of source records and target records falling onto the node i, respectively. A layer-by-layer tuning strategy can be represented by Eq. 4, where the function f is the expanded form of Formula 2.

$$R_t = \sum_{i \text{ in the path}} f(r_{0,i}, r_{1,i}, n_{0,i}, n_{1,i}, \beta) \tag{4}$$

This layer-by-layer tuning strategy considers more global distribution of the two datasets, while the leaf-only tuning strategy (Formula 2) focuses more on the local distribution. The layer-by-layer strategy actually smoothes out the tuning process, making the change over nearby boxes less dramatic. In practice, we have observed that the layer-by-layer strategy indeed gives better results.

Algorithm 2. Tuning Both Bounding Boxes and Responses. This algorithm more aggressively tunes the base tree model. As we have described, each internal node in the path from the root to the leaf node is associated with a predicate, in form of feature $F < \tau$, which makes one of the dimensions of the bounding box represented by the path. In this algorithm, we still keep the the feature F unchanged, while tuning both the threshold τ and the corresponding node response.

Assume that the current node has a split with feature F, $F < v_0$.

(1) calculate the weight $\hat{p}_0$ with Eq. 3;
(2) partition the new data with the specific feature F and find the best split point v_1;
(3) adjust the split by

$$v_{comb} = \hat{p}_0 \times v_0 + (1 - \hat{p}_0) \times v_1$$

(4) re-partition the new data with the condition $F < v_{comb}$;
(5) adjust the response for the current node with Eq. 2
(6) move to the child nodes and repeat the above steps.

Figure 7 illustrates the basic steps in the Adaptation Algorithm 2 for one node adaptation. In the figure, the top left tree is the base model and the right process has the major split adaptation steps. There are two threads going on: one is generating and updating the tree for the new data on the right side, where the new data is the part of data that goes through the ancestor nodes. The other on the left is updating the base tree, while still preserving the information of source data distribution. We will use the output of the left side as the final adapted tree. This process is repeated for any nodes in the subtrees and applied to all trees in the base model.

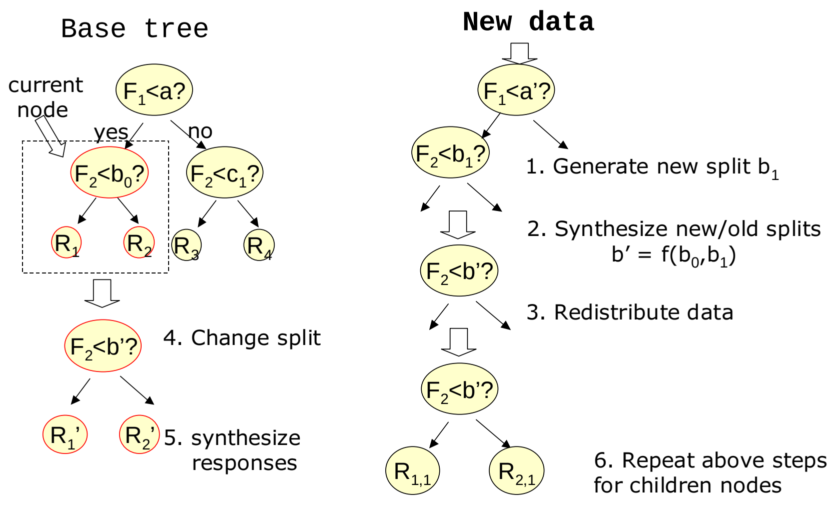


Fig. 7. A sample algorithm for tree adaptation

Note that in both Algorithm 1 and 2, some branches of the base model may not be reached by the target domain examples. It would be difficult to assert whether trimming such branches will result in better model or not. Instead, we will evaluate both trimming and non-trimming in experiments.

Algorithm 3: Appending Trees to Incorporate New Features. Since the base model is trained for the source domain, it may not contain the features and the part of feature distribution (Figure 6) that are specific to the target domain. Tuning the base model with Algorithm 1&2 cannot address this problem. Fortunately, the GBT training method can be used to uniquely address this problem – we can expend the adapted model by appending a certain number of additional trees. These additional trees are trained on the residuals from the adapted base model. This method is easy to implement and less likely to overfit the target domain data as it follows the general boosting framework.

One may raise a question: why not solely append trees without applying Algorithm 1 or 2 on the existing trees (as known as *additive modeling*), to achieve the adaptation goal? Additive modeling may not be sufficient because the working mechanism of GBT limits the effect of the appended trees. The earlier trees dominate the predicted value, while the appended trees only finely tune the result. We will show that Trada algorithms with appended trees have better performance than additive modeling (Figure 10).

Other Algorithms. We have only presented a subset of possible directions for adapting GBTs. There are more tuning algorithms such as the following ones.

— *Trimming Branches:* When we repartition the new data with the adjusted split value, it is possible that some branches will have little (or even no) new data falling onto. There are options for trimming or not trimming such branches. Both have equal possibility to benefit or to harm the result. 1) Without trimming, we expect the branch learned from the source data is still useful even though there is no example from the new domain. 2) Since there is no example for tuning the branch, new examples from the target domain may be processed wrongly. However, since one of the purposes of adaptation is to remedy the problems caused by missing training examples, we conjecture that case one may have better chances. We depend on experiments to validate it.

— *Growing Branches:* We can also further partition a leaf node where the new data have a proportionlly much higher population than the old data. This may give potential benefits: we can not only incorporate new features but also refine the partition. The risk is that growing the leaf with only the small data may overfit the target domain. It is also difficult to determine whether we should grow a leaf at a certain condition. We experimented in a conservative way: when the number of new examples, n_1 at the node satisfies the condition $\eta \cdot n_1 > n_0$, we grow the leaf. n_0 is the number of examples from the source domain at this node and $\eta \geq 1$ is some value scaling up n_1. The appropriate η can be determined by cross-validation. However, we consider this option might be inferior to appending more trees to the end of the base tree model, as appending trees are less likely to overfit the data as previous experiments on boosting trees have shown [Friedman 2001; Schapire 2003].

— *Feeding Back Split Adjustment to the Source Domain:* More aggressively, we can also feedback the adjusted split value to the source domain, repartitioning the old data with the new split value to reversely propagate the change. However, the concern is that this dramatic model change is triggered by a small number of examples in the new domain, which may overfit the new domain.

Our assumption is that the base model should have captured most common distributions shared by different domains. Therefore, we conjecture that dramatic structural changes triggered by the small target domain data may result in unsatisfactory models. That is the reason we separate the discussion on these additional options from the three basic algorithms. We will conduct experiments to study whether these algorithmic options have advantages.

4.3. Trada Algorithm with Stochastic Gradient Boosting Trees

Previous sections have described the basic ideas used in the tree adaptation algorithm for a single regression tree. Below we give a complete description on the tree adaptation algorithm in the framework of gradient boosting trees. We use $\{v_j\}$ to represent a set of objects indexed by j. For example, $\{h_j\}$ represents a set of boosting regression trees and $\{t_j\}$ represents a set of new target values for the feature vectors $\{\mathrm{x}_j\}$, respectively.

