In *Proceedings of the Twenty-Seventh International Conference on Machine Learning* (ICML 10), Haifa, Israel, June 2010.

Boosting for Regression Transfer

David Pardoe and Peter Stone {DPARDOE, PSTONE}@CS.UTEXAS.EDU
The University of Texas at Austin, 1 University Station C0500, Austin, TX 78712 USA

Abstract

The goal of transfer learning is to improve the learning of a new *target* concept given knowledge of related *source* concept(s). We introduce the first boosting-based algorithms for transfer learning that apply to regression tasks. First, we describe two existing classification transfer algorithms, ExpBoost and TrAdaBoost, and show how they can be modified for regression. We then introduce extensions of these algorithms that improve performance significantly on controlled experiments in a wide range of test domains.

1. Introduction

The idea behind transfer learning (Pan & Yang, 2009) is that it is easier to learn a new concept (such as how to play the trombone) if you are already familiar with a similar concept (such as playing the trumpet). In the context of supervised learning, inductive transfer learning is often framed as the problem of learning a concept of interest, called the *target concept*, given data from multiple sources: a typically small amount of *target data* that reflects the target concept, and a larger amount of *source data* that reflects one or more different, but possibly related, *source concepts*. A number of algorithms have been developed to address this situation in classification settings, but much less attention has been paid to regression settings.

One general approach that has been applied successfully to classification transfer is boosting. In this paper, we introduce and evaluate the first boosting-based algorithms for regression transfer. These algorithms can be divided into two categories: algorithms that make use of models trained on the source data, and algorithms that use the source data directly. We first describe an existing classification transfer algorithm from each category (ExpBoost (Rettinger et al., 2006) and TrAdaBoost (Dai et al., 2007), respectively), and we show how these algorithms can be modified for a regression setting. Next, we present the primary contribution of this paper: two new algorithms designed to overcome shortcomings observed in these modified algorithms. Finally, we present experimental results for all algorithms in seven test domains.

Appearing in *Proceedings of the 27^{th} International Conference on Machine Learning*, Haifa, Israel, 2010.

2. Regression Transfer

In this section, we specify our learning problem and outline two approaches to solving this problem. Then we provide necessary background on boosting for regression problems.

2.1. Problem Specification

Our goal is to learn a model of a concept c_{target} mapping feature vectors from the space X to labels in the space Y. In binary classification problems, $Y = \{0, 1\}$, while in the regression problems studied here, $Y = \mathbb{R}$. We are given a set of training instances $T_{target} = \{(x_i, y_i)\}$, with $x_i \in X$ and $y_i \in Y$ for $1 \leq i \leq n$, that reflect c_{target}. In addition, we are given data sets $T^1_{source}, \ldots, T^B_{source}$ reflecting B different, but possibly related, concepts also mapping X to Y. In order to learn the most accurate possible model of c_{target}, we must decide how to use both the target and source data sets. If T_{target} is sufficiently large, we can likely learn a good model using only this data. However, if T_{target} is small and one or more of the source concepts is similar to c_{target}, then we may be able to use the source data to improve our model.

2.2. ExpBoost and TrAdaBoost

In this paper, we will consider regression transfer algorithms that fit into two categories: those that make use of models trained on the source data, and those that use the source data directly as training data. The algorithms we will present in these two categories are inspired by two boosting-based algorithms for classification transfer, ExpBoost (Rettinger et al., 2006) and TrAdaBoost (Dai et al., 2007). Boosting is an ensemble method in which a sequence of models (or hypotheses) $h_1 \ldots h_N$, each mapping from X to Y, are iteratively fit to some transformation of a data set using a base learner. The outputs of these models are then combined into a final hypothesis h_f.

In ExpBoost, a separate hypothesis (or expert, hence the name) h^i is learned for each of the B source data sets, and learning is performed using only T_{target}. At each step of the boosting process, ExpBoost chooses to use either the hypothesis h_t learned from the weighted training data or one of the experts, depending on which is most accurate.

In contrast, TrAdaBoost uses the source data sets directly by combining them with T_{target} to form a single data set. At each boosting step, TrAdaBoost increases the relative weights of target instances that are misclassified. When a source instance is misclassified, however, its weight is decreased. In this way, TrAdaBoost aims to identify and make use of those source instances that are most similar to the target data while ignoring those that are dissimilar.

We provide additional details on these algorithms and their extensions below, but first we address the issue of applying boosting algorithms to regression problems.

