2009 IEEE International Conference on Data Mining Workshops

Set-Based Boosting for Instance-level Transfer

Eric Eaton
*Lockheed Martin Advanced Technology Laboratories**
Artificial Intelligence Laboratory
Cherry Hill, NJ USA
eeaton@atl.lmco.com

Marie desJardins
University of Maryland Baltimore County
Department of Computer Science and Electrical Engineering
Baltimore, MD USA
mariedj@umbc.edu

***Abstract*—The success of transfer to improve learning on a target task is highly dependent on the selected source data. Instance-based transfer methods reuse data from the source tasks to augment the training data for the target task. If poorly chosen, this source data may inhibit learning, resulting in negative transfer. The current best performing algorithm for instance-based transfer, TrAdaBoost, performs poorly when given irrelevant source data.**

We present a novel set-based boosting technique for instance-based transfer. The proposed algorithm, *TransferBoost*, boosts both individual instances and collective sets of instances from each source task. In effect, TransferBoost boosts each source task, assigning higher weight to those source tasks which show positive transferability to the target task, and then adjusts the weights of the instances within each source task via AdaBoost. The results demonstrate that TransferBoost significantly improves transfer performance over existing instance-based algorithms when given a mix of relevant and irrelevant source data.

***Keywords*- knowledge transfer; boosting; ensemble methods**

I. Introduction

Learning in some domains is an expensive process due to the cost of obtaining labeled training instances. Knowledge transfer can reduce the cost of learning a model for a new *target* task by reusing information from previously learned *source* tasks. Model-based transfer techniques [1], [2] attempt to transfer parameter values or portions of source models to improve learning. While powerful, model-based transfer can be computationally expensive.

In contrast, instance-based transfer techniques reuse data from the source tasks to augment the target task's training data. Although the source and target tasks have different distributions, some of the source tasks' data could have been drawn from the target task's distribution. Such data could then be used as additional training data to improve learning on the target task.

Consider the following scenario: an investment company requires a model for predicting whether clients are good credit risks for recovery housing projects in New Orleans, following the devastation of hurricane Katrina in 2005. Since this is a new area for disaster recovery, there is extremely little data about the region. However, there are data from other areas that successfully recovered from disaster, and a large body of data on credit risk for other investment types, including mortgages, credit cards, and business loans. Surmising that the credit risks in New Orleans are somehow related to these other existing data, the investment company could determine which instances from these other data sets could be reused to learn their model for New Orleans. The instance-based transfer technique we explore in this paper addresses this problem.

We propose a novel *set-based boosting* technique to automatically select individual data from the source tasks to augment the target task's training data. Over the past two decades, boosting has become one of the foundational mechanisms for machine learning and data mining, by virtue of its ability to transform a weak learner into a robust classifier. Recently, boosting has been applied in combination with the weighted majority algorithm [3] to instance-based transfer in the TrAdaBoost algorithm [4]. We propose the *TransferBoost* algorithm as an improvement over TrAdaBoost to use boosting for transfer, overcoming several limitations inherent in TrAdaBoost's design.

TransferBoost automatically determines the weight to assign to each source instance in learning the target task's model, building on the AdaBoost algorithm [5]. TransferBoost iteratively constructs an ensemble of classifiers, reweighting both the source and target data via two types of boosting: individual and set-based. It increases the weight of individual mispredicted instances following AdaBoost. In parallel, it also performs *set-based boosting* by reweighting all instances from each source task based on their aggregate transfer to the target task. In effect, TransferBoost increases the weight of those source tasks that show positive transfer to the target task, and then reweights the instances within each task via AdaBoost.

Following a theoretical analysis of TransferBoost, we demonstrate its improved performance against several transfer algorithms in two real-world domains. The variety of transfer scenarios demonstrates TransferBoost's effectiveness in selecting source instances that improve performance on the target tasks.

* This work was conducted in its entirety while the first author was at the University of Maryland Baltimore County.

978-0-7695-3902-7/09 $26.00 © 2009 IEEE
DOI 10.1109/ICDMW.2009.97

II. Comparison with Related Work

TrAdaBoost [4] was the first successful application of boosting to knowledge transfer. TrAdaBoost considers the target data and source data separately. For the target data, it employs standard AdaBoost to increase the weight of mispredicted target instances, using the AdaBoost reweighting factor β_t computed from the training error on each iteration. For the source data, TrAdaBoost uses the weighted majority algorithm [3] to adjust the weights, repeatedly decreasing the weight of mispredicted source instances by a constant factor β, set according to Littlestone and Warmuth. Since the error of the weighted majority algorithm is only guaranteed to converge to zero (0) after $\lceil K/2 \rceil$ iterations, TrAdaBoost discards the first $\lfloor K/2 \rfloor$ ensemble members, basing the final classifier only on the $\lceil K/2 \rceil$th through Kth members.

