Trada: Tree Based Ranking Function Adaptation

Keke Chen Rongqing Lu CK Wong Gordon Sun Larry Heck Belle Tseng
Yahoo! Labs
2821 Mission College Boulevard
Santa Clara, CA 95054, USA
{kchen, rlu, ckwong, gzsun, larryh, belle}@yahoo-inc.com

ABSTRACT

Machine Learned Ranking approaches 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 training data, which has significantly limited the fast development and deployment of machine learned ranking functions for different web search domains. In this paper, we propose a new approach called tree based ranking function adaptation ("tree adaptation") to address this problem. Tree adaptation assumes that ranking functions are trained with regression-tree based modeling methods, such as Gradient Boosting 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 weight training examples considering both local and global distributions. Experiments are performed to show that tree adaptation can provide better-quality ranking functions for a new domain, compared to other modeling methods.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval Models; I.2.6 [**Learning**]: Concept Learning

General Terms

Algorithms

Keywords

Web Search Ranking, Learning to Rank, Model Adaptation, Regression Tree

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 [23, 7, 9, 21, 22, 16, 5] to a set of labeled relevance examples to learn a ranking function. Compared to the traditional ranking functions [2] developed in the information retrieval community, learning to rank has several unique benefits: (1) it is convenient to incorporate new features to the ranking 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 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 [16, 17], the expert-labeled data are still regarded as a more reliable source for training high-quality ranking functions. Due to the increasing demands from different web search domains, e.g., different regions or countries or topics, it has been urgent 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 sufficient amount of *labeled* training data.

One approach to addressing this problem is to utilize 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 difference in each feature will aggregate to significant difference in the multidimensional feature space.

Although each search domain has its own characteristics, we observe that many of them share 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 crossing different domains. We name the domain having sufficient training data as *the source domain*, and that we do not have or have only small amount of training data as *the target domain*. How to adapt a good ranking function from the source do-

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.
Copyright 2008 ACM 978-1-59593-991-3/08/10 ...$5.00.

main to a target domain and get a better ranking function, is the major problem we are going to tackle.

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 models, we will use ranking functions trained with the gradient boosting trees (GBT) method [10] 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 also explore the structural benefit of regression tree over other models, since the tree structure is highly tunable. 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, 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.

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 how tree adaptation works and then present a few tree adaptation algorithms. Experimental results will be reported in Section 5 to validate the proposed algorithms and evaluate different algorithmic settings.

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 [20] or ordinal regression problem [12, 6, 10, 23]. This category has known weaknesses since ranking cares about only the relative ordering between any pair of documents regarding to a specific query, rather than accurate prediction of categories. Therefore, the second category of algorithms proposes to take pairwise data as training data and develops pairwise ranking function. The representatively algorithms includes Ranking SVM[16], RankNet[5], RankBoost[9], and GBRank[23], 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. The typical algorithms are LamdaRank[4], ListNet[7], and AdaRank[22].

Recently, model adaptation has been of great interest in other areas, in particular, natural language processing and speech recognition, for the similar training data problems we have encountered in learning to rank. The standard 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 [1], parsing [13] and tagging [3]. In speech recognition, the maximum likelihood linear regression (MLLR) approach is also proposed for speaker adaptation [18]. The problem of distributional difference between the source domain and the target domain is formally addressed by the paper [8], which is further decomposed as 1) the difference between the prior distribution of feature vectors and 2) the difference between the label distributions [15]. The paper [15] also used a simple data combination technique by appropriately overweighting the target domain data in training. As we will show, 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 importance. In contrast, our tree adaptation technique can automatically adapt to the fine-grained distribution difference between the source domain and the target domain.

3. PRELIMINARY

Tree adaptation follows the general GBT training framework, while the major algorithms are more related to the mechanism of generating regression tree. 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

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.

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. Totally 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, and 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 ordinal regression. Many algorithms can do this purpose and 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 [11]. 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.

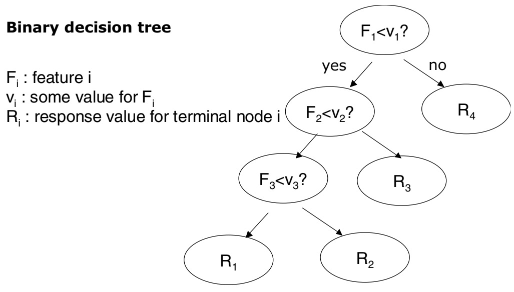

Figure 1: A sample regression tree

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 the 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. As we have known, the best-effort predictor for all records falling onto the current node is the mean of all r_{ij}, i.e., $\hat{r}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} r_{ij}$ [11]. 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 to the number of features. In the sequel, 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 $F_1 < a \wedge F_2 < b_0$.