2.3. AdaBoost and Regression

One of the best known boosting methods for classification, and the one upon which ExpBoost and TrAdaBoost are based, is AdaBoost (specifically, AdaBoost.M1) (Freund & Schapire, 1997). In AdaBoost, each training instance receives a weight w_i that is used when learning each hypothesis; this weight indicates the relative importance of each instance and is used in computing the error of a hypothesis on the data set. After each iteration, instances are reweighted, with those instances that are not correctly classified by the last hypothesis receiving larger weights (as in step 5 of Algorithm 1). Thus, as the process continues, learning focuses on those instances that are most difficult to classify.

A number of methods have been proposed for modifying AdaBoost for regression, and as TrAdaBoost and ExpBoost are based on AdaBoost, these modifications can be used on them as well. In our work, we explored two of these methods that have been shown to be generally effective and that can be applied to TrAdaBoost and ExpBoost in a straightforward way: AdaBoost.R2 and AdaBoost.RT.

The key to AdaBoost is the reweighting of those instances that are misclassified at each iteration. In regression problems, the output given by a hypothesis h_t for an instance x_i is not correct or incorrect, but has a real-valued error $e_i = |y_i - h_t(x_i)|$ that may be arbitrarily large. Thus, we need a method of mapping an error e_i into an adjusted error e'_i that can be used in the reweighting formula used by AdaBoost.

Algorithm 1 `AdaBoost.R2` (Drucker, 1997)

Input the labeled target data set T of size n, the maximum number of iterations N, and a base learning algorithm *Learner*. Unless otherwise specified, set the initial weight vector $\mathbf{w}^1$ such that $w_i^1 = 1/n$ for $1 \le i \le n$.

For $t = 1, \ldots, N$:

1. Call *Learner* with the training set T and the distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
2. Calculate the adjusted error e_i^t for each instance:
   let $D_t = max_{j=1}^{n} |y_j - h_t(x_j)|$
   then $e_i^t = |y_i - h_t(x_i)|/D_t$
3. Calculate the adjusted error of h_t:
   $\epsilon_t = \sum_{i=1}^{n} e_i^t w_i^t$; if $\epsilon_t \ge 0.5$, stop and set $N = t - 1$.
4. Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
5. Update the weight vector:
   $w_i^{t+1} = w_i^t \beta_t^{1-e_i^t}/Z_t$ (Z_t is a normalizing constant)

Output the hypothesis:
$h_f(x)$ = the weighted median of $h_t(x)$ for $1 \le t \le N$, using $\ln(1/\beta_t)$ as the weight for hypothesis h_t.

The method used in AdaBoost.R2 (Drucker, 1997) is to express each error in relation to the largest error $D = max_{i=0}^{n} |e_i|$ in such a way that each adjusted error e'_i is in the range $[0, 1]$. In particular, one of three possible loss functions is used: $e'_i = e_i/D$ (linear), $e'_i = {e_i}^2/D^2$ (square), or $e'_i = 1 - exp(-e_i/D)$ (exponential). The degree to which instance x_i is reweighted in iteration t thus depends on how large the error of h_t is on x_i relative to the error on the worst instance. AdaBoost.RT (Shrestha & Solomatine, 2006), on the other hand, continues to label each output as correct ($e'_i = 0$) or incorrect ($e'_i = 1$) using an error threshold ϕ. That is, if $e_i > \phi$, then $e'_i = 1$; otherwise, $e'_i = 0$.

In preliminary experiments, we found AdaBoost.R2 with the linear loss function to work consistently well, and were unable to find values of ϕ that allowed AdaBoost.RT to regularly match this performance. In the remainder of this paper we consider only AdaBoost.R2 with the linear loss function, shown in Algorithm 1.

3. Using Source Models

In this section we describe four regression transfer algorithms based on making use of source models. In addition to the target data, each algorithm receives as input a set of experts $H^B = \{h^1, \ldots, h^B\}$, each corresponding to a source data set.

3.1. ExpBoost.R2

Combining the principles of AdaBoost.R2 with those of ExpBoost results in the new regression algorithm ExpBoost.R2. The steps involving computing the ad-

Algorithm 2 `Transfer Stacking`

Input a labeled data set $T = \{(x_i, y_i)\}$ of size n, a set of experts $H^B = \{h^1, \ldots, h^B\}$, the number of folds F for cross validation, and a base learning algorithm *Learner*.

1. Let $O_{i,j} = h^j(x_i)$ for $1 \le i \le n$ and $1 \le j \le B$.
2. Perform F-fold cross validation on T using *Learner*. For $1 \le i \le n$, let $O_{i,B+1}$ equal the output of the learned model for the fold where instance i is in the validation set.
3. Call *Learner* with the full training set T and get hypothesis h^{B+1}.
4. Perform linear least squares regression on the system of equations $(\sum_{j=1}^{B+1} a_j O_{i,j}) + a_{B+2} = y_i$ for $1 \le i \le n$; that is, find the linear combination of hypotheses that minimizes squared error.