One of TrAdaBoost's weaknesses is this choice to discard the first half of the ensemble. Intuitively, it is the classifiers from early iterations that fit the majority of the data, with later classifiers focusing on "harder" instances. Each additional classifier added by AdaBoost has (on average) less influence on the ensemble's prediction than its predecessors [6]. While discarding the early classifiers does ensure theoretical guarantees limiting the loss on the source data, discarding the earliest committee members could degrade the ensemble's performance in practice. Indeed, TrAdaBoost's performance is often improved when the first half of the ensemble is retained (personal communication with Dai et al.) at the cost of theoretical guarantees on the error.

Also, in TrAdaBoost's reweighting scheme, the difference between the weights of the source and target instances only increases. There is no mechanism in place to recover the weight of source instances in later boosting iterations when they become beneficial. For example, particular source instances may only be useful for constructing models for the harder target instances. The weights of these source instances have already decreased substantially from the early iterations, and once they become useful, their weights may be so tiny that they no longer have substantial influence.

Dai et al. [4] note that TrAdaBoost sometimes yields a final classifier that always predicts one label for all instances. They note that this occurs from TrAdaBoost substantially unbalancing the weights between the different classes of training instances, and compensate for it by resampling the data at each step to balance the classes, thereby encouraging the base learning algorithm to predict both classes. While seemingly effective in practice, the need for this adjustment reduces the algorithm's potential as a general-purpose transfer algorithm. Additionally, other researchers have noted that TrAdaBoost performs poorly when it is given irrelevant source data [7].

Beyond TrAdaBoost, the method proposed in this paper is related to the instance-based transfer method of Wu and Dietterich [8], which uses auxiliary data in SVMs to both constrain the learning and identify support vectors. Ensembles have been used previously for transfer by Gao et al. [9], who developed a locally weighted ensemble that weights each member classifier differently depending on the data region. Like Gao et al.'s method, TransferBoost can utilize base algorithms not inherently designed for transfer. Naïve Bayes, the base classifier in the experiments, has also been used before for transfer in a hierarchical method by Rosenstein et al. [10].

III. The TransferBoost Algorithm

TransferBoost is a novel set-based boosting method that improves upon TrAdaBoost for instance-based transfer. The TransferBoost algorithm learns an ensemble by boosting both individual instances and each set of instances corresponding to a source task.

We define a *task* as a mapping from an instance space $X \subset \mathbb{R}^d$ to a set of labels $Y \in \mathbb{N}$. Intuitively, a task is a particular label distribution over the data, in most cases corresponding to one labeled data set. In the transfer scenarios, there is one target task with a limited amount of labeled training data $T = \{(x_i, y_i)\}_{i=1}^{n}$ that is insufficient to learn the true mapping. Prior knowledge is available from the set of source tasks $S_1, \ldots, S_k$ with different mappings, each with numerous labeled training instances. Let source task $S_i = \{(x_j, y_j)\}_{j=1}^{|S_i|}$. All source tasks and the target task map from the same X to the same Y. TransferBoost's goal is to recover the true mapping $X \rightarrow Y$ for the target task, using a portion of the labeled source data to augment the limited target task data T in learning the model.

TransferBoost builds on AdaBoost to transfer source instances when learning a target distribution. The algorithm boosts each source task based on a notion of *transferability* [2] from the source task to the target task. Transferability is defined to be the change in performance on the target task between learning with and without transfer. While this imprecise definition suffices for now, Section III-C provides a formal definition of transferability, when it is used to compute the reweighting factors. Transferability can be either positive or negative, depending on whether transfer increases or decreases performance on the target task. Intuitively, transfer algorithms should avoid negative transfer. TransferBoost boosts each *set* of instances from the same task, increasing the weights of all instances from a source task if the source task shows positive transferability to the target task. Similarly, it decreases the weights of all instances from source tasks that show negative transferability to the target task.

Simultaneously, TransferBoost uses regular AdaBoost on individual source and target instances, adaptively increasing the weight of mispredicted instances to force learning to focus on these "harder" instances. Effectively, this adjusts the distribution of instance weights *within* each source task. Through this combination of novel set-based and regular

Algorithm 1 TransferBoost

Input: source tasks $S_1, \ldots, S_k$, where $S_i = \{(x_j, y_j)\}_{j=1}^{|S_i|}$, target training examples $T = \{(x_i, y_i)\}_{i=1}^{n}$, and the number of iterations K.