ALGORITHM 1: Trada-Boosting-Trees($\{h_i\}$, D^s, D^t, l, γ, s, b, mode)

```
input  : $h_i, i = 1 \ldots p$, are trees from the source domain,
         $D^s$: source training data $\{(\mathbf{x}_i^s, y_i^s)\}$, $D^t$: target training data $\{(\mathbf{x}_i^t, y_i^t)\}$
         $q$: the number of output trees; $\gamma$: learning rate; $s$: sampling rate for GBT; $b$: the number
of leaf nodes; $mode$: adaptation algorithm
output: $h'_i, i = 1 \ldots q, q \geq p$, adapted trees
for each $(\mathbf{x}_i^t, y_i^t)$ do
    $t_i \leftarrow y_i$;
end
for $i = 1$ to $p$ do
    $h'_i$ = Trada($h_i$, $D^s$, $D^t$, $\{t_j\}$, mode);
    for each $(\mathbf{x}_j^t, y_j^t)$ do
        $t_j \leftarrow y_j^t - \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_j)$;
    end
end
for $i = p + 1$ to $q$ do
    // apply normal gradient boosting tree algorithms to append trees;
    $h'_i$ = StochasticRegressionTree($D^t$, $\{t_j\}$, $s$, $b$);
    for each $(\mathbf{x}_j^t, y_j^t)$ do
        $t_j \leftarrow y_j^t - \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_j)$;
    end
end
```

In Algorithm 1, the previously presented Trada algorithms are represented as the Trada() function in this framework (different algorithms are distinguished by the "mode" parameter). If the number of output trees is larger than the original trees, we will use the normal GBT algorithm to train the additional trees and append them to the final model. The StochasticRegressionTree() function randomly samples a number of data records and builds a regression tree on the sample set. The sampling step was shown to improve the overall performance [Friedman and Popescu 2003]. We ignored other parameters in training regression trees such as the sampling rate and the number of terminal nodes.

5. PAIRWISE TRADA: UTILIZING THE PAIRWISE PREFERENCE DATA

In this section, we first present our previously developed GBRank framework for training ranking functions with pairwise preference data, and then describe how to combine the Trada algorithm and the GBRank framework to incorporate pairwise data in adaptation.

5.1. Training Models with Pairwise Preference Data

The unique advantage of pairwise preference data is that they can be automatically extracted from user clickthrough log without paying the time and financial cost as normally we do to get multi-grade labeled data. However, there are two challenges in using pairwise preference data for training. First, the clickthrough log contains very noisy clicks that it is challenging to extract high-quality preference data from the log. There have been heuristics for extracting relevance pairs [Joachims 2002; Joachims et al. 2005] such as "skip-above" and "skip-next" pairs, and recently there are studies on which types of pairs are most useful in learning a ranking algorithm [Ji et al. 2009; Dong et al. 2009]. The second challenge is to design algorithms that can utilize pairwise data for training. This problem can be addressed by finding an appropriate loss function for minimizing the pairwise ranking errors. There have been algorithms such as Ranking SVM [Joachims 2002], RankNet [Burges et al. 2005], RankBoost [Freund et al. 2003], and GBRank [Zheng et al. 2007]. As extracting pairwise data is not the focus of this paper, we will simply use the same method used for extracting pairwise data in papers [Ji et al. 2009; Dong et al. 2009] and interested readers should refer to the related literatures. This section focuses on the basic idea of using the pairwise data in the regression framework, which will be extended for adaptation. Below we will give the basic idea of the GBRank algorithm.

ALGORITHM 2: Trada-Pairwise($\{h_i\}$, D_p^s, D_p^t, l, γ, τ, s,b, mode)

```
input  : $h_i, i = 1 \dots p$, are trees from the source domain trained with GBRank,
         $D_p^s$: pairwise source training data $\{(\mathbf{x}_{i,1}^s, \mathbf{x}_{i,2}^s, \mathbf{x}_{i,1}^s \succ \mathbf{x}_{i,2}^s)\}$,
         $D_p^t$: pairwise target training data $\{(\mathbf{x}_{i,1}^t, \mathbf{x}_{i,2}^t, \mathbf{x}_{i,1}^t \succ \mathbf{x}_{i,2}^t)\}$,
         $q$: the number of output trees; $\gamma$: learning rate; $\tau$: the margin; $s$: sampling rate for GBT;
$b$: the number of leaf nodes; $mode$: adaptation algorithm
output: $h'_i, i = 1 \dots q, q \geq p$, adapted trees
for each $(\mathbf{x}_{i,1}^t, \mathbf{x}_{i,2}^t, \mathbf{x}_{i,1}^t \succ \mathbf{x}_{i,2}^t)$ do
    $t_{i,1} \leftarrow \tau$ for $\mathbf{x}_{i,1}^t$;
    $t_{i,2} \leftarrow 0$ for $\mathbf{x}_{i,2}^t$;
end
for $i = 1$ to $p$ do
    $h'_i$ = Trada($h_i$, $D_p^s$, $D_p^t$, $\{t_j\}$, mode);
    for each $(\mathbf{x}_{j,1}^t, \mathbf{x}_{j,2}^t, \mathbf{x}_{i,1}^t \succ \mathbf{x}_{i,2}^t)$ do
        $v_1 = \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_{j,1}^t)$ ;
        $v_2 = \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_{j,1}^t)$ ;
        if $v_1 < v_2$ then
            $t_{j,1} \leftarrow v_1 + \tau$ ;
            $t_{j,2} \leftarrow v_2 - \tau$ ;
        end
    end
end
for $i = p + 1$ to $q$ do
    // apply GBRank to append trees;
    $h'_i$ = StochasticRegressionTree($D_p^t$, $\{t_j\}$, $s$, $b$);
    for each $(\mathbf{x}_{j,1}^t, \mathbf{x}_{j,2}^t, \mathbf{x}_{i,1}^t \succ \mathbf{x}_{i,2}^t)$ do
        $v_1 = \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_{j,1}^t)$ ;
        $v_2 = \sum_{k=1}^{i} \gamma h'_k(\mathbf{x}_{j,1}^t)$ ;
        if $v_1 < v_2$ then
            $t_{j,1} \leftarrow v_1 + \tau$ ;
            $t_{j,2} \leftarrow v_2 - \tau$ ;
        end
    end
end
```

The GBRank algorithm is a variant of Gradient Boosting Trees algorithm. It uses the same regression tree training algorithm for training each tree, while it changes the gradient boosting algorithm, i.e., the way changing the target values for training the next tree. In the original GBT algorithm, since the purpose is to minimize the square loss between the predicted value $\hat{r}_i$ and the target value r_i, the target value for the next tree is set to $r_i - \hat{r}_i$. For pairwise data, the purpose is to reduce the number of contradictory pairs, i.e., the predicted pairwise relationship $d_i^1 \prec d_i^2$ that contradicts the expected relationship $d_i^1 \succ d_i^2$. The next tree should try to reduce such prediction errors. Let $h(q, d)$ represent the predicted score for query q and document d with the currently trained j GBT trees, i.e., $h(q, d) = \sum_{k=1}^{j} h_k(\mathbf{x}_d)$ and $\mathbf{x}_d$ is the feature vector for the query/document tuple (q, d). If $(q_i, d_i^1) \succ (q_i, d_i^2)$ we define $h(q_i, d_i^1) > h(q_i, d_i^2)$. Let τ be a positive constant and N be the number of instances. The contradictory pair loss function is defined as

$$L(h) = \sum_{i=1}^{N} (\max\{0, h(q_i, d_i^2) - h(q_i, d_i^1) + \tau\})^2 \tag{5}$$

To minimize this loss function, $h(q_i, d_i^1)$ has to be larger than $h(q_i, d_i^2)$ with a margin τ. As this loss function is only determined by the values of $h(q_i, d_i^1)$ and $h(q_i, d_i^2)$, we use an approximate algorithm and the gradient boosting framework to minimize this loss. Namely, at the beginning of the algorithm, for each pair (q_i, d_i^1, d_i^2) we assign a score τ to the document/query tuple (q_i, d_i^1) and 0 to (q_i, d_i^2). At the end of each round, when we detect a pair having a contradictory relationship we revise the target value $t_{q,d}$ for each document/query tuple in the pair,

$$t_{q_i, d_i^1} = h(q_i, d_i^1) + \tau, \qquad t_{q_i, d_i^2} = h(q_i, d_i^2) - \tau.$$

By doing so, the next round of regression tree training will try to correct the prediction so that the pairwise relationship will be met. Note that this algorithm is simple to implement. The only difference from the basic GBT is the target value setting at the end of each round. In practice, we have shown that this algorithm is very effective in both minimizing pairwise contradiction rate and improving ranking quality, with the pairwise data converted directly from the multi-grade labeled training data [Zheng et al. 2007].