Output the hypothesis:
$h_f(x) = a_1 h^1(x) + ... + a_{B+1} h^{B+1}(x) + a_{B+2}$

justed error and outputting the final hypothesis correspond to the same steps from Algorithm 1. The primary difference is in step 1 of each boosting iteration. After obtaining h_t, ExpBoost.R2 computes the weighted errors of each expert in H^B on the current weighting of T, and if any expert has a lower weighted error than h_t, h_t is replaced with the best expert.

3.2. (Boosted) Transfer Stacking

In ExpBoost.R2, the final hypothesis that is produced represents a combination of the provided experts and additional hypotheses learned with the base learner. However, at each boosting iteration, ExpBoost.R2 must choose between either the newly learned hypothesis or a single expert. We now consider relaxing this constraint by allowing a linear combination of hypotheses to be chosen at each iteration.

The details of our approach are very similar to those of stacking (or stacked generalization) (Wolpert, 1992). Stacking is an ensemble approach in which a meta-level model combines multiple base models, all trained independently on the same set of data using different learning algorithms. The meta-level model is learned (typically using linear regression) from a meta-level data set created as follows. For each instance in the original training set, a meta-level instance is created using the outputs of each base model as features and using the original label. Cross validation is performed for each base learner so that the output for each instance in the original training set is obtained when it is out-of-sample. Once the meta-level model is learned, a new instance is handled by using the model to combine the outputs of the base models on the instance.

Here, instead of multiple base learners, we consider a single base learner and (potentially) multiple experts previously trained on source data; hence cross validation is required only for the base learner and not for the experts. Thus, at each boosting iteration, we perform linear least squares regression to find a linear combination of the new hypothesis and experts that best fits the data for the current iteration, and store the result as the iteration's hypothesis. As a result of the similarity to stacking, we call this combination approach *transfer stacking*. Since our full boosting approach reduces to calling AdaBoost.R2 with transfer stacking as the base learner, we give details only for transfer stacking, shown as Algorithm 2.

We note that transfer stacking by itself (without the use of boosting) could be used as a transfer algorithm, and so in the experiments of Section 6 we evaluate both plain transfer stacking (using AdaBoost.R2 as its base learner for a fair comparison) and the full approach, which we call *boosted transfer stacking*.

3.3. Best Expert

Finally, as a baseline, we test the algorithm that simply uses the best expert from H^B; that is, the expert with the lowest error on the target training data.

4. Using Source Data Directly

We now describe three algorithms that take as input both the target and all source data sets and that train on a combination of this data.

4.1. TrAdaBoost.R2

Combining the principles of AdaBoost.R2 with those of TrAdaBoost results in the new regression algorithm TrAdaBoost.R2. TrAdaBoost.R2 takes two data sets as input, T_{target} and T_{source}, of size n and m, respectively, and combines them into a single set T used in boosting. Although the original work on TrAdaBoost does not consider the issue of multiple sources, we are interested in cases where any number of sources may exist. When there is more than one source, we simply combine all source data sets into a single data set. As TrAdaBoost.R2 handles the reweighting of each training instance separately, there should be no harm in mixing data in this fashion, but care should be taken in setting the initial weight vector. Our experiments involve source data sets of (roughly) equal sizes, and so we simply assign all source instances (and target instances) the same weight. If one source data set were larger than another, however, setting weights uniformly would result in more emphasis being given to that source, at least in early boosting iterations.

As with ExpBoost.R2, the steps involving computing the adjusted error correspond to the same steps from Algorithm 1. The primary difference between TrAdaBoost.R2 and Algorithm 1 is in step 5 of each iteration. Instead of treating all data equally, ExpBoost.R2 increases the weights of target instances by setting $w_i^{t+1} = w_i^t \beta_t^{-e_i^t}/Z_t$ and decreases the weights of source instances by setting $w_i^{t+1} = w_i^t \beta^{e_i^t}/Z_t$, where $\beta = 1/(1+\sqrt{2\ln n/N})$. In addition, TrAdaBoost.R2 considers only the final $\lceil N/2 \rceil$ hypotheses when taking the weighted median to determine output (as a result of theoretical considerations in the original TrAdaBoost).