1: Merge the source and target training data $D = S_1 \bigcup \ldots \bigcup S_k \bigcup T$.
2: Initialize $w_1(x_i) = 1/|D|$, for $(x_i, y_i) \in D$. Let $\mathbf{w}_1(D)$ be the weight vector for all instances in D. (Optionally, these initial weights could be specified by the user.)
3: **for** $t = 1, \ldots, K$ **do**
4: Train model $h_t : X \rightarrow Y$ on D with weights $\mathbf{w}_t(D)$.
5: **for** $i = 1, \ldots, k$ **do**
6: Choose $\alpha_t^i \in \mathbb{R}$.
7: **end for**
8: Choose $\beta_t \in \mathbb{R}$.
9: Update the weights for all $(x_j, y_j) \in D$:

$$w_{t+1}(x_j) = \begin{cases} \frac{w_t(x_j)\exp\left(-\beta_t y_j h_t(x_j) + \alpha_t^i\right)}{Z_t} & (x_j, y_j) \in S_i \\ \frac{w_t(x_j)\exp(-\beta_t y_j h_t(x_j))}{Z_t} & (x_j, y_j) \in T \end{cases}$$

where Z_t normalizes $\mathbf{w}_{t+1}(D)$ to be a distribution.
10: **end for**

Output: the hypothesis $H(x) = sign\left(\sum_{t=1}^{K} \beta_t h_t(x)\right)$

boosting, TransferBoost automatically selects which source instances appear to be drawn from the target distribution.

TransferBoost is given as Algorithm 1. On each iteration t, the weights form a probability distribution over the training data, which is the union of the source and target data. TransferBoost trains a classifier with this distribution, yielding a hypothesis model h_t. It then chooses the reweighting factor α_t^i for each source task S_i based on the transferability from S_i to T, and the AdaBoost reweighting factor β_t for mispredicted instances. Section III-C describes a greedy approach for choosing the α_t parameters and β_t. TransferBoost then reweights each instance, increasing or decreasing its weight by a factor of $\exp(\alpha_t^i)$ based on whether its source task S_i shows (respectively) positive or negative transfer to the target task, and increasing its weight by a factor of $\exp(\beta_t)$ if the instance is mispredicted by h_t. The final ensemble classifier is given by a weighted sum of the individual classifiers' predictions, using the weights $\{\beta_t\}_{t=1}^{K}$, giving hypotheses with lower error more weight in the final decision.

After several iterations, instances from source tasks that show positive transfer to the target task will have higher weights, and source tasks that show negative transfer will have lower weights. Additionally, the distribution of instances within each source task and the target task will emphasize instances that are repeatedly mislabeled. In this manner, TransferBoost determines the instances that likely are from the target distribution.

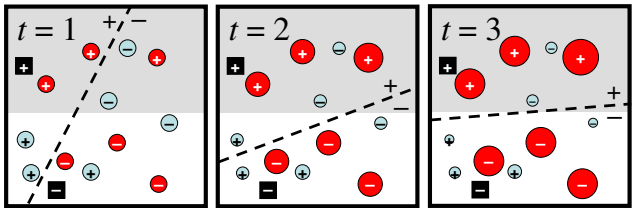


Figure 1. An illustration of TransferBoost depicting the first three boosting iterations. The scenario shows two source tasks (circle points of different colors) transferring to a target task (square points). Each point is labeled with its class (+/–), and the size of the point indicates its weight. The dashed line indicates the learned model h_t at each iteration. Note that the dark red source task transfers perfectly to the target task, while the light blue source task results in poor or negative transfer. After each iteration, TransferBoost increases the weights of the dark red source task since it shows strong transfer to the target task, resulting in the learned function approaching the target distribution given by the shaded regions.

A. Obtaining the source tasks

As described above, TransferBoost attempts to assign high weights to source instances that are drawn from the same distribution as the target task. If the target task is viewed a mixture of components, TransferBoost can be viewed as considering each source task as a possible component of this mixture. It can handle cases where most instances of a source task are from the target distribution but a few instances are not, by virtue of individually adjusting the distribution of instance weights *within* each source task.

However, it may the case that a given source task itself is a mixture of several components: one component shared with the target task, and another component from a different or conflicting distribution [11], [12]. All source instances that were drawn from the shared component could be used to augment the target training data. In such a case, the given source task should be split into its respective components and each component input to TransferBoost as a separate source task, thereby allowing TransferBoost to estimate whether each component is from the target distribution.

Various algorithms are available to break a source task into its individual components, including Gaussian mixture models, clustering algorithms, and latent Dirichlet allocation [13]. To determine the individual components of the source task, one of these models would be fit to the source task, and then each instance of the source task would be assigned to the component most likely to have generated it.

B. Analysis

For our theoretical analysis of TransferBoost, we begin by bounding the algorithm's training error on the target data. This bound is analogous to Schapire and Singer's [14] Theorem 1: that AdaBoost's training error is upper bounded by $\prod_{t=1}^{K} Z_t$. Indeed, TransferBoost with an empty set of source tasks reduces to AdaBoost (and Theorem 1 reduces to Schapire and Singer's Theorem 1).