5.2. Adapting Models with Pairwise Preference Data

In addition to a small amount of multi-grade labeled training data, now we also have pairwise training data extracted from the user clickthrough log that is based on users' reactions to an experimental ranking function (e.g., one reasonably good ranking function from another domain). Recent study shows that using only the pairwise data for training will not be sufficient for getting a good ranking function [Dong et al. 2009]. We believe a better way is to adapt the base function to the pairwise data and thus take advantage of both sets of data. In this section, we describe the "Pairwise Trada" algorithm for this purpose.

In Pairwise Trada, we assume the base function is trained with the GBRank algorithm and pairwise data (which could be converted from multigrade labeled data) from the source domain. The basic idea is to use the same Trada algorithm for adapting each individual regression tree while changing the process of setting the target values $\{t_j\}$ for training the next tree as GBRank does. The intuition is the same as the Trada algorithm. Only those local areas in the distribution space that have been overlapped with the new pairwise examples are fine adjusted to reduce the contradictory pairs in the new domain.

Algorithm 2 in Appendix gives the detail of this algorithm. An additional advantage of this algorithm is that we can utilize both sets of data in the target domain: the multi-grade labeled data and the pairwise preference data from the clickthrough logs to build a model. What we need to do is to simply convert the multi-grade labeled data into pairwise data so that Pairwise Trada can use both sets of data.

6. EXPERIMENTS

The experiments are organized in two groups. The first group is dedicated to the basic Trada algorithm and the second to the Pairwise Trada algorithm. For the first

Table I. Public Yahoo! Learning to Rank Datasets

	total number of queries	total number of documents
S_0	19,944	473,134
S_1	1,266	34,815

S_0 and S_1 are public datasets from the Yahoo! learning to rank challenge. They are used to study parameter tuning and perform detailed comparison.

Table II. Real Datasets

	total number of queries	total number of documents
D_0	6,012	146,307
D_1	1,282	37,952
D_2	823	12,092
D_3	804	34,276
D_4	540	16,950
D_5	1,258	29,165

Real datasets from a major search engine, collected in 2008.

group of experiments, we set several goals in the experimental evaluation as follows. First, we want to see how the settings (the base model, the β parameter, the number of additive trees) can affect the quality of adaptation with varying size of the small training data from the target domain; Second, we want to compare the effectiveness of different adaptation algorithms we have presented; Finally, the adaptation approach is compared to other methods, including (1) the base model only; (2) models that are trained only with the small data from the target domain; and (3) data combination models that combine the two sets of data for training.

The second group of experiments is dedicated to Pairwise Trada. We will investigate two problems: (1) whether Pairwise Trada can utilize the click data to improve the base model and (2) whether the combination of click data and the small amount of multi-grade labeled data can help further improve the model.

6.1. Datasets

We used three groups of data in experiments. The first group of data is the public yahoo learning to rank data [2] (Table I). It consists of two labeled sets. The set one (S0) has 19,944 queries and 473,134 documents. The set two (S1) has 1,266 queries and 34,815 documents. We use these two datasets (S0 for the source domain and S1 for the target domain) to study the parameter settings and the algorithms for the Trada approach.

The second group of data is from a major web search engine. The domain D_0 serves as the source domain and other domains are the target domains. The models used $200 \sim 300$ features scattered in the three categories we have described in Section 3. All training examples are labeled by relevance experts and have been divided into five batches according to the time the data were labeled. A five-fold cross validation will use this natural split to take the time factor into consideration. Table II summarizes the size of the datasets. These datasets are used to compare the performance of different adaptation methods.

The third group of datasets is used to study Pairwise Trada (Table III). It includes five more datasets from other five domains E_1 to E_5, respectively, which have both small labeled target domain data and pairwise preference data extracted from the clickthrough log. We use the heuristic rules which is introduced by the paper [Dong et al. 2009] to extract pairwise preference data from the clickthrough log. This ap-

[2]http://learningtorankchallenge.yahoo.com/

Table III. Real Datasets with User Preference

	multi-grade labeled examples	preference pairs from clicks
E_1	91,638	73,977
E_2	75,989	108,350
E_3	243,790	194,114
E_4	174,435	166,396
E_5	101,076	94,875

Size of multi-grade labeled training data and preference pairs in the target domains, collected in 2009.

proach yields both "skip-next" and "skip-above" pairs [Joachims et al. 2005]. We describe the method for extracting pairs as follows.

Namely, for the same query, we only look at the user clicks on the top ten results. For a tuple (q, url$_1$, url$_2$, pos$_1$, pos$_2$) where q is the query, url$_1$ and url$_2$ are urls representing two documents in the query result, pos$_1$ and pos$_2$ are ranking positions for the two documents with pos$_1$ $<$ pos$_2$, i.e., url$_1$ has higher rank than url$_2$. Let *imp* be the impression of the query result, i.e., the total number of user sessions submitted the same query. Let *cc* be the number of sessions that both url$_1$ and url$_2$ are clicked, *ncc* be the number that url$_1$ is not clicked but url$_2$ clicked, *cnc* be the number that url$_1$ is clicked but url$_2$ not clicked, and *ncnc* be the number that both are not clicked. The "skip-above" pairs are extracted with the following rule: if *ncc* is much larger than *cnc*, and $\frac{cc}{imp}$ and $\frac{ncnc}{imp}$ are very small ($<$ a threshold), then we say url$_1$ is skipped above url$_2$. Skip-above suggests strong preference on url$_2$ than url$_1$. Similar definition can be defined for "skip-next" [Joachims et al. 2005; Dong et al. 2009]. Table III includes both the number of labeled examples and the preference pairs extracted from the click-through log. Each of these datasets is split into three parts: 3/5 queries for training, 1/5 for validation and 1/5 for testing.

6.2. Evaluation Metrics

Discounted Cumulative Gain (DCG) [Jarvelin and Kekalainen 2000] is a metric designed for evaluating the quality of ranked list if the grades for items in the list are known. In our case, DCG is defined as follows. We use a five-grade labeling scheme for editorial judgment {'4', '3', '2', '1', '0'}, corresponding to the most relevant to the most irrelevant. Suppose there are k documents used for testing the query q and each query-document pair (q, d_i) in the test set is labeled with a grade l_i. The test result will give a list of the k documents that is sorted by the scores given by the ranking function H to each pair (q, d_i). Let $i = 1, \ldots, k$ be the order of the sorted documents. DCG_k score is computed for the sorted list as follows.

$$DCG_k = \sum_{i=1}^{k} \frac{2^{l_i} - 1}{\log(i+1)}$$

By definition, when reverse orderings happen at earlier positions (i is small), they will be punished more than those happening later. By doing so, we prefer that high quality results show up at the top of the ranked list. The normalized DCG (NDCG) is defined as

$$NDCG_k = DCG_k / Ideal\_DCG_k,$$

where Ideal_DCG$_k$ is the DCG for the ideal ranking result.