4.2. Two-stage TrAdaBoost.R2

In analyzing the performance of TrAdaBoost.R2, we observed it to be highly susceptible to overfitting (that is, beyond some point, accuracy decreased as the number of boosting iterations N increased). In contrast, AdaBoost.R2 and the algorithms of Section 3 do not appear to suffer from this problem. After experimenting with cross validation to select N, we still saw mixed performance from TrAdaBoost.R2. Closer inspection of the results revealed two problems. First, when the size of T_{source} is much larger than T_{target}, it can take many iterations for the total weight of the target instances to approach the total weight of the source instances, and by this time the weights of the target data may be heavily skewed – those target instances that are either outliers or most dissimilar to the source data may represent most of the weight. Second, even those source instances that are representative of the target concept tend to have their weights reduced to zero eventually. The use of the adjusted error scheme from AdaBoost.R2 is the reason. Whereas in TrAdaBoost the relevant source instances will generally be classified correctly and not have their weights reduced, in TrAdaBoost.R2 even small errors lead to weight reductions. The fact that TrAdaBoost.R2 uses only the hypotheses generated during the final half of boosting iterations exacerbates this problem. (We note that we also tried using all hypotheses, with mixed results.)

To address these problems, we designed a version of TrAdaBoost.R2 that adjusts instance weights in two stages. In stage one, the weights of source instances are adjusted downwards gradually until reaching a certain point (determined through cross validation). In stage two, the weights of all source instances are frozen while the weights of target instances are updated as normal in AdaBoost.R2. Only the hypotheses generated in stage two are stored and used to determine the output of the resulting model. We call this algorithm two-stage TrAdaBoost.R2, and show it in Algorithm 3. Note that the weighting factor β_t is not chosen based on the hypothesis error, as before, but is chosen to result in a certain total weight for the target instances. In this way, the total weight of the target instances increases uniformly from $m/(n+m)$ to 1 in S steps. In our implementation, we approximated the value of β_t satisfying the conditions shown in Algorithm 3 using a binary search. In addition, it is not necessary to progress through all S steps once it has been determined that errors are increasing.

Algorithm 3 `Two-stage TrAdaBoost.R2`

Input two labeled data sets T_{source} (of size n) and T_{target} (of size m), the number of steps S, the maximum number of boosting iterations N, the number of folds F for cross validation, and a base learning algorithm *Learner*. Let T be the combination of T_{source} and T_{target} such that the first n instances in T are those from T_{source}. Set the initial weight vector $\mathbf{w}^1$ such that $w_i^1 = 1/(n+m)$ for $1 \le i \le n+m$.

For $t = 1, \ldots, S$:

1. Call AdaBoost.R2$'$ with T, distribution $\mathbf{w}^t$, N, and *Learner* to obtain $model_t$, where AdaBoost.R2$'$ is identical to AdaBoost.R2 except that the weights of the first n instances are never modified. Similarly, use F-fold cross validation to obtain an estimate $error_t$ of the error of $model_t$.
2. Call *Learner* with T and distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
3. Calculate the adjusted error e_i^t for each instance as in AdaBoost.R2.
4. Update the weight vector:

$$w_i^{t+1} = \begin{cases} w_i^t \beta_t^{e_i^t}/Z_t, & 1 \le i \le n \\ w_i^t/Z_t, & n+1 \le i \le n+m \end{cases}$$

where Z_t is a normalizing constant, and β_t is chosen such that the resulting weight of the target (final m) instances is $\frac{m}{(n+m)} + \frac{t}{(S-1)}(1 - \frac{m}{(n+m)})$.

Output $model_t$ where $t = argmin_i\ error_i$.

4.3. Best Uniform Initial Weight

Finally, as another baseline for comparison, we test an algorithm that simply calls AdaBoost.R2 with the combined source and target data, but attempts to find the best initial ratio of total weight between the source and target data. As in two-stage TrAdaBoost.R2, we try total target weights ranging from $m/(n+m)$ to 1 in S steps and choose the best weighting using cross validation. However, in this case all source instances have equal initial weights (i.e., there is no attempt to set individual weights based on errors), and no distinction is made between source and target instances once AdaBoost.R2 is called – source instances with high errors will have their weights increased just like target instances will. (In fact, this is not a boosting-specific algorithm, as any learner could be used in place of AdaBoost.R2; we use AdaBoost.R2 as the learner only to allow a direct comparison between the results.)

5. Data Transformation

In Section 2.1, we stated that both source and target concepts had labels in the same output space. In many regression settings in which we might consider transfer, however, different concepts might have labels with considerably different label distributions. While we largely view this as a data preparation issue (e.g., labels can be expressed in comparable terms, such as using relative instead of absolute prices in financial data) and thus beyond the scope of this paper, in our experiments we do take some simple measures to ensure similar label distributions.