Theorem 1. *The training error on the target task for TransferBoost is bounded by*

$$\frac{1}{|T|}\left|\{j : H(x_j) \neq y_j\}\right| \leq \frac{|D|}{|T|}\prod_{t=1}^{K} Z_t \left(\sum_{j\in T} w_{K+1}(x_j)\right) \ .$$

We omit the proof of this theorem due to lack of space; the interested reader may find it in Chapter 3 of the first author's dissertation [15].

There are several important consequences of Theorem 1. The first is that the training error can be minimized by greedily minimizing Z_t on each iteration, the same as in AdaBoost [14]. The second is that the weights assigned to the target data $\sum_{j\in T} w_{K+1}(x_j)$, which is in $(0, 1]$, must also be minimized. Equivalently, the source data weights $\sum_{i=1}^{k}\sum_{j\in S_i} w_{K+1}(x_j) = \left(1-\sum_{j\in T} w_{K+1}(x_j)\right)$ could be maximized.

C. Greedily choosing the α parameters and β.

Each boosting iteration, TransferBoost can greedily choose the α_t parameters based on *transferability* in order to maximize $\sum_{i=1}^{k}\sum_{j\in S_i} w_{K+1}(x_j)$. With this greedy choice of the α_t parameters, the optimal β_t that minimizes Z_t can be computed analytically.

In order to bound the training error, the algorithm should assign high weights to source instances from the same distribution as the target task. These source instances will improve learning on the target task, reducing both the training and the generalization errors. Conversely, the algorithm should assign low weights to source instances that are not from the target distribution, since those instances will interfere with learning the target task.

TransferBoost uses the concept of *transferability* to greedily compute α_t^i on each iteration for source task S_i. Following the definition by Eaton, desJardins, and Lane [2], transferability is the change in performance on a target task between learning with and without transfer. On each iteration t, TransferBoost trains a classifier $\vec{h}_t$ without transfer on the target data T with distribution $\frac{\mathbf{w}(T)}{||\mathbf{w}(T)||_1}$, where $||v||_1$ is the L_1-norm of vector v. Similarly, TransferBoost trains a classifier $\tilde{h}_t^i$ with transfer on $S_i \bigcup T$ with distribution $\frac{\mathbf{w}(S_i \cup T)}{||\mathbf{w}(S_i \cup T)||_1}$.

The weighted error of classifier h on dataset T at time t is given by

$$\epsilon = \sum_{(x_i,y_i)\in T} \frac{w_t(x_i)}{||\mathbf{w}_t(T)||_1}\left|h(x_i) - y_i\right| \ . \qquad (1)$$

Let $\vec{\epsilon}_t$ be the weighted error of $\vec{h}_t$ on T, and let $\tilde{\epsilon}_t^i$ be the weighted error of $\tilde{h}_t^i$ on T.

The transferability from S_i to T at time t is given by the difference in the errors between learning with and without transfer, so TransferBoost can greedily set $\alpha_t^i = \vec{\epsilon}_t - \tilde{\epsilon}_t^i$. Note that $\exp(\alpha_t^i) \in (1, e]$ when S_i shows positive transfer, $\exp(\alpha_t^i) = 1$ when there is no transfer, and $\exp(\alpha_t^i) \in [\frac{1}{e}, 1)$ when there is negative transfer.

With these greedy choices for the α_t parameters, TransferBoost can analytically choose β_t to minimize Z_t on each round of boosting. When the hypothesis is restricted such that $h \in [-1, 1]$,

$$\begin{aligned} Z_t &= \sum_{i=0}^{k}\sum_{j\in S_i} w_t(x_j)\exp\left(-\beta_t y_j h_t(x_j) + \alpha_t^i\right) \\ &= \sum_{i=0}^{k} e^{\alpha_t^i}\sum_{j\in S_i} w_t(x_j) e^{-\beta_t y_j h_t(x_j)} \ , \end{aligned}$$

with the convention that $S_0 = T$ and $\alpha_t^0 = 0$. Continuing to follow the analysis of Schapire and Singer [14], Z_t can be upper-bounded by

$$\sum_{i=0}^{k} e^{\alpha_t^i}\sum_{j\in S_i} w_t(x_j)\left(\frac{1+y_j h_t(x_j)}{2}e^{-\beta_t} + \frac{1-y_j h_t(x_j)}{2}e^{\beta_t}\right).$$