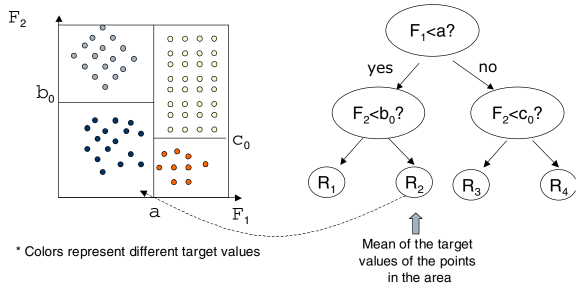
F_2
b_0
c_0
a
F_1
* Colors represent different target values
$F_1<a$?
yes
no
$F_2<b_0$?
$F_2<c_0$?
R_1
R_2
R_3
R_4
Mean of the target values of the points in the area

Figure 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

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

Note that 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 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 [10]. The GBT learning method trains the k-th tree, based on the previous trees h_j, $1 \le j < k$, 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 $k > 1$, $1 \le j < k$, $r_i = y_i - \sum_{j=1}^{k-1} \gamma_j h_j(\mathbf{x}_i)$.
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 GBT

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

4.1 Problem of Domain Adaptation

In what scenarios will domain adaptation work? To answer this question, we need to formally analyze the learning problem. 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 joint distribution $p(\mathbf{x}, y)$. $p(\mathbf{x}, y)$ can be decomposed into two parts, as $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$. $p(\mathbf{x})$ is the underlying unlabeled data distribution and $p(y|\mathbf{x})$ is the label distribution. Let (s) denote the source domain and (t) the target domain. In general, the source and target domains differ in both $p(y|\mathbf{x})$ and $p(\mathbf{x})$. Model adaptation works in the following scenarios.

- Apparently, when $p^{(s)}(\mathbf{x})$ and $p^{(t)}(\mathbf{x})$ are orthogonally distributed, the source domain data will not help in training the target domain function. $p^{(s)}(\mathbf{x})$ and $p^{(t)}(\mathbf{x})$ have to be significantly correlated to some extent so that domain adaptation is possible. In the ranking problem, the relevant training examples are far less than the irrelevant examples in the overall true distribution $p(\mathbf{x})$. Therefore, in constructing the training data, some biases have already existed in collecting 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 distribution (Figure 3).

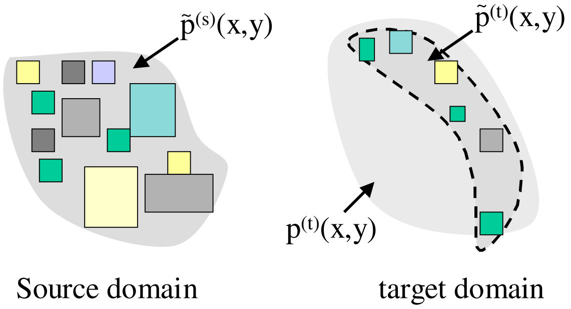

Figure 3: Source domain data may be able to patch the missing part of target domain.

- 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 significant amount of noisy labels in the target domain. In this case, the source domain data may help to correct the bias from the noisy labels.

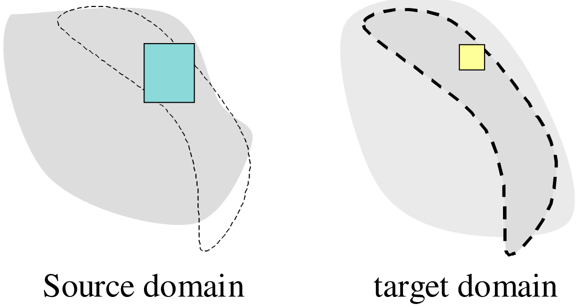

Figure 4: In overlapped part of distribution, the label distribution may still differ between the source and the target.

Figure 3 and 4 illustrate the above scenarios. The small blocks in the figures represent the local areas in the high-dimensional space ("the high-dimensional bounding boxes") that are covered by the training data. Different colors represent different target values for the blocks. In regression tree modeling, each leaf node tries to approximately model one of these blocks, and each internal node groups a few nearby blocks with close target values.