Each model test will generate a list of NDCGs corresponding to the list of testing queries. To compare the statistical significance between two results, we perform t-test [Lehmann and Casella 1998] on the two NDCG lists. If p-value < 0.05, the results are significantly different. t-test is only performed on the comparison for some experiments. For parameter tuning of the same model, we will show only average NDCG of the multi-fold cross validation. Note that the relevance differences between high-quality commercial web search engines are often less than 5%. Therefore, any small *statistically significant* improvements will have practical impact.

6.3. Algorithms for Comparison

Two sets of algorithms are compared to the proposed adaptation approach: the simple algorithms and the recently developed new algorithms.

Simple Algorithms. The simple adaptation methods include 1) the base model only, 2) the small-data model, 3) the additive model, and 4) the data combination model. We briefly describe each of them.

The base model is trained with a large training dataset from the source domain. In experiments, we find the parameter setting of the S_0 GBT model with the five-fold cross validation on S_1 data. For D_0 we used an empirical setting that was validated in production. Testing results show that the D_0 model with 300 trees, 12 leaf nodes, 0.05 learning rate, and 0.5 sampling rate works reasonably well for all target D_i domains. We will use the ranking quality of the base model on the target domain as the base line for comparison.

A small-data model is trained only with the small amount of training data from the target domain with the GBT method. Since the training data is small, it is highly possible that the model will be overfitting to the training data, which means it may not work well for new examples from the target domain in the future although it works well in cross-validation on the existing data.

An additive model does not change the base model but appends a few new trees to the base model, which are trained on the residuals of the new data on the base model. The training method is the default GBT method.

A data combination model [Jiang and Zhai 2007] uses the combination of two sets of training data: one from the source domain (in the above specific case, the 150K query-document-grade examples) and the other from the target domain, with possibly overweighting the target domain data. The same GBT training method is applied to the combined data to generate the final model.

We also compare our approach with the recently developed adaptation algorithms: the RiskMin algorithm [Long et al. 2009] and the RA-SVM algorithm [Geng et al. 2009].

RiskMin Algorithm. The RiskMin algorithm, based on the risk minimization framework [Vapnik 1999], iteratively adjusts the weights of the source domain data to minimize a risk function until convergence. It represents another way to fine tuning the individual sample weights. Note that this algorithm is originally designed for general-purpose regression adaptation[3], but it can be applied to the ranking problem. However, there is no reported result on ranking data yet. We will try this algorithm on the public datasets to compare its performance with ours.

The algorithm can be described in the following iterations. Initially, the sample weights are set to the same. In one iteration, the model is trained with weighted samples from both domains. In the next iteration, this model is used to test the source domain data, and the result is denoted as y_j^t for record j. The target domain data (n sample records) has a fixed weight $w_j = 1/n$ for all iterations, while the source domain

[3]Our algorithm is originally designed for the ranking problem, but it can also be applied to general-purpose regression problems.

sample weight is adjusted with

$$w_j = \frac{\alpha r(y_j^t, y_j^s)}{m}, \tag{6}$$

where m is the number of source domain samples, y_j^s is the source domain label for the record j, α is a predefined tuning parameter, and $r()$ is defined as $r(y_j^t, y_j^s) = \exp(-(y_j^t, y_j^s)^2)$. This weight tuning process continues until the model converges (no significant improvement on the model quality). This algorithm uses a similar idea to AdaBoost [Freund and Schapire 1999], reweighting the samples after each iteration. It is also similar to the TraDaBoost algorithm [Dai et al. 2007] that was designed for the transfer classification problem. We implemented the RiskMin algorithm with the GBT algorithm as the base learner.

RA-SVM algorithm. The RA-SVM algorithm [Geng et al. 2009] is an extension to the RankSVM algorithm [Joachims 2002]. It uses the model from the source domain to help reduce the bias introduced by the small training data from the target domain. Basically, it changes the SVM's primal objective function to combine the source domain function in training. RA-SVM, similar to RankSVM, works on pairwise data and generates a linear ranking function, denoted as f. Assume the feature vector for the query-document examples (q, d_i) and (q, d_j) are $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively; the relevance labels to the query q are y_i and y_j, respectively. The original data are transformed to pairwise data for learning using the following method. If $y_i > y_j$, we require only the ranking output $f(\mathbf{x}_i)$ greater than $f(\mathbf{x}_j)$; and if $y_i < y_j$, $f(\mathbf{x}_i)$ less than $f(\mathbf{x}_j)$. Thus, we can define the pairwise relevance by the sign of $f(\mathbf{x}_i) - f(\mathbf{x}_j)$. Since f is a linear function, $f(\mathbf{x}_i) - f(\mathbf{x}_j) = f(\mathbf{x}_i - \mathbf{x}_j)$. Therefore, the labeled training dataset is converted to pairwise training dataset $\{\mathbf{x}_i - \mathbf{x}_j, sign(y_i - y_j)\}$. The task is to find a good SVM classifier f for the pairwise training data.

For easier understanding, let the source domain model f_0 also trained with RankSVM: $f_0(\mathbf{x}) = \mathbf{w}_0^t\mathbf{x}$. Let $\{\mathbf{x}_i^{(p)}, y_i^{(p)}\}$ denote the m pairwise data records in the target domain and δ tune the contribution from the source domain function. The RA-SVM's objective function is represented as

$$\arg\min_{\mathbf{w}} = \frac{1-\delta}{2}\|\mathbf{w}\| + \frac{\delta}{2}\|\mathbf{w} - \mathbf{w}_0\| + C\sum_{i=1}^{m}\xi_i,$$
$$\text{subject to } y_i^{(p)} f(\mathbf{x}_i^{(p)}) \geq 1 - \xi_i, \text{and } \xi_i > 0. \tag{7}$$

The resultant RA-SVM model is $f(\mathbf{x}) = \delta f_0(\mathbf{x}) + (\sum_{i=1}^m \alpha_i y_i^{(p)} \mathbf{x}_i^{(p)})^t\mathbf{x}$, where α_i are the parameters in the dual form of the SVM modeling [Cristianini and Shawe-Taylor 2000]. In practice, the function $f_0()$ can be of any form - we use $f_0(\mathbf{x}) = \mathbf{w}_0^t\mathbf{x}$ for easier understanding in the SVM framework. Since the RA-SVM model is basically a linear model - a linear combination of the source domain function and the target domain function, we expect their performance would be lower than non-linear approaches such as the tree-based approaches. We implement the RA-SVM algorithm based on the liblinear SVM package [Fan et al. 2008].

6.4. Experimental Results for Basic Trada

The first two groups of data are used for experimenting with the basic Trada algorithm. The first group (S0 and S1) is used to study the parameter settings and the second group is used to show the effectiveness of different Trada algorithms on different datasets. All the following experimental results are based on five-fold cross validation. For example, for a dataset with 200 queries, the dataset is first equally and

Table IV. Notations for Trada adaptation settings.

notation	description
R	tuning responses layer by layer
RA	tuning aggregated responses at leaves
S	tuning splitting values
T	trimming branches that no new example reaches.
G	growing branches
C	feeding the split change back to the source domain

randomly partitioned into five subsets - each subset has 40 queries. In each fold of evaluation, four of the five subsets (160 queries) are used for training and the remaining 40 queries for testing. For clear representation, we use only NDCG5 (or DCG5) in evaluation. Each NDCG5 (DCG5) value in the figures is the average over the testing results in the five folds.

For clear presentation, we setup the notations for different settings of the Trada algorithms (Table IV).

6.4.1. Choosing the Base Model. The first problem for using the Trada algorithm is the selection of the base model. Any of the settings of the base model (e.g., the number of trees, the number of leaf nodes, and the learning rate) may affect the result of adaptation. It would be overwhelming to test all the possibilities to find the best setting. We design the following simplified experiment to show the effect of the base model.