Since the α_i parameters are fixed, this expression can be minimized by

$$\beta_t = \frac{1}{2}\ln\left(\frac{1+\sum_{j\in D} w_t(x_j) y_j h_t(x_j)}{1-\sum_{j\in D} w_t(x_j) y_j h_t(x_j)}\right) \ , \qquad (2)$$

which is identical to the optimal β_t for AdaBoost [14]. Substituting this β_t into the upper-bound on Z_t yields

$$Z_t \leq \sqrt{1 - \left(\sum_{j\in D} w_t(x_j) y_j h_t(x_j)\right)^2} \ .$$

Now, TransferBoost's training error can be bounded by:

$$\frac{1}{|T|}\left|\{j : H(x_j) \neq y_j\}\right| \leq \qquad (3)$$
$$\frac{|D|}{|T|}\prod_{t=1}^{K}\sqrt{1 - \left(\sum_{j\in D} w_t(x_j) y_j h_t(x_j)\right)^2}\left(\sum_{j\in T} w_{K+1}(x_j)\right) \ .$$

While this greedy choice of the α_t parameters and β_t ensures a rapid runtime, TransferBoost could ensure a tighter bound on the training error through numeric optimization to minimize Z_t on each iteration. This tighter bound will be explored in future work.

IV. Experiments

The experiments compare TransferBoost against three algorithms: TrAdaBoost, AdaBoost(T) trained on only the target data, and AdaBoost(S&T) trained on both the source and target data. Naïve Bayes is used as the base classifier for all methods, and the number of boosting iterations is set to 100. The implementation of TransferBoost is available for download as an extension to the Weka machine learning toolkit [16] at http://maple.cs.umbc.edu/∼ericeaton/TransferBoost/.

Data Set	AdaBoost(T)			AdaBoost(S&T)			TrAdaBoost			TransferBoost		
	1%	10%	25%	1%	10%	25%	1%	10%	25%	1%	10%	25%
comp.sys.ibm.pc.hardware	52.8−	59.0−	66.4−	**68.0**	**69.4**	71.2	50.0−	61.4−	67.2−	66.8	68.6	**71.9**
comp.windows.x	51.2−	59.8−	**66.4**	59.0	59.3−	60.4−	50.0−	61.8	65.5	**59.7**	**63.2**	63.9
rec.sport.baseball	51.8−	60.2	66.4	59.9	59.7−	60.1−	50.0−	59.0−	**67.0**+	**60.2**	**62.2**	64.3
sci.electronics	52.0−	51.0−	53.6−	57.7	**58.1**	58.4	50.0−	53.6−	55.2−	**58.4**	57.4	**58.5**
sci.med	49.2−	57.0	**62.2**	53.2	55.4−	54.8−	50.0−	55.6−	61.6	**53.7**	**58.2**	61.3
talk.politics.mideast	51.0	53.3−	56.3	51.1	52.2−	51.8−	50.2	54.7	57.0	**51.6**	**57.1**	**58.9**
talk.politics.misc	**50.4**	53.8+	**56.0**	47.4−	47.3−	48.2−	49.9	**55.0**+	55.6	49.7	49.9	54.1
letter-A	49.8−	69.6−	**84.2**	54.9−	58.9−	64.0−	50.0−	74.4	82.2	**62.4**	**76.6**	82.9
letter-B	49.8−	63.7	**74.8**+	55.4−	54.9−	55.4−	50.0−	**66.7**	71.8	**59.0**	63.5	69.7
letter-C	50.9−	64.9−	82.9	74.4	74.8−	75.4−	50.0−	69.3−	79.3−	**76.3**	**81.9**	**83.9**
letter-D	49.5−	64.5	**76.5**+	**56.6**	56.2−	59.4−	50.5−	**66.6**	72.9	55.7	64.7	72.1
letter-E	50.5−	64.1−	**79.8**+	65.8−	65.2−	64.4−	50.3−	66.3−	77.2	**68.4**	**71.7**	76.1
letter-F	**50.4**+	**64.6**+	**76.8**+	42.1−	46.8−	53.7−	**50.4**+	64.0	73.6+	46.7	61.6	67.0
letter-G	52.0−	68.1−	81.1	74.1	74.7−	74.9−	50.7−	68.4−	78.4−	**75.1**	**78.9**	**82.9**
letter-H	50.9−	55.4	**61.0**+	51.7−	51.1−	51.3−	49.9−	**58.9**	60.4+	**54.4**	57.3	57.0
letter-I	58.3−	73.8−	**93.4**+	82.5	83.2−	83.3−	50.7−	77.0−	84.6−	**83.2**	**88.2**	89.8
letter-J	51.6−	61.4−	74.3−	66.7−	66.6−	68.0−	49.8−	66.1−	74.0−	**69.3**	**73.1**	**77.0**
letter-K	50.2−	60.8−	73.0−	73.5	74.8−	73.9−	49.9−	66.0−	70.4−	**74.2**	**77.7**	**78.1**
letter-L	54.4−	62.0−	80.4−	**78.8**	81.2−	81.6−	50.7−	72.8−	79.9−	78.1	**86.0**	**88.0**
letter-M	50.9−	70.1−	**83.4**+	49.4−	55.7−	65.1−	50.6−	73.7	79.9	**56.9**	**74.5**	79.9