The above illustration also shows that training with only the small amount of target training data may cause serious overfitting. A feasible solution could be combining the two sets of data or even two models. However, simply combining the datasets may not work well. The major difficulty is to appropriately weight the samples from the target domain: without sufficient understanding of the source data and the target domain, often, we can only roughly handle the weighting scheme. When inappropriately overweighting the small data from the target domain, we will get unsatisfactory models.

Tree adaptation provides us 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 tend to be more robust and less possible to overfit the small data.

4.2 Tree Adaptation Algorithms

The basic idea of tree adaptation includes 1) using the base model to partition the new dataset, i.e., approximating $\tilde{p}^{(t)}(\mathbf{x})$ with $\tilde{p}^{(s)}(\mathbf{x})$; 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 $\tilde{p}(\mathbf{x})$. In a regression tree, each path from root to any node represents a multidimensional bounding box, which is a subspace of $\tilde{p}(\mathbf{x})$. 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 $\tilde{p}^{(t)}(\mathbf{x})$ based on the source data distribution $\tilde{p}^{(s)}(\mathbf{x})$.

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 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 distribution that the two sets of points fall on this node follow a binomial distribution, and the corresponding probabilities are p_0 and $(1 - p_0)$ for the source and target data, respectively. A balanced estimate of the combined value is calculated by

$$f(R_0, R_1, p_0) = p_0 \times R_0 + (1 - p_0) \times R_1 \qquad (2)$$

$f(R_0, R_1, p_0)$ 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

$$\hat{p}_0 = \frac{n_0}{n_0 + \beta \times n_1} \qquad (3)$$

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.

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

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. It 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 5 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.

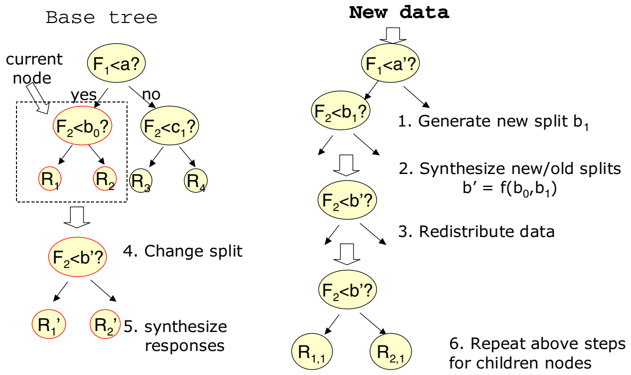

Figure 5: A sample algorithm for tree adaptation

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: Additional Trees to Incorporate New Features

Since the base model is trained for the source domain, which may not contain the features 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 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 overfitting 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 is not 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. For significantly different distributions, it is ineffective to catch up the difference with appended trees. We will show that additive modeling has limited advantage if not combined with Algorithm 1 or 2.

5. EXPERIMENTS

There are several goals in this experimental evaluation. First, we want to see how the setting of β 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; Third, additional trees (the additive method) can further benefit adaptation and we study how much additional gain we can get; Finally, the adaptation approach is compared to other existing 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.

5.1 Datasets

The data used for training the base model from the source domain D_0 is large, including around 150,000 (query, web document, grade) examples for about 6,000 sample web queries. The base model uses 200 $\sim$ 300 features scattered in the three categories we have described in Section 3.

The target domain D_1 has about 38,000 examples from around 1,300 queries. This is not a typical small dataset. We will use samples from the target domain to simulate the scenario that we have only very small training data. These training examples are labeled by relevance experts and can be divided into 5 batches according to the time the labeling was done. A five-fold cross validation will use this natural split.

There are a few more datasets that come from the target domains D_2 - D_5, respectively. We will focus on D_1 for detailed study on the properties of tree adaptation, while using other domain data for comparing different modeling methods. Table 1 summarizes the size of the datasets.

	query	document
D_0	6012	146307
D_1	1282	37952
D_2	823	12092
D_3	804	34276
D_4	540	16950
D_5	1258	29165

Table 1: Size of domain training data.

5.2 Evaluation Metrics

Discounted Cumulative Gain (DCG) [14] 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, which are mapped to integers {'10', '7', '3', '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.

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

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.

Each model test will generate a list of DCGs corresponding to the list of testing queries. To compare the statistical significance between two results, we perform t-test [19] on the two DCG lists. If p-value < 0.05, the results are significantly different. t-test is only performed on the comparison between different models. For parameter tuning of the same model, we will show only the average DCG of the multi-fold cross validation. Note that the relevance differences between high-quality commercial web search engines are often less than 5%. Therefore, the small *statistically significant* improvements will have practical impact.