First, we train two models with the entire S1 training datasets using the settings of (500 trees, 8 leaf nodes) and (500 trees, 12 leaf nodes), respectively. The learning rate is set to 0.05 and the sampling rate is 50% to avoid overfitting, according to the literature [Friedman 2001]. By default, we will also use these settings of the learning rate and the sampling rate for other experiments.

Then, the first n (n=100∼500) trees of a trained model is used as the base model for the adaptation to the domain S1. We increase n by 100 each time to observe the effect of the size of base model to the result of adaptation. For simplicity, we fix the setting for the adaptation algorithm (600 sample queries, the algorithm setting "R", $\beta = 10$, and 50 additional trees).

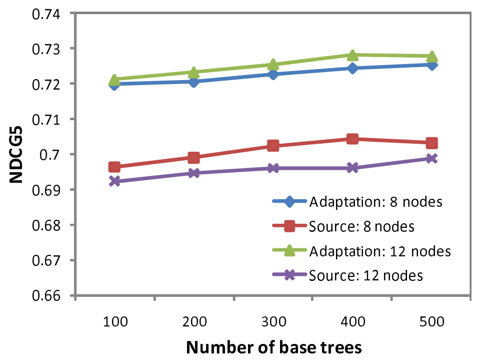


Fig. 8. finding the best base model setting.

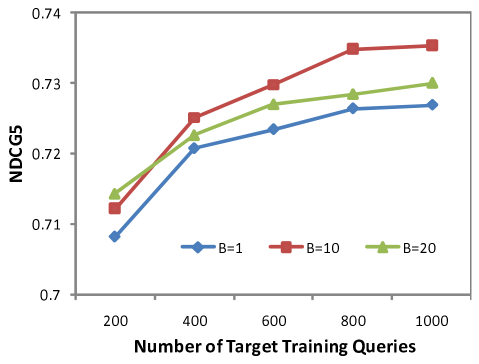


Fig. 9. β setting, training data size and adaptation performance (RS)

Figure 8 shows the effect of the base model setting to Trada adaptation. With the increasing number of base model trees, the adaptation performance is increased steady and slow, and finally reach a platform around 400 base trees. The result with 12 leaf nodes shows slightly better than that with 8 leaf nodes. We will use 400 base trees and 12 leaf nodes for the default setting of the subsequent experiments.

6.4.2. Training data size, β setting, and model performance. The size of target training data is an important factor in ranking function adaptation. It is understood that adaptation will not be necessary when the size of target training data increases to certain amount (although we do not know the exact amount). In this experiment, we experiment with Adaptation Algorithm RS, i.e., tuning both responses and splitting values to investigate the effect of both the size of the target training data and the β setting.

When performing adaptation, we do not change the structure of the base model trees learned from S0, i.e., the number of trees and the number of leaf nodes per tree do not change. Based on previous investigation, we use 400 base model trees and 12 leaf nodes. Figure 9 shows the result for different settings on Algorithm RS. With the increase of training data size, the overall performance increases as we expected. We observe significant improvement from $\beta = 1$ to $\beta = 10$. However, increasing β further from 10 to 20 will gain. Furthermore, large β overweights the small data too much and may cause overfitting. In practice, we will use a smaller β setting, if two β settings give similar performance, to reduce the chance of overfitting. As the experiment result shows, we will choose $\beta = 10$ for the subsequent experiments.

6.4.3. Adaptation with Appending Trees. The next problem is how many additional trees will be sufficient. We also show the advantages of Trada adaptation over simple "additive models". We define an additive model as a model that does not change the base model, but appends trees based on the residuals of the target training data to the base model. In Trada, the additional trees are used to adapt the new features. We use the setting of 400 base trees, 12 leaf nodes, B=10, 600 S1 queries, and the RS algorithm for the Trada modeling.

Figure 10 shows the comparison based on different number of additional trees. With the initial trees appended, the performance increases; but beyond some point, the performance will be going down. Approximately, both Trada and Additive give the best performance around +60 trees. The additive models with settings of 8 leaf nodes or 12 leaf nodes do not show much difference.

Next, we use the best setting (400 base model trees and 60 additional trees) to study the effect of training data size. Figure 11 shows that the Trada method performs always better than the additive model, and the differences at 600,800, and 1,000 queries are also statistically significant.

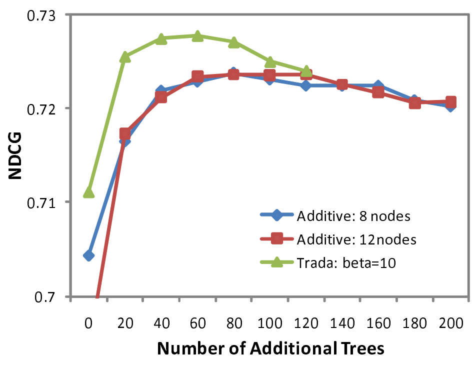


Fig. 10. Effect of increasing number of additional trees with fixed size of training data.

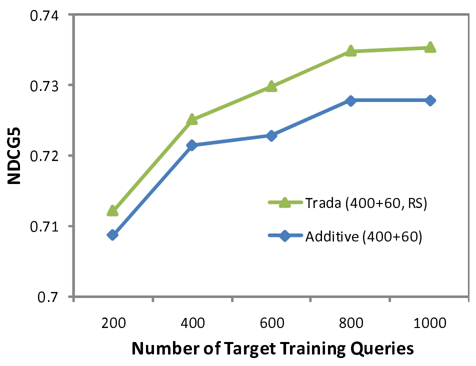


Fig. 11. Effect of additional trees with increasing training data.

6.4.4. Comparing Trada Algorithms. In this experiment, we compare several Trada algorithms: R, RA, RS and TRS. First, we want to compare the two response tuning algorithms: tuning the layer-by-layer residual fitting (R) or tuning the final leaf response (RA). Figure 12 shows that layer-by-layer response tuning is better than leaf-node only response tuning. This matches our expectation that it is better to consider a more global distribution than a local distribution.

Figure 13 the algorithm RS (tuning both responses and splitting values) has the best performance. We have not found that trimming branches (TRS) will significantly affect the performance. Trimming branches assumes that the finer partition developed on the base model will not fit the new data – a more generalized model (with less deep branches) will work better. Non-trimming trusts the structure learned from the source domain more and assumes the branches will eventually work for future data in the target domain. As the small sample set is not so representative, we expect that non-trimming will work better for future data, assuming the similarity between the source and target domains is high. Without sufficient data, trimming or not can only be determined by certain prior beliefs or heuristics, which will be a part of extended study. Growing branches (GR) also overweights the target domain data and changes the structure more aggressively. The result shows this dramatic change is not preferred in adaptation.

Although the algorithm RS shows the best performance among other algorithms for adapting S0 to S1, this is not certain for other datasets as we will show later.

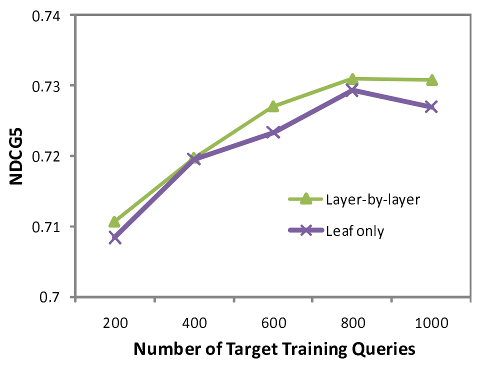


Fig. 12. Two response adaptation algorithms: layer-by-layer response tuning and aggregated response tuning at leaf nodes

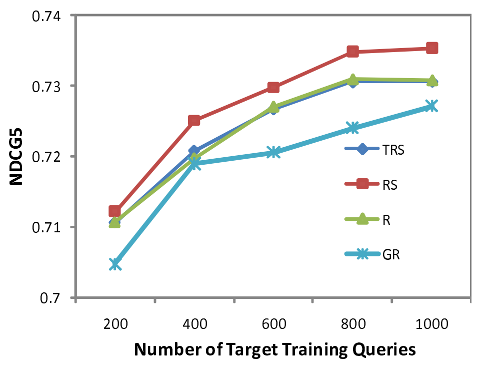


Fig. 13. Comparing different Trada algorithms.