In algorithms making use of experts trained on source data, we can directly modify the experts so that their outputs on the target data fall in an appropriate range. We do so by evaluating the experts on the target training data (thus making use only of data available to the learner) and performing linear regression to find the linear transformation that best fits the outputs to the true labels. This transformation is then applied whenever the expert is used by the learning algorithm. In algorithms using the source data directly, for each source data set we train an expert on the set, find the linear transformation in the same manner, and then apply this transformation to the labels in the source data set before passing it to the learning algorithm. On the data sets described in the following section, we found that this procedure was worthwhile, as it often resulted in a significant increase in accuracy while only occasionally producing a slight decrease in accuracy. We note that trying regression with higher degree polynomials tended to produce modest improvements at best and large decreases in accuracy at worst.

6. Experiments

We now evaluate our boosting algorithms on seven different problems: four data sets from the UCI Repository, a space of artificial data sets created from a known function, and two prediction problems from multiagent systems. Experiments were performed using the WEKA 3.4 (Witten & Frank, 1999) machine learning package with default parameters for the base learners. In the first group of experiments, we tested two base learners. For the remainder, we used the regression algorithm in WEKA giving the lowest error when used alone as the base learner. The following parameters were used (where appropriate): $N = 30$, $S = 30$, and $F = 10$. Experts were generated by running AdaBoost.R2 on a complete source data set. We used AdaBoost.R2 as the baseline non-transfer algorithm in each experiment as it consistently produced lower errors than using the base learner alone and offers a fair comparison against boosting transfer algorithms. Results said to be significant are statistically significant ($p < .05$) according to paired t-tests.

6.1. Four UCI Data Sets

We begin by comparing the results of all eight algorithms described above on four data sets taken from the UCI Machine Learning Repository[1]: concrete strength, housing, auto MPG, and automobile. (We chose the first four data sets that represented standard regression problems and had a few hundred instances; no other data sets were tried.) We divide these standard regression data sets into target and source sets by using a variation on the technique used by Dai et al. (2007) in a classification setting. For each data set, we identify a continuous feature that has a moderate degree of correlation (around 0.4) with the label. We then sort the instances by this feature, divide the set in thirds (low, medium, and high), and remove this feature from the resulting sets. By dividing based on a feature moderately correlated with the label, we hope to produce three data sets that represent slightly different concepts; if the correlation were zero, the concepts might be identical, and if the correlation were high, the concepts might be significantly different and have very different label ranges. In each experiment, we use one data set as the target and the other two as sources, for a total of 12 experiments.

Table 1 shows the results of all eight learning algorithms on all 12 experiments using both M5P model trees and neural networks as base learners. Target data training sets contained 25 instances. (Increasing this number resulted in qualitatively similar results.) Source data sets ranged from 68 to 343 instances. Each result represents the average RMS error over 30 runs. Numbers in bold represent results that are among the best – either the lowest error, or not significantly higher. Numbers in italics represent results that are not significantly better than AdaBoost.R2, that is, those where transfer failed.

The best expert is significantly better than AdaBoost.R2 exactly half of the time, but is sometimes much worse, suggesting that the degree of similarity between source and target data sets varies considerably across the range of experiments. Not surprisingly, the cases where the best expert fares worst are often those where other expert-based algorithms fare poorly.

ExpBoost.R2 performs poorly, beating AdaBoost.R2 significantly only five out of 24 times. Transfer stacking (performing stacking once with AdaBoost.R2 as a base learner) and boosted transfer stacking do much better, each beating AdaBoost.R2 significantly 15 times, suggesting that there is a benefit to considering linear combinations of models instead of only individual models. Interestingly, the error of transfer stacking is usually fairly close to that of boosted transfer stacking when both perform well. When both perform poorly, however, the error of boosted transfer stacking

[1] http://archive.ics.uci.edu/ml/index.html

is typically close to that of AdaBoost.R2, while the error of transfer stacking is much worse. It may be the case that performing transfer stacking across multiple boosting iterations is not necessary for effective transfer but is effective in preventing overfitting when transfer is not possible.

TrAdaBoost.R2 (with the number of boosting iterations chosen using cross validation) gives promising but somewhat erratic results, beating AdaBoost.R2 significantly 16 times but performing much worse in a few cases. Two-stage TrAdaBoost.R2 produces much better results and is the clear winner in this set of experiments, finishing among the top algorithms 20 out of 24 times and failing to significantly beat AdaBoost.R2 only once. Interestingly, simply finding the best uniform initial weighting also performs well, significantly outperforming AdaBoost.R2 17 times.

Overall, these results suggest that making use of source data directly is more effective than using source experts. However, the expert-based algorithms still had the best performance in a few cases, and it is worth noting that they are much less computationally intensive, due to using smaller amounts of data (target data only) and not requiring extensive cross validation.