5.3 Modeling Methods for Comparison

We are going to compare four types of modeling methods to the proposed adaptation approach. They are 1) the base model only, 2) the small-data model, 3) the additive model, and 4) the data combination model. We briefly describe how they are generated as follows.

The base model is trained with a large training dataset from the source domain. In experiments, we train a GBT model on the D_0 data with parameters: 300 trees, 12 leaf nodes, and 0.05 shrinkage rate. Testing results show this model works reasonably well for all target domains. We will use the ranking quality of this 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 [15] 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.

There are also a few settings for performing adaptation as we have described. For clear presentation, we setup the notations for these settings (Table 2).

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.

Table 2: Notations for adaptation settings.

5.4 Experiment Setup

The target training dataset consists of batches of data, which was judged by experts during different periods. The cross validation is done based on the batches, i.e., we directly map the batches to the folds in cross validation. Depending on the number of batches, different domains (D_1 -D_5) may have different number of folds. In each fold of cross validation, one batch is used for testing and the rest are used for training. Meanwhile, when we test the effect of training data size to the performance of adaptation with D_1, we also sample specific amount of queries from the training folds in each round. When training adaptation models, the target domains use the same base model that is trained on domain D_0 (300 trees and 12 leaf nodes).

5.5 Experimental Results

5.5.1 *Training data size, β setting, and model performance*

The size of target training data is an important factor in ranking function adaptation. We anticipate that we will not need adaptation when the size of target training data increases to certain amount. In the first set of experiments, we use Adaptation Algorithm 2, i.e., tuning both responses and splitting values to investigate the effect of training data size and β setting on adaptation.

When performing adaptation, we do not change the structure of the base model trees, i.e., the number of trees and the number of leaf nodes per tree are not change. Figure 6 shows the result for different settings on Algorithm 2. With the increase of training data size, the overall performance increases as we expected. With larger β the performance tends to better as well. However, increasing β from 10 to 20 will not change the performance much. In fact, large β may be subject to over-tuning the local distributions, and thus overfitting the small data. In practice, we will use a smaller β setting if the two settings give similar performance, to reduce the chance of overfitting.

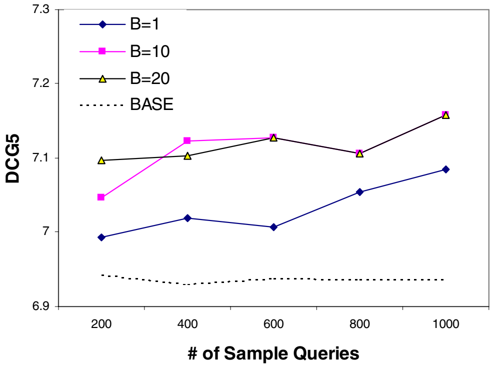

Figure 6: β setting, training data size and model SR performance

5.5.2 *Comparing Adaptation Algorithms*

First, we want to compare the two response tuning algorithms: tuning the layer-by-layer residual-fitting values or tuning the final leaf response. Figure 7 shows that layer-by-layer tuning is much better than aggregated-response tuning. This matches our expectation that it is good to consider more global distribution.

However, we have not found that trimming branches 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.

In addition, the algorithm 2, tuning both splitting values and responses, actually reduces the performance slightly for D_1 data. However, this is not sure for all cases as we will show later.

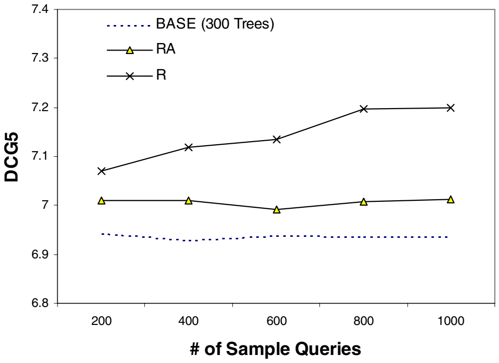

Figure 7: Two response adaptation algorithms

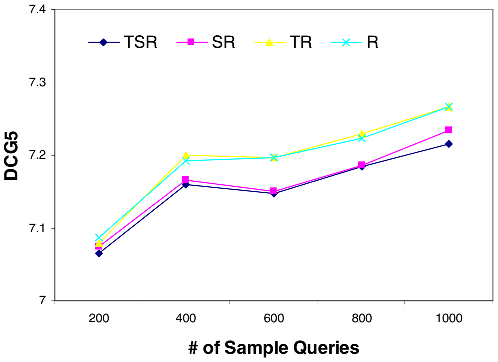

Figure 8: Effect of trimming branches and tuning both splitting values and responses.