Table I
A COMPARISON OF THE ALGORITHMS' PERFORMANCES AT 1%, 10%, AND 25% OF THE TARGET TRAINING DATA, WITH THE BEST PERFORMANCE OF EACH EXPERIMENT IN BOLD. STATISTICALLY SIGNIFICANT DIFFERENCES AGAINST TRANSFERBOOST ARE MARKED BY $+/-$, WITH $+$ INDICATING THE PERFORMANCE WAS SIGNIFICANTLY BETTER THAN TRANSFERBOOST, AND $-$ INDICATING THAT IT WAS SIGNIFICANTLY WORSE.

We conducted the experiments using tasks from two domains: letter and newsgroup recognition. The Letters data set [17] represents various fonts of each letter using 16 features normalized to the range $[0, 1]$. For the 20 Newsgroups data set [18], we represented each post as a binary vector of the 100 most discriminating words determined by Weka's string-to-wordvector filter [16]. We use only 5% of the original data sets in the experiments, since the originals are very large.

For each domain, we generated a set of binary tasks (Table II) to distinguish one class from a set of negative classes, ensuring that each task had unique negative examples and equal class priors. With this construction, each task will have unique data, enabling us to attribute performance improvements to transfer. For Letters, we use the letters A–M as the positive classes against the letters N–Z, yielding 13 tasks. For example, the task of recognizing the letter C used 35 "C"s as positive examples and 35 random letters N–Z as negative examples. The Newsgroups tasks are constructed similarly to the Letters tasks, using the first newsgroup in each major category as negative examples for the tasks given by the 13 remaining newsgroups. These negative examples are drawn from the following newsgroups: alt.atheism, comp.graphics, misc.forsale, rec.autos, sci.crypt, soc.religion.christian, and talk.politics.guns.

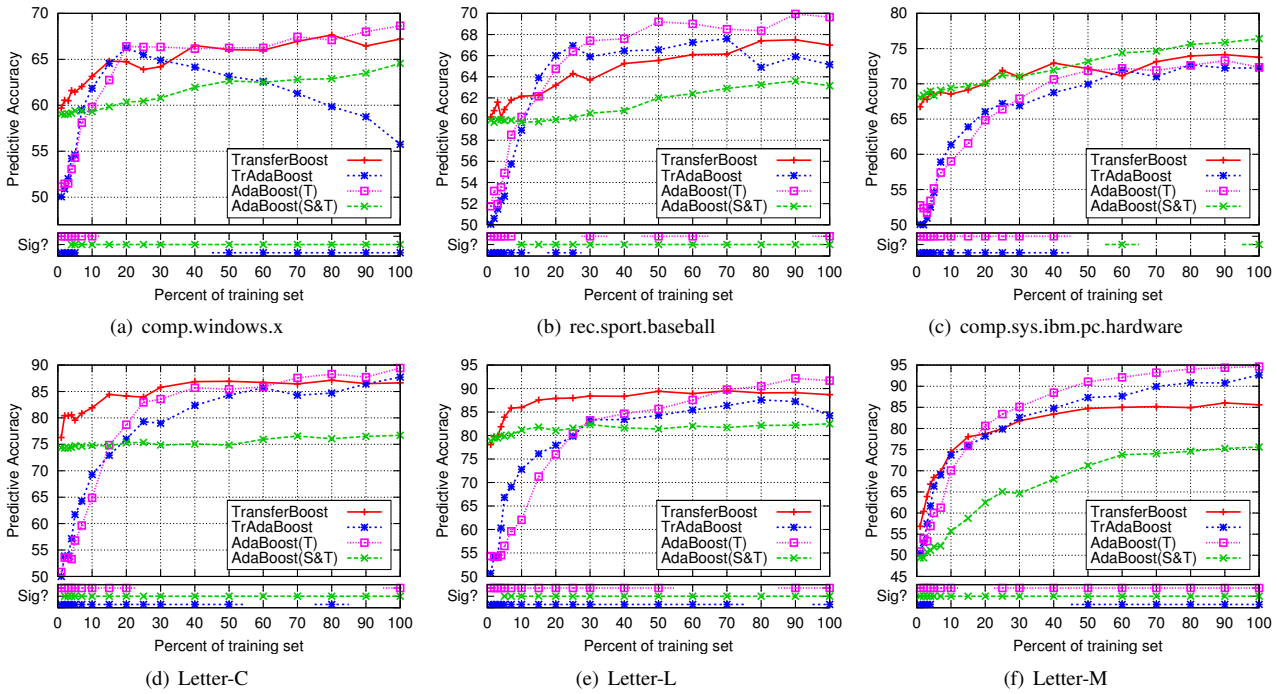
Table II
SUMMARY OF TASKS.