For the remaining experiments, we note that boosted transfer stacking and two-stage TrAdaBoost.R2 (the primary contributions of this paper) continue to perform as well as or (usually) better than their counterparts (algorithms using source experts or source data, respectively), and so for clarity we omit the results of the other transfer algorithms.

6.2. Friedman #1

Friedman #1 (Friedman, 1991) is a well known regression problem, and we use a modified version that allows us to generate a variety of related concepts. Each instance x is a feature vector of length ten, with each component x_i drawn independently from the uniform distribution $[0, 1]$. The label for each instance is dependent on only the first five features:

$$\begin{aligned} y \quad = \quad & a_1 \cdot 10 \sin(\pi(b_1 x_1 + c_1) \cdot (b_2 x_2 + c_2)) + \\ & a_2 \cdot 20(b_3 x_3 + c_3 - 0.5)^2 + a_3 \cdot 10(b_4 x_4 + c_4) + \\ & a_4 \cdot 5(b_5 x_5 + c_5) + N(0, 1) \end{aligned}$$

where N is the normal distribution, and each a_i, b_i, and c_i is a fixed parameter. In the original Friedman #1 problem, each a_i and b_i is 1 while each c_i is 0, and we use these values when generating the target data set T_{target}. To generate each of the source data sets, we draw each a_i and b_i from $N(1, 0.1d)$ and each c_i from $N(0, 0.05d)$, where d is a parameter that controls how similar the source and target data sets are.

We performed experiments using several values d and values of 1 and 5 for B (the number of source data sets), expecting that transfer would be most effective for smaller values of d and the larger value of B. For each value of d, we randomly generated 100 of each of the following: i) target training data sets (of varying sizes), ii) target testing sets (of size 10,000), and iii) groups of 5 source data sets (each of size 1000). Neural networks were chosen as the best base learner.

Figure 1 shows the results when $d = 1$; results for other values of d were qualitatively similar. As expected, using transfer increased accuracy the most for lower values of d and higher values of B. When we used one source, boosted transfer stacking significantly outperformed AdaBoost.R2 when there were 250 target instances or fewer, while two-stage TrAdaBoost.R2 was significantly better than either algorithm for up to 300 instances. With five sources, boosted transfer stacking actually performed slightly better then 2-stage TrAdaBoost.R2 (the difference was significant for at least 75 instances), and both transfer algorithms were significantly better than AdaBoost.R2 for all points plotted.

6.3. TAC SCM and TAC Travel

While the previous data sets are useful for testing the performance of our algorithms, it is important to also experiment with naturally occurring data in domains where transfer would be applied in the real world. We now consider two such domains, taken from two e-commerce scenarios from the Trading Agent Competition: a supply chain management scenario (TAC SCM) (Eriksson et al., 2006), and a travel agent scenario (TAC Travel) (Wellman et al., 2007). In both scenarios, autonomous agents compete against each other in simulated economies to maximize profits. Many agents use some form of learning to make predictions about future prices, but the manner in which these prices change over time can depend heavily on the identities of the competing agents – essentially, different groups of agents represent different economies. This fact suggests the possibility of an agent using transfer learning to make use of past experience in different economies. In fact, many agents designed for the competition, while not explicitly casting the problem as transfer learning, deal in some way with the issue of making use of training data from these different sources. While these competitions are only abstractions of real-life markets, opportunities for applying transfer learning certainly exist in real markets as well, and these competitions represent valuable testbeds for research into these opportunities.

The first scenario we consider is TAC SCM, in which agents compete as computer manufacturers. We col-

Table 1. RMS error on four UCI datasets, each divided into three concepts, using M5P model trees and neural networks as base learners. **Bold**: lowest error; *Italic*: not significantly better than AdaBoost.R2 (95% confidence in each case)