6.4.5. Comparing with Other Simple Algorithms. Finally, we compare the adaptation approach to other common adaptation methods: 1) using only small data in the target domain; 2) combining and appropriately weighting the data from both the source and the target domains; 3) using the source domain model directly. Data combination is a widely used adaptation method. The first set of experiments is done with the S0 and S1 datasets, serving as the source and target domains, respectively. The best small data model (200 trees, 8 leaf nodes) is chosen among the settings of 100-500 trees and 8 or 12 leaf nodes. The best combination model (400 trees, 12 leaf nodes, W=10) is chosen among the settings of 100-500 trees, 8 or 12 leaf nodes, and overweighting factor (W=1,10). The best source model has 400 trees and 8 nodes.

Figure 14 shows that the Trada method has the best result, but the difference between the combination method and the Trada method is small and not statistically

Table V. Significance test.

Target Queries	200	400	600	800	1k
Trada vs. Source	x	x	x	x	x
Trada vs. Small-data	x	x	x	x	x
Trada vs. Combination					

Significance test: Trada vs. other methods on S0-S1 adaptation, with varying training data size. 'x' means the difference is statistically significant in t-test.

significant. In data combination modeling, we try two settings: simply pooling the two sets of data with equal weight (W=1) and overweighting the small data (W=10). Overweighting gives better results. As we have discussed, appropriately weighting individual samples according to the local distribution is a challenging topic in data combination, which, however, is automatically done in tree adaptation. Note that data combination may work better than Trada for some datasets in some cases, as shown in our second set of experiments with real datasets Figure 16.

The small-data model and the source domain model show much worse performance than the Trada and data combination models. However, in the previous results [Chen et al. 2008] (Figure 15, the small-data model performs comparably well in cross-validation to the adaptation models on 800∼1000 target training queries. This happens when the small target-domain data is highly biased and easy to model, so that training with the target domain only gives an overfitted model. Be aware that since the evaluation is done with the data we have and the assumption is that the target data is too small to be representative, this result is just a weak indication on the future performance. In general, we expect that the adaptation models (e.g., Trada, data combination, additive modeling) should be more robust on the future data than the small-data model.

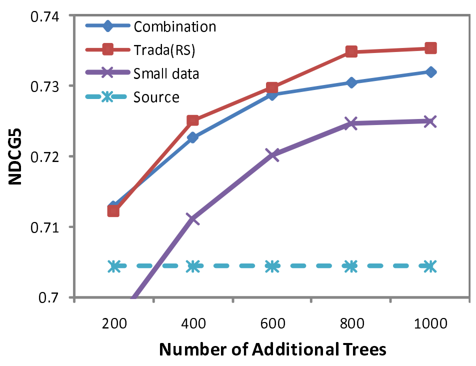


Fig. 14. Comparing tree adaptation to other methods on S0 to S1 adaptation. Trada and data combination are close and much better than small-data and source domain model.

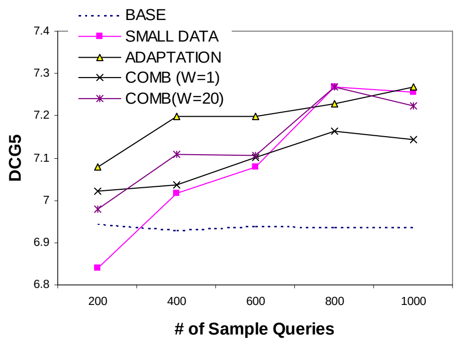


Fig. 15. Comparing tree adaptation to other methods on D0 to D1 adaptation. Trada is the best. Small-data models shows comparative performance at 800 and 1000 queries, which are subject to overfitting.

Finally, we summarize different settings of the Trada adaption algorithm and compare them to data combination models with the second group of data (D0 - D5) (Figure 16). We use D0 as the source domain and train a model using the setting of 300 trees, 12 leaf nodes, 0.05 learning rate, and 0.5 sampling rate. All adapted models used 330 trees (i.e., 30 additional trees) and the same learning and sampling rates[4]. The training data may vary according to different domains [5]. The combination models are selected among the different weight settings $W = 1, 10, 20$. The evaluation is done in five-fold cross validation. Among the five target domains, we find that adaptation is better than data combination for D_1, D_4, and D_5, while data combination is slightly better for the other two domains. Among the different adaptation settings, the first four settings "R", "TR", "SR", and "TSR" have about the same performance, while the more aggressive settings "TSRG" that includes growing branches according to the local size of the target data, and "TSRC" that feeds the splitting values back to the source domain to alter the source domain data distribution, do not work satisfactorily. Note that for D_1 and D_3 the "TSRC" results are too low to be included in comparison. This confirms our intuition that the base model should have captured the shared distributions crossing domains fairly, and dramatic structural changes guided by the small target domain data may result in worse model quality.

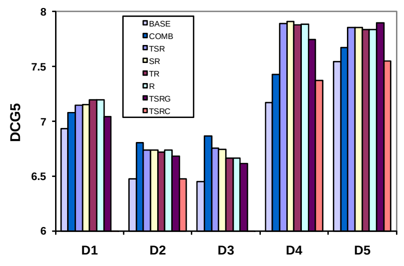


Fig. 16. Comparison with more tree adaptation options and datasets. Results are based on k-fold cross validation.

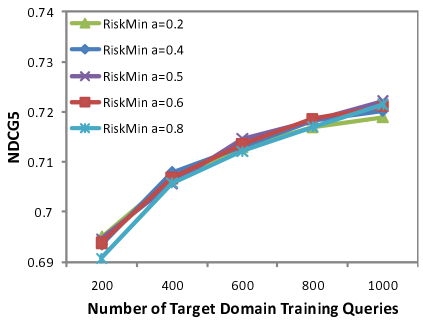


Fig. 17. Effect of parameter α to the model quality in the RiskMin algorithm.

6.4.6. Comparing with RiskMin and RA-SVM. In this section, we compare our method with these two recently developed algorithms based on the public data S0 and S1.

RiskMin. The key idea of the RiskMin method is to tune the weights of the source domain data, while keeping the target data weights unchanged, in order to minimize a joint risk function on both the source domain data and the target domain data. We fix the setting to the base learner (GBT models with 400 trees, 12 leaf nodes, and sample rate 0.5), and investigate two factors in the RiskMin framework: the global weighting factor α for the source domain data, and the number of iterations of the algorithm.

We use the GBT algorithm to train the weighted training data in each iteration. The model converges quickly around five rounds in terms of the model quality (NDCG), which is also observed by the authors in [Long et al. 2009]. Figure 17 shows that the effect of different α settings to model quality is small.