5.5.3 Adaptation with Additional Trees

When testing the effect of additional trees, we first fix the number of additional trees (30 trees) to see the relationship between the size of training data and the performance of different models. Adapted models are compared to "additive models" that append trees to the base model without changing the base model trees. Figure 9 shows the comparison. Additional trees improve the quality of the base model to certain extent. Adaptation plus additional trees give more gains and the resultant models are much better than the simple additive models. t-test shows that the performance difference between the additive models and the adapted models is statistically significant (p-value< 0.05)

Next, let's keep appending more trees with fixed training data size. Figure 10 shows that with the same training data size (600 queries), appending more trees will slightly increase the performance for both adapted models and additive models. However, the performance differences keep almost constant and they are also statistically significant.

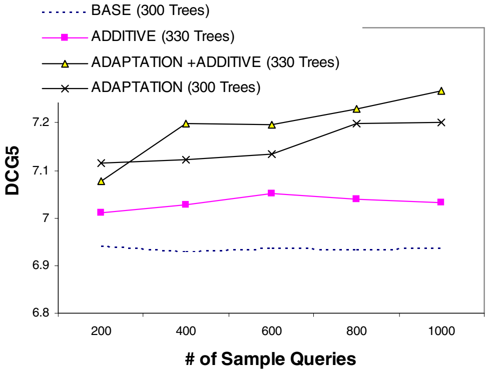

Figure 9: Effect of additional trees with increasing training data.

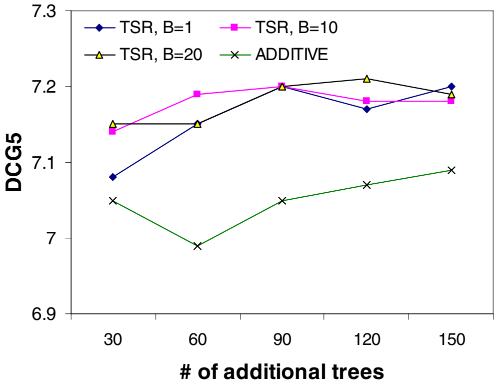

Figure 10: Effect of increasing number of additional trees with fixed size of training data.

5.5.4 Comparing with other methods

Finally, we compare the adaptation approach to other methods: 1) small-data modeling; 2) data combination modeling; 3) the base model. For fair comparison, except the base model, all models have the same size: 330 trees and 12 leaf nodes per tree. The performance of adapted model is the best among all algorithms and settings. In particular, we indeed observed that when the training data is very small, e.g., $\leq$ 600 queries, adaptation models clearly show advantages. We find that with the increase of training data, the performance of small-data model increases quickly and pars with the adapted models around the size of 1000 queries, in Figure 11. However, for training a good ranking function, 1000 queries are often insufficient. It is highly possible that the small-data model with 1000 queries is still overfitting the incompletely distributed training data, as we illustrated in Figure 3. Further testing with future data will show the bias. Since adapted models consider both the training data for the base model and those from the new domain, in general, we consider they should have more generalization power than the small-data models.

	200	400	600	800	1k
ADP vs. BASE		x	x	x	x
ADP vs. SMALL	x	x			
ADP vs. COMB					

Table 3: significant test: adaptation vs. other methods with varying training data size. 'x' means statistically significant in t-test.

In data combination modeling, we try two settings: simply pooling the two sets of data with equal weight and overweighting the small data. Note that overweighting the small data too much may result in a model very similar to a small-data model. As Figure 12 shows, by weighting 20 times (W=20), the curve of combination model is actually very close to that of small-data model, with some exception at extremely small data (200 queries). With simple pooling (W=1), the combination models perform worse than the adaptation models. As we have analyzed, appropriately weighting individual samples according to the local distribution is a challenging topic in data combination, which, however, is naturally addressed by the tree adaptation algorithms.