Base Lrnr.	*Algorithm*	*Data set (divided into 3 subsets)*											
		Concrete Strength			Housing			Auto MPG			Automobile		
M5P	AdaBoost.R2	10.26	11.01	13.26	3.65	3.59	6.52	2.90	2.92	4.35	1963	3576	4893
	best expert	**8.63**	8.08	9.55	**2.98**	*5.27*	*9.98*	2.38	2.57	*4.44*	**1374**	*3741*	*6059*
	ExpBoost.R2	*10.11*	9.64	11.76	**3.02**	*3.74*	*6.66*	2.53	*2.94*	*4.39*	*1791*	*3661*	*4932*
	boosted t. stacking	**8.47**	7.48	10.03	**3.03**	*3.99*	*7.24*	2.30	2.75	*4.48*	**1325**	*3480*	*4811*
	transfer stacking	**8.60**	7.31	10.17	**3.07**	*5.49*	*8.39*	2.47	2.65	*4.57*	**1327**	*3631*	*5640*
	best unif. init. wt.	*10.25*	6.98	**8.66**	**2.99**	*3.42*	*6.52*	2.35	2.59	*4.33*	*1734*	2678	**2940**
	TrAdaBoost.R2 (CV)	*10.76*	7.04	9.71	3.38	*3.57*	*7.03*	2.19	2.58	**4.24**	*1815*	2851	3527
	2-Stage TrAdaBoost	**8.74**	**6.49**	**8.66**	**2.99**	**3.12**	**6.12**	**2.14**	**2.52**	**4.21**	1564	**2555**	3202
NN	AdaBoost.R2	10.47	11.95	14.84	3.89	3.67	7.54	2.76	3.55	5.17	1593	3200	3836
	best expert	*10.12*	9.67	13.87	*7.00*	*4.99*	*9.22*	*2.66*	2.77	4.43	*1481*	2484	*6119*
	ExpBoost.R2	*10.14*	*11.62*	13.37	*3.88*	*3.66*	*7.63*	*2.79*	*3.50*	*5.20*	1392	*3174*	*3829*
	boosted t. stacking	**9.49**	9.80	12.93	*3.75*	*3.58*	*7.74*	2.48	3.00	4.40	1215	2640	*3761*
	transfer stacking	**9.65**	9.46	13.13	*4.63*	*4.36*	*8.37*	2.43	2.83	4.53	**1144**	2632	*5081*
	best unif. init. wt.	*10.48*	**8.02**	10.77	*3.89*	**3.00**	**6.46**	2.44	2.80	**4.19**	1312	**2277**	**2858**
	TrAdaBoost.R2 (CV)	*11.34*	9.05	11.91	*4.02*	3.29	*7.68*	2.33	2.80	4.37	*1718*	2573	3268
	2-Stage TrAdaBoost	*10.43*	**8.09**	**9.92**	**3.27**	**2.99**	**6.45**	**2.14**	**2.60**	**4.18**	1290	**2276**	**2843**

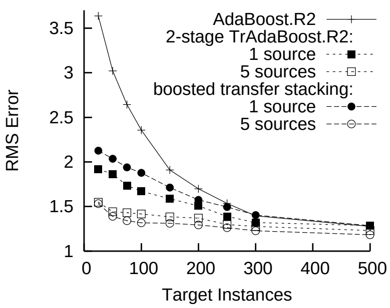

Figure 1. Friedman #1 (NN)

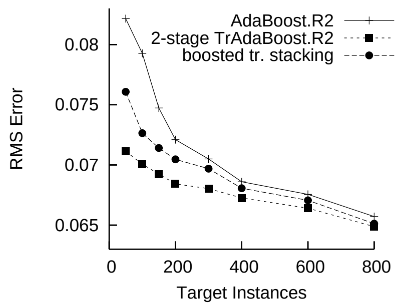

Figure 2. TAC SCM (M5P)

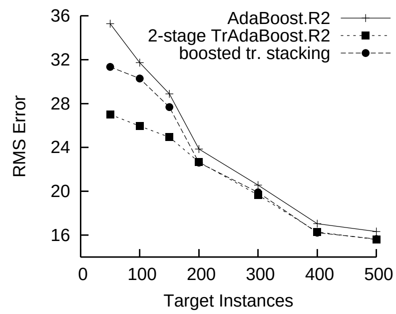

Figure 3. TAC Travel (M5P)

lected experience in three different economies as follows. We generated three source data sets using three different groups of agent binaries provided by competition participants. The target data set came from the final round of the 2006 competition. Each instance consists of 31 features of the economy at some point in time and is labeled with a particular change in future computer prices that would be of interest to an agent. Full details of these data sets are available in (Pardoe & Stone, 2007).

In TAC Travel, agents complete travel packages by bidding in simultaneous auctions for flights, hotels, and entertainment. We consider the problem of predicting the closing prices of hotel auctions given the current state of all auctions, represented by 51 features (as described in (Schapire et al., 2002)). We use data from the 2006 competition final round as the target data, and data from the 2004 and 2005 final rounds as the source data sets. (In each year's final round, all games consisted of the same agents, but between years the agents changed.)

Learning curves for 30 runs on each data set are shown in Figures 2 and 3. M5P model trees were chosen as the best base learner in both cases. Both two-stage TrAdaBoost.R2 and boosted transfer stacking significantly outperform AdaBoost.R2 for any number of target instances. Two-stage TrAdaBoost.R2 significantly outperforms boosted transfer stacking for any number of instances on the SCM data set and for 150 instances or less on the Travel data set.