RA-SVM. We use the GBT ranking function (400 trees, 12 leaf nodes, and sample rate 0.5) trained on the source domain S0 for training the RA-SVM functions. We study the setting of two major parameters in RA-SVM modeling: the parameter C and the weight of source domain function δ. The candidate C values are (0.00001, 0.00003, 0.00005, 0.00007, 0.0001, 0.0003, 0.0005, 0.005, 0.05, 0.1, 0.5). We found large C results in worse quality and the optimal values are around 0.00001 to 0.00005. The candidate

[4]Note we used DCG in this result, which was published in [Chen et al. 2008]
[5]The numbers for D_1 are based on the models trained with 600 queries

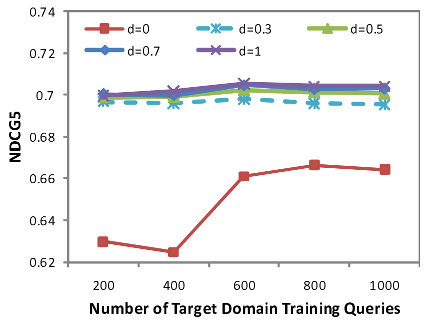


Fig. 18. The model quality of RA-SVM with different δ settings (d is the parameter δ).

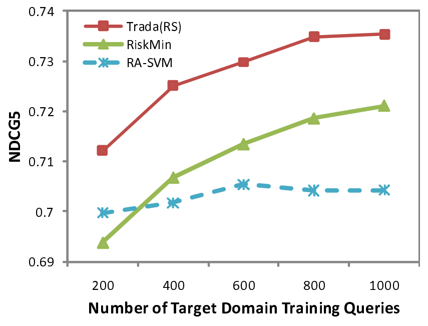


Fig. 19. Comparing our approach with RiskMin and RA-SVM.

δ values are (0, 0.1, 0.3, 0.5, 0.7, 1). When $\delta = 0$, only the target data is used for training; the influence of the source domain model increases with the increase of δ. For each size of training data and each combination of parameters of C and δ, we use five-fold cross validation to find the average model quality. Figure 18 shows the effect of δ to the model quality.

Comparison. For each size of data in the target domain, we choose the best parameter setting for RiskMin and RA-SVM, and then compare their performance with our approach. Figure 19 shows that our method is better than the compared methods. RA-SVM is worse than other two approaches possibly because it is a linear modeling method. Further studies on non-linear SVM adaptation models are needed to explore their potentials.

6.5. Experimental Results on Pairwise Trada

In this set of experiments, we examine the performance of Pairwise Trada in utilizing pairwise training data from the target domain. The base model is trained with GBRank algorithm on the pairwise data converted from the multi-grade labeled data of the source domain D_0. As it has been shown in previous study [Zheng et al. 2007] the pairwise learning algorithm can learn better models from the converted multi-grade labeled data than the original GBT models from the multi-grade labeled data. Similarly, we will study whether the pairwise data converted from the multi-grade labeled data in the target domain can also help Pairwise Trada.

We compare three methods of applying Pairwise Trada: (1) adapting the base model with only the pairwise data converted from the labeled target domain data, (2) adapting with only the pairwise data extracted from the clickthrough log (i.e., the "skip-above" and "skip-next" pairs), (3) adapting with the combination of these two sets of pairwise data. In Table VI, "BASE+Labeled", "BASE+Click" and "BASE + (Labeled+Click)" represent these three methods[6], respectively. "BASE" represents applying the source domain function directly to the target domain.

We also calculated the statistical significant for the improvements with the t-test. The bold font numbers represent they are statistically significantly better than the corresponding BASE model. The experimental results show some interesting patterns. With only the converted pairwise data from the multi-grade labeled data in the target domain, the adapted models can improve the ranking quality for most cases (except for

[6]Note that this result was initially calculated based on DCG. Since the proprietary data and results are not accessible anymore, we keep using DCG in the table.

Table VI. Pairwise Trada on different sources of pairwise training data.

	E1	E2	E3	E4	E5
BASE	6.894	7.9366	9.0159	8.5763	9.7641
BASE + Labeled	**7.1455**	**8.2811**	**9.3577**	**8.9205**	9.9531
BASE + Click	**7.1812**	**8.2640**	9.1149	8.7622	9.8818
BASE + (Labeled+Click)	**7.2414**	**8.4111**	**9.4898**	**9.0177**	**10.1156**

E5, which is not statistically significant), while with click data only the adapted model may have lower quality - although most of the testing cases are improved many are not statistically significant (i.e., for E3,E4,E5). This implies that the click pairs extracted with the current method can be used solely as the training resource but it still does not outperform expert-labeled data. Surprisingly, with the combination of click data and the labeled data, we can get better adapted models with statistical significance guarantee, which indicates the labeled data and click data are to some extent complementary. These results show the unique advantages of using the Pairwise Trada.

7. CONCLUSION

Training with insufficient data has been a major challenge for developing effective ranking functions crossing domains. Based on the observation that domains must share a certain level of similarity, we propose the tree adaptation (Trada) approach, which is based on Gradient Boosting Trees. The tree structure of the base model from the source domain provides sufficient local and global information that enables tree adaptation to conveniently incorporate the small training data from the new domain. We present a few tree adaptation algorithms and perform extensive experimental study to show the characteristics of these algorithms. The basic Trada algorithm is also extended to handle pairwise training data (Pairwise Trada), as pairwise training data can be potentially obtained with low cost from the clickthrough log. The experimental result shows that our tree adaptation approach is robust and it can improve the ranking quality with both multi-grade labeled data and pairwise preference data from multiple web search domains. The current study has been built on the intuition of localized sample weighting and model tuning. We will continue to study the theoretical underpinnings for the tree adaptation method.

REFERENCES

AGICHTEIN, E., BRILL, E., AND DUMAIS, S. 2006. Improving web search ranking by incorporating user behavior information. In *Proceedings of ACM SIGIR Conference* (Seattle, WA, USA).

ANDO, R. K. AND ZHANG, T. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research 6*, 1817–1853.

ARGYRIOU, A., EVGENIOU, T., AND PONTIL, M. 2007. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19*. MIT Press.

BACCHIANI, M. AND ROARK, B. 2003. Unsupervised language model adaptation. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

BAEZA-YATES, R. AND RIBEIRO-NETO, B. 1999. *Modern Information Retrieval*. Addison Wesley, New York City, NY.

BEN-DAVID, S., BLITZER, J., CRAMMER, K., AND SOKOLOVA, P. M. 2007. Analysis of representations for domain adaptation. In *Proceedings Of Neural Information Processing Systems (NIPS)*. MIT Press.

BLITZER, J., CRAMMER, K., KULESZA, A., PEREIRA, O., AND WORTMAN, J. 2008. Learning bounds for domain adaptation. In *Proceedings Of Neural Information Processing Systems (NIPS)*.

BLITZER, J., MCDONALD, R., AND PEREIRA, F. 2006. Domain adaptation with structural correspondence learning. In *Conference on Empirical Methods in Natural Language Processing*. Sydney, Australia.

BURGES, C., LE, Q., AND RAGNO, R. 2006. Learning to rank with nonsmooth cost functions. In *Proceedings Of Neural Information Processing Systems (NIPS)*.

BURGES, C., SHAKED, T., RENSHAW, E., LAZIER, A., DEEDS, M., HAMILTON, N., AND HULLENDER, G. 2005. Learning to rank using gradient descent. In *Proceedings of International Conference on Machine Learning (ICML)* (Bonn, Germany).

BURGES, C. J. 2010. From ranknet to lambdarank to lambdamart: An overview. In *Microsoft Research Technical Report MSR-TR-2010-82*.

CAO, Y., XU, J., LIU, T.-Y., HUANG, Y., AND HON, H.-W. 2006. Adapting ranking svm to document retrieval. In *Proceedings of ACM SIGKDD Conference* (Seattle, WA, USA).

CAO, Z., QIN, T., LIU, T.-Y., TSAI, M.-F., AND LI, H. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*. ACM, New York, NY, USA, 129–136.