20 Newsgroups		**Letters**	
Task	**#inst**	**Task**	**#inst**
comp.os.ms-windows.misc	100	A	86
comp.sys.ibm.pc.hardware	100	B	74
comp.sys.mac.hardware	98	C	70
comp.windows.x	100	D	70
rec.motorcycles	100	E	88
rec.sport.baseball	100	F	84
rec.sport.hockey	100	G	70
sci.electronics	100	H	70
sci.med	100	I	60
sci.space	100	J	72
talk.politics.mideast	94	K	74
talk.politics.misc	78	L	68
talk.religion.misc	64	M	92

Each experiment uses one task as the target task; all other tasks from the same domain serve as the source tasks. The learning algorithms were trained on all data on the source tasks and a subset of the target training data. The learning curves for the algorithms were generated over 20 trials of 10-fold cross-validation on the target data. Although each individual task contains few instances, TransferBoost acts on all tasks from a domain in each experiment. Therefore, TransferBoost acts on 1,234 instances in each Newsgroup experiment and 978 instances in each Letter experiment.

Table I compares the predictive accuracy of the algorithms when trained on 1%, 10%, and 25% of the target training data. Statistical significance against TransferBoost was assessed using a paired t-test with $\alpha = 0.05$. For several example tasks, Figure 2 shows their full learning curves, with statistical significance from TransferBoost noted as the underlined portions below the plots.

These results show that TransferBoost performs the best overall given little target data (1% and 10%), and then the



(a) comp.windows.x (b) rec.sport.baseball (c) comp.sys.ibm.pc.hardware

(d) Letter-C (e) Letter-L (f) Letter-M

Figure 2. Instance-based transfer learning curves on example target tasks, with significant differences from TransferBoost as measured by a paired t-test with $\alpha = 0.05$ marked below each plot.

best performance shifts to AdaBoost(T) trained on only the target data. The results indicate that TransferBoost clearly outperforms TrAdaBoost when given little training data. This performance improvement can be attributed to the set-based boosting ability of TransferBoost to boost all instances of the relevant source tasks, rather than just individual instances as in TrAdaBoost. We also experimented with a modified TrAdaBoost that used the full ensemble to correct for the problem of using only the last $\lceil \frac{K}{2} \rceil$ members of the ensemble (see Section II), but the results were not significantly different from the original TrAdaBoost algorithm.

Theoretically, transfer will not improve learning when given enough target data to learn a model with high performance.[1] This shift is indicated in the learning curves by the best performance shifting from TransferBoost to AdaBoost(T). The crossover in the learning curves indicates the point at which there is enough target data such that transfer is unnecessary. In some scenarios, none of the source tasks appear to transfer well to the target task, resulting in AdaBoost(T) having higher performance than all algorithms utilizing the source data (TransferBoost, TrAdaBoost, and AdaBoost(S&T)).

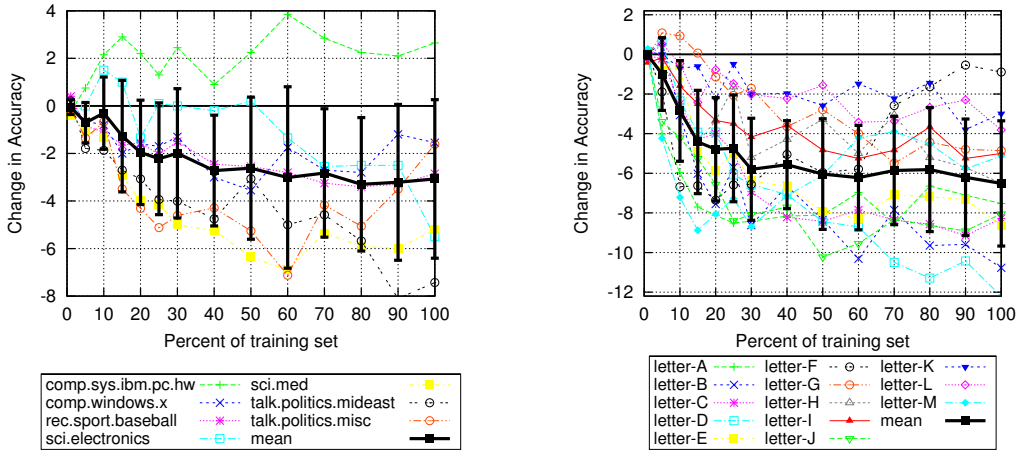
Experiments were also run using AdaBoost's early termination mechanism in TransferBoost, stopping the algorithm when $\vec{\epsilon}_t$ reached 0 or exceeded 0.5. Early termination with TransferBoost does not yield optimal results as with AdaBoost, since the performance may still be improved by reweighting the source data. Figure 3 shows the performance loss of early termination over $K = 100$ boosting iterations for each domain. This figure indicates a rough assessment of the average performance improvement obtained by adjusting the source data weights once $\vec{\epsilon}_t$ reaches 0 or exceeds 0.5.