7. Related Work

The most closely related work to this research, that on boosting and classification transfer, is described in Section 2. One important item we have not discussed, as this paper is empirical in its focus, is the theoretical properties of the algorithms discussed here. One of the attractive features of AdaBoost is its theoretical guarantees (e.g., convergence to zero error on the training set) (Freund & Schapire, 1997). We note that no theoretical results currently exist for ExpBoost or AdaBoost.R2; however, analogues of the main properties of AdaBoost have been proven to apply to TrAdaBoost, and a straightforward transformation of these proofs shows that these properties also extend to the combination of TrAdaBoost and AdaBoost.RT (mentioned in Section 2.3). Developing theoretical guarantees for the other algorithms discussed here, in both classification and regression settings, is an important area for future work.

The lowest common denominator of transfer learning methods is the leveraging of information from a source domain to speed up or otherwise improve learning in a different target domain. Transfer learning bears resemblance to classic case-based reasoning (Kolodner, 1993), especially in the need to reason about the similarity between tasks and instances. More recently, transfer learning has been studied in a variety of different settings, including statistical relational learning (Mihalkova et al., 2007), reinforcement learning (Taylor, 2009), and classification as described in Section 2. A key property of the classification and our regression setting is that the source and target domains typically have the same input and output spaces (X and Y in our notation), which is not always the case, for example in the reinforcement learning setting. As such, the problem studied here could be considered one of concept drift (Schlimmer & Granger, 1986), in which the target concept changes over time. This property also differentiates our setting from multitask learning (Caruana, 1997), in which multiple related concepts sharing an input representation but with potentially unrelated outputs are to be learned simultaneously; however, some multitask learning methods could potentially be modified to address our setting.

8. Conclusions and Future Work

We explored a number of boosting-based regression transfer algorithms that make use of either models trained on source data or the source data itself. The primary contribution of this paper is the introduction of boosted transfer stacking and two-stage TrAdaBoost.R2, both of which have their roots in existing classification transfer approaches. Both show promise, and two-stage TrAdaBoost.R2 in particular was consistently effective across a wide range of experimental domains.

There are a number of areas in which this work could be expanded. So far, we have only experimented with the domains and base learners described. Future work is needed to better understand which transfer algorithms are best suited for which domains, and how different choices of base learners and learning parameters interact with these algorithms. Also, additional methods of adapting boosting for regression could be explored, and additional techniques for improving boosting (such as regularization) could be tried. Finally, it would be interesting to see whether the extensions to ExpBoost and TrAdaBoost described here prove useful in the classification setting for which those algorithms were originally designed.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the FHWA (DTFH61-07-H-00030).

References

Caruana, Rich. Multitask learning. In *Machine Learning*, pp. 41–75, 1997.

Dai, Wenyuan, Yang, Qiang, Xue, Gui-rong, and Yu, Yong. Boosting for transfer learning. In *Proceedings of the Twenty Fourth International Conference on Machine Learning*, 2007.

Drucker, Harris. Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 107–115, 1997.

Eriksson, Joakim, Finne, Niclas, and Janson, Sverker. Evolution of a supply chain management game for the Trading Agent Competition. *AI Communications*, 19:1–12, 2006.

Freund, Yoav and Schapire, Robert. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

Friedman, Jerome. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19:1–141, 1991.

Kolodner, Janet. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

Mihalkova, Lilyana, Huynh, Tuyen, and Mooney, Raymond. Mapping and revising markov logic networks for transfer learning. In *Proceedings of the 22nd Conference on Artificial Intelligence*, pp. 608–614, July 2007.

Pan, Sinno Jialin and Yang, Qiang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2009. ISSN 1041-4347.

Pardoe, David and Stone, Peter. Adapting price predictions in TAC SCM. In *AAMAS 2007 Workshop on Agent Mediated Electronic Commerce*, 2007.

Rettinger, Achim, Zinkevich, Martin, and Bowling, Michael. Boosting expert ensembles for rapid concept recall. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, July 2006.

Schapire, Robert E., Stone, Peter, McAllester, David, Littman, Michael L., and Csirik, János A. Modeling auction price uncertainty using boosting-based conditional density estimation. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

Schlimmer, J. and Granger, R. Beyond incremental processing: Tracking concept drift. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pp. 502–507, 1986.

Shrestha, D. L. and Solomatine, D. P. Experiments with AdaBoost.RT, an improved boosting scheme for regression. *Neural Comput.*, 18(7):1678–1710, 2006.

Taylor, Matthew E. *Transfer in Reinforcement Learning Domains*. Springer Verlag, 2009.

Wellman, Michael P. Greenwald, Amy, and Stone, Peter. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, 2007.

Witten, Ian H. and Frank, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

Wolpert, David H. Stacked generalization. *Neural Networks*, 5:241–259, 1992.