CHAPELLE, O. AND ZHANG, Y. 2009. A dynamic bayesian network click model for web search ranking. In *WWW '09: Proceedings of the 18th international conference on World wide web*. ACM, New York, NY, USA, 1–10.

CHEN, K., LU, R., WONG, C., SUN, G., HECK, L., AND TSENG, B. 2008. Trada: Tree based ranking function adaptation. In *Proceedings of ACM Conference on Information and Knowledge Management (CIKM)*.

COSSOCK, D. AND ZHANG, T. 2006. Subset ranking using regression. In *ACM Conference on Learning Theory*. 605–619.

CRISTIANINI, N. AND SHAWE-TAYLOR, J. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

DAI, W., YANG, Q., XUE, G.-R., AND YU, Y. 2007. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*. ICML '07. ACM, New York, NY, USA, 193–200.

DAI, W., YANG, Q., XUE, G.-R., AND YU, Y. 2008. Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*. ICML '08. ACM, New York, NY, USA, 200–207.

DAUMÉ III, H. AND MARCU, D. 2006. Domain adaptation for statistical classifiers. *Journal of Machine Learning Research*.

DONG, A., CHANG, Y., JI, S., LIAO, C., LI, X., AND ZHENG, Z. 2009. Empirical exploitation of click data for task specific ranking. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Morristown, NJ, USA, 1086–1095.

EVGENIOU, T. AND PONTIL, M. 2004. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '04. ACM, New York, NY, USA, 109–117.

FAN, R.-E., CHANG, K.-W., HSIEH, C.-J., WANG, X.-R., AND LIN, C.-J. 2008. Liblinear: A library for large linear classification. *J. Mach. Learn. Res. 9*, 1871–1874.

FREUND, Y., IYER, R., SCHAPIRE, R. E., AND SINGER, Y. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research 4*, 933–969.

FREUND, Y. AND SCHAPIRE, R. E. 1999. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1401–1406.

FRIEDMAN, J. H. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics 29,* 5, 1189–1232.

FRIEDMAN, J. H. AND POPESCU, B. E. 2003. Importance sampled learning ensembles. *Journal of Machine Learning Research 94305*.

GAO, J., WU, Q., BURGES, C., SVORE, K., SU, Y., KHAN, N., SHAH, S., AND ZHOU, H. 2009. Model adaptation via model interpolation and boosting for Web search ranking. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 505–513.

GENG, B., YANG, L., XU, C., AND HUA, X.-S. 2009. Ranking model adaptation for domain-specific search. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*. ACM, New York, NY, USA, 197–206.

GEURTS, P. 2010. Learning to rank with extremely randomized regression trees. In *Yahoo! Learning to Rank Challenge Workshop in ICML*.

GULIN, A. AND KURALENOK, I. 2010. Yetirank: Everybody lies. In *Yahoo! Learning to Rank Challenge Workshop in ICML*.

HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. 2001. *The Elements of Statistical Learning*. Springer-Verlag.

HERBRICH, R., GRAEPEL, T., AND OBERMAYER, K. 2000. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, 115–132.

HWA, R. 1999. Supervised grammar induction using training data with limited constituent information. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

JARVELIN, K. AND KEKALAINEN, J. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of ACM SIGIR Conference*.

JI, S., ZHOU, K., LIAO, C., ZHENG, Z., XUE, G.-R., CHAPELLE, O., SUN, G., AND ZHA, H. 2009. Global ranking by exploiting user clicks. In *SIGIR*. 35–42.

JIANG, J. AND ZHAI, C. 2007. Instance weighting for domain adaptation in NLP. In *Conference of the Association for Computational Linguistics (ACL)*.

JOACHIMS, T. 2002. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD Conference*.

JOACHIMS, T., GRANKA, L., PAN, B., AND GAY, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of ACM SIGIR Conference*.

LEGGETTER, C. AND WOODLAND, P. 1995. Flexible speaker adaptation using maximum likelihood linear regression. In *Proceedings of Eurospeech*.

LEHMANN, E. L. AND CASELLA, G. 1998. *Theory of Point Estimation*. Springer-Verlag.

LI, P., BURGES, C. J., AND WU, Q. 2007. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Proceedings Of Neural Information Processing Systems (NIPS)*.

LIAO, X., XUE, Y., AND CARIN, L. 2005. Logistic regression with an auxiliary data source. In *Proceedings of the 22nd international conference on Machine learning*. ICML '05. ACM, New York, NY, USA, 505–512.

LONG, B., LAMKHEDE, S., VADREVU, S., ZHANG, Y., AND TSENG, B. L. 2009. A risk minimization framework for domain adaptation. In *CIKM*. 1347–1356.

MANSOUR, Y., MOHRI, M., AND ROSTAMIZADEH, A. 2009. Domain adaptation: Learning bounds and algorithms. In *Proceedings Of Neural Information Processing Systems (NIPS)*.

MOHAN, A., CHEN, Z., AND WEINBERGER, K. 2010. Tree ensemble and transfer learning. In *Yahoo! Learning to Rank Challenge Workshop in ICML*.

NALLAPATI, R. 2004. Discriminative models for information retrieval. In *Proceedings of ACM SIGIR Conference*. 64–71.

PAN, S. J. AND YANG, Q. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22,* 10, 1345–1359.

PAVLOV, D. AND BRUNK, C. 2010. Bagboo: Bagging the gradient boosting. In *Yahoo! Learning to Rank Challenge Workshop in ICML*.

RAINA, R., BATTLE, A., LEE, H., PACKER, B., AND NG, A. Y. 2007. Self-taught learning: transfer learning from unlabeled data. In *ICML '07: Proceedings of the 24th international conference on Machine learning*. ACM, New York, NY, USA, 759–766.

SCHAPIRE, R. E. 2003. The boosting approach to machine learning: An overview. *Nonlinear Estimation and Classification"*.

SOROKINA, D. 2010. Application of additive groves to the learning to rank challenge. In *Yahoo! Learning to Rank Challenge Workshop in ICML*.

TSAI, M.-F., LIU, T.-Y., QIN, T., CHEN, H.-H., AND MA, W.-Y. 2007. Frank: a ranking method with fidelity loss. In *Proceedings of ACM SIGIR Conference*. ACM, New York, NY, USA, 383–390.

VAPNIK, V. N. 1999. *The Nature of Statistical Learning Theory*. Springer Science and Bussiness Media, LLC.

WANG, Z., SONG, Y., AND ZHANG, C. 2008. Transferred dimensionality reduction. In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*. ECML PKDD '08. Springer-Verlag, Berlin, Heidelberg, 550–565.

WU, P. AND DIETTERICH, T. G. 2004. Improving svm accuracy by training on auxiliary data sources. In *Proceedings of International Conference on Machine Learning (ICML)*. 871–878.

WU, Q., BURGES, C. J., SVORE, K., AND GAO, J. 2008. Ranking, boosting, and model adaptation. *Microsoft Research Technical Report*.

WU, Q., BURGES, C. J. C., SVORE, K. M., AND GAO, J. 2010. Adapting boosting for information retrieval measures. *Inf. Retr. 13,* 3, 254–270.

XU, J. AND LI, H. 2007. AdaRank: a boosting algorithm for information retrieval. In *Proceedings of ACM SIGIR Conference*.

ZHENG, Z., CHEN, K., SUN, G., AND ZHA, H. 2007. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR*. 287–294.

ZHENG, Z., ZHA, H., ZHANG, T., CHAPELLE, O., CHEN, K., AND SUN, G. 2007. A general boosting method and its application to learning ranking functions for web search. In *Proceedings Of Neural Information Processing Systems (NIPS)*.

Received June 2010; revised January 2011; accepted July 2011