These early termination results also explain the poor performance on one of the newsgroup tasks: comp.sys.ibm.pc.hardware (Figure 2(c)). Early termination on this target task increases the performance of TransferBoost an average of approximately two percentage points, which would have ranked it above the other methods. Therefore, TransferBoost's poor performance on this particular task was caused by using too many boosting iterations. TransferBoost also performed below other methods on several other tasks, but early termination does not help explain these results. From these experiments with early termination, it appears that TransferBoost may be sensitive to the number of boosting iterations in some situations. Exploration of this avenue is left to future work.

V. Conclusion and Future Work

Set-based boosting is a novel approach to knowledge transfer, and the experiments show it to be successful in practice. It is especially useful when given little target data, since it can identify source tasks that transfer well to the target task.

[1] It is for this reason that we restrict the size of each learning task.



(a) Newsgroups (b) Letters

Figure 3. TransferBoost's performance loss due to early termination on the Newsgroups (a) and Letters (b) domains. In each plot, the heavy black line indicates the mean over all tasks, with the standard deviation given by the error bars.

The current TransferBoost algorithm uses transferability to heuristically estimate the reweighting factor for each source task. While the results show this method to be effective, the α parameters and β could also be set via numerical optimization, which is left to future work. Also, the results with early termination indicate that correctly choosing the number of boosting iterations may be important to maximize TransferBoost's potential. Guidelines for setting the number of boosting iterations, and a study of its effect on TransferBoost, is left to future work. Additionally, TransferBoost's set-based boosting may have applications in addition to knowledge transfer as a standard boosting algorithm for machine learning and data mining.

ACKNOWLEDGMENTS

This work was supported by NSF ITR-0325329. We thank Terran Lane, Tim Oates, and Yun Peng for their feedback on this work, Wenyuan Dai for discussions on TrAdaBoost, and the reviewers for their constructive comments.

REFERENCES

[1] R. Raina, A. Y. Ng, and D. Koller, "Constructing informative priors using transfer learning," in *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: ACM, 2006, pp. 713–720.

[2] E. Eaton, M. desJardins, and T. Lane, "Modeling transfer relationships between learning tasks for improved inductive transfer," in *Proceedings of the 19th European Conference on Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 317–332.

[3] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212–261, 1994.

[4] W. Dai, Q. Yang, G. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of the 24th International Conference on Machine Learning*. New York, NY, USA: ACM, 2007, pp. 193–200.

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[6] L. Reyzin and R. E. Schapire, "How boosting the margin can also boost classifier complexity," in *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: ACM, 2006, pp. 753–760.

[7] X. Shi, W. Fan, and J. Ren, "Actively transfer domain knowledge," in *Proceedings of the 19th European Conference on Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 342–357.

[8] P. Wu and T. G. Dietterich, "Improving SVM accuracy by training on auxiliary data sources," in *Proceedings of the 21st International Conference on Machine Learning*. New York, NY, USA: ACM, 2004, pp. 871–878.

[9] J. Gao, W. Fan, J. Jiang, and J. Han, "Knowledge transfer via multiple model local structure mapping," in *Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2008, pp. 283–291.

[10] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, "To transfer or not to transfer," in *NIPS 2005 Workshop on Inductive Transfer*, Whistler, BC, Canada, 2005.

[11] S. Kaski and J. Peltonen, "Learning from relevant tasks only," in *Proceedings of the 18th European Conference on Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 608–615.

[12] H. Daumé III and D. Marcu, "Domain adaptation for statistical classifiers," *Journal of Artificial Intelligence Research*, vol. 26, pp. 101–126, 2006.

[13] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[14] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Proceedings of the 11th Conference on Computational Learning Theory*. New York, NY, USA: ACM, 1998, pp. 80–91.

[15] E. Eaton, "Selective knowledge transfer for machine learning," Ph.D. dissertation, University of Maryland Baltimore County, 2009.

[16] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools with Java Implementations*. San Francisco, CA: Morgan Kaufmann, 2000.

[17] A. Asuncion and D. Newman. (2007) The University of California Irvine (UCI) machine learning repository. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[18] J. Rennie. (2003) 20 Newsgroups data set, sorted by date. [Online]. Available: http://www.ai.mit.edu/ ∼jrennie/20Newsgroups/