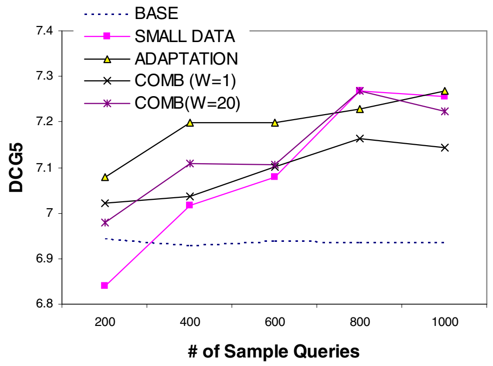

Figure 11: Comparing tree adaptation to other methods

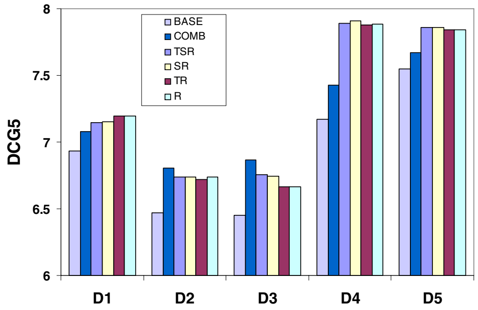

Figure 12: Comparing tree adaptation to other methods

Finally, we summarize the settings of adaption algorithm and compare them to data combination models for a few more target domains (Figure 12). All models expect the base model have 330 trees and 12 leaf nodes. However, the training data may vary according to different domains [1]. The combination models are selected among the different weight settings $W = 1, 10, 20$. Among the five 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.

6. 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 some similarity, we propose the tree adaptation 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 result shows that the tree adaptation approach is robust and usually better than other methods.

7. REFERENCES

[1] Bacchiani, M., and Roark, B. Unsupervised language model adaptation. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2003).

[2] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval*. Addison Wesley, New York City, NY, 1999.

[3] Blitzer, J., McDonald, R., and Pereira, F. Domain adaptation with structural correspondence learning. In *Conference on Empirical Methods in Natural Language Processing* (Sydney, Australia, 2006).

[4] Burges, C., Le, Q., and Ragno, R. Learning to rank with nonsmooth cost functions. In *Proceedings Of Neural Information Processing Systems (NIPS)* (2006).

[5] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of International Conference on Machine Learning (ICML)* (2005).

[6] Cao, Y., Xu, J., Liu, T.-Y., Huang, Y., and Hon, H.-W. Adapting ranking svm to document retrieval. In *Proceedings of ACM SIGKDD Conference* (2006).

[7] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning* (New York, NY, USA, 2007), ACM, pp. 129–136.

[8] Daumé III, H., and Marcu, D. Domain adaptation for statistical classifiers. *Journal of Machine Learning Research* (2006).

[9] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research 4* (2003), 933–969.

[1] The numbers for D_1 are from previous 600-query results

[10] Friedman, J. H. Greedy function approximation: A gradient boosting machine. *Annals of Statistics 29*, 5 (2001), 1189–1232.

[11] Hastie, T., Tibshirani, R., and Friedmann, J. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[12] Herbrich, R., Graepel, T., and Obermayer, K. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers* (2000), 115–132.

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

[14] Jarvelin, K., and Kekalainen, J. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of ACM SIGIR Conference* (2000).

[15] Jiang, J., and Zhai, C. Instance weighting for domain adaptation in NLP. In *Conference of the Association for Computational Linguistics (ACL)* (2007).

[16] Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD Conference* (2002).

[17] Joachims, T., Granka, L., Pan, B., and Gay, G. Accurately interpreting clickthough data as implicit feedback. In *Proceedings of ACM SIGIR Conference* (2005).

[18] Leggetter, C., and Woodland, P. Flexible speaker adaptation using maximum likelihood linear regression. In *Proceedings of Eurospeech* (1995).

[19] Lehmann, E. L., and Casella, G. *Theory of Point Estimation*. Springer-Verlag, 1998.

[20] Nallapati, R. Discriminative models for information retrieval. In *Proceedings of ACM SIGIR Conference* (2004), pp. 64–71.

[21] Tsai, M.-F., Liu, T.-Y., Qin, T., Chen, H.-H., and Ma, W.-Y. Frank: a ranking method with fidelity loss. In *Proceedings of ACM SIGIR Conference* (New York, NY, USA, 2007), ACM, pp. 383–390.

[22] Xu, J., and Li, H. AdaRank: a boosting algorithm for information retrieval. In *Proceedings of ACM SIGIR Conference* (2007).

[23] Zheng, Z., Chen, K., Sun, G., and Zha, H. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR* (2007), pp. 287–294.