Selective Transfer Between Learning Tasks Using Task-Based Boosting

Eric Eaton
Bryn Mawr College
Computer Science Department
Bryn Mawr, PA USA
eeaton@cs.brynmawr.edu

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 learning on a target task is highly dependent on the selected source data. Instance 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 most widely used algorithm for instance transfer, TrAdaBoost, performs poorly when given irrelevant source data.

We present a novel task-based boosting technique for instance transfer that selectively chooses the source knowledge to transfer to the target task. Our approach performs boosting at both the instance level and the task level, assigning higher weight to those source tasks that show positive transferability to the target task, and adjusting the weights of individual instances within each source task via AdaBoost. We show that this combination of task- and instance-level boosting significantly improves transfer performance over existing instance transfer algorithms when given a mix of relevant and irrelevant source data, especially for small amounts of data on the target task.

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 transfer techniques (Raina et al. 2006; Eaton et al. 2008) attempt to transfer parameter values or model components to improve learning. While powerful, model transfer can be computationally expensive.

In contrast, instance 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 for the target task.

Consider the following scenario: an investment company needs to predict 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 disaster recovery projects, and a large body of data on credit risk for other investment types. Surmising that the credit risks in New Orleans are related to these other existing data, the company could determine which instances could be reused to learn their model for New Orleans. The instance transfer technique we explore in this paper addresses this problem.

We propose a novel *task-based boosting* method to automatically select individual data from the source tasks to augment the target task's training data. Recently, boosting has been applied in combination with the weighted majority algorithm (Littlestone and Warmuth 1994) to instance transfer in the TrAdaBoost algorithm (Dai et al. 2007). We propose the *TransferBoost* algorithm as an improvement over TrAdaBoost and its variants 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 (Freund and Schapire 1997). TransferBoost iteratively constructs an ensemble of classifiers, reweighting both the source and target data via two types of boosting: individual and task-based. It increases the weight of individual mispredicted instances following AdaBoost. In parallel, it also performs *task-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 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, showing its effectiveness in selecting source instances that improve learning on the target tasks.

Comparison with Related Work

TrAdaBoost (Dai et al. 2007), the first application of boosting to knowledge transfer, considers the target 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. For the source data, TrAdaBoost uses the weighted majority algorithm (Littlestone and Warmuth 1994) to adjust the weights, repeatedly decreasing the

weight of mispredicted source instances by a constant factor β, which is set following 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 for K ensemble members, 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. 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 (Reyzin and Schapire 2006). While discarding the early classifiers does ensure theoretical limits on the loss over the *source* data, discarding the early committee members could degrade the ensemble's performance on the *target* task 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 these theoretical guarantees.

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. (2007) 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, and compensate for it by resampling the data at each step to balance the classes, thereby encouraging the base classifiers to predict both classes. Additionally, other researchers have noted that TrAdaBoost performs poorly when given irrelevant source data (Shi et al. 2008). Yao and Doretto (2010) propose the MultiSourceTrAdaBoost algorithm to overcome this limitation by considering each source task individually in combination with the target and retaining only the single best combination after each boosting round.

Beyond TrAdaBoost and MultiSourceTrAdaBoost, our proposed algorithm is related to the instance transfer method of Wu and Dietterich (2004), 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. (2008), 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. (2005).

The TransferBoost Algorithm

We define a *task* as a mapping from an instance space $X \subset \mathbb{R}^d$ to a set of discrete labels Y. 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 $S_i = \{(x_j, y_j)\}_{j=1}^{|S_i|}$. TransferBoost's goal is to recover the true mapping $X \rightarrow Y$ for the target task, using instances from $S_1, \ldots, S_k$ 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* (Eaton et al. 2008) from the source task to the target task. Transferability is the change in performance on the target task between learning with and without transfer; we later provide a formal definition in the *Analysis* section. Transferability can be 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 it 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.

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 novel combination of task- and instance-based 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 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. The *Analysis* section describes a greedy approach for choosing the α_t^i's 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 will have higher weights, and source tasks that show negative transfer will have lower weights. Additionally, the weight distribution within each source task and the target task will emphasize instances that are repeatedly mislabeled. In this manner, TransferBoost determines the instances that are likely from the target distribution.

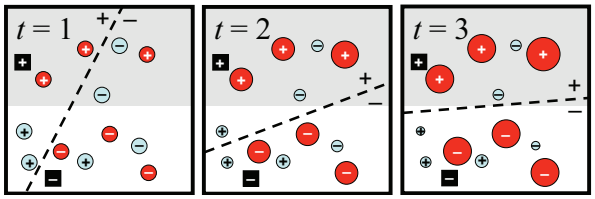


Figure 1: An illustration of TransferBoost's first three boosting iterations. The scenario shows two source tasks (circle points of different colors) transferring to a target task (square points). Each point is labeled with its class (+/–), and the size of the point indicates its weight. The dashed line 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.

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 as 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 (Kaski and Peltonen 2007; Daumé III and Marcu 2006). 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 (Blei et al. 2003). To determine the individual components, one of these models would be fit to the source task, and then each instance would be assigned to the component most likely to have generated it.

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 (1998) 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 the standard AdaBoost bound).

Algorithm 1 TransferBoost [1]

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

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 $(x_i, y_i) \in D$. (Optionally, these initial weights could be specified by the user.)
3: **for** $t = 1, \ldots, K$ **do**
4: Train model $h_t : X \to Y$ on D with weights $\mathbf{w}_t(D)$.
5: Choose $\alpha_t^i \in \mathbb{R}$ for $i = 1, \ldots, k$.
6: Choose $\beta_t \in \mathbb{R}$.
7: 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.
8: **end for**

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

Theorem 1. *The training error* $\epsilon_T = \frac{1}{|T|}\left|\{j : H(x_j) \neq y_j\}\right|$ *on the target task for TransferBoost is bounded by* [1]

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

There are several important consequences of Theorem 1. The first is that ϵ_T can be minimized by greedily minimizing Z_t on each iteration, as in AdaBoost. The second is that the sum of the weights assigned to the target data, $\sum_{j \in T} w_{K+1}(x_j)$, should also be minimized. Equivalently, we could maximize the sum of the source data weights $\sum_{i=1}^{k} \sum_{j \in S_i} w_{K+1}(x_j) = \left(1 - \sum_{j \in T} w_{K+1}(x_j)\right)$.

Since TransferBoost's resulting ensemble model is of the same form as AdaBoost, its generalization error is similarly upper-bounded by $\epsilon_T + O\left(\sqrt{\frac{K}{n} d_{VC}}\right)$, where d_{VC} is the VC-dimension of the base classifier (Schapire and Singer 1998).

Greedily choosing the α parameters and β

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, the optimal β_t that minimizes Z_t can be computed analytically.

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, 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.

[1] An implementation of TransferBoost and the proof of Theorem 1 may be found in a supplement available online at http://cs.brynmawr.edu/∼eeaton/TransferBoost/.

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 et al. (2008), 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 data set T is given by

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

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. When the hypothesis is restricted such that $h \in [-1, 1]$,

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

with the convention that $S_0 = T$ and $\alpha_t^0 = 0$. Following Schapire and Singer (1998), 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's are fixed, this can be minimized by choosing

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

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

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

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

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

While this greedy choice of α_t 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, which we will explore in future work.

Experiments

The experiments compare TransferBoost against four algorithms: TrAdaBoost, MultiSourceTrAdaBoost, 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.

Table 1: 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

We use tasks from two domains: letter and newsgroup recognition. The Letters data set (Asuncion and Newman 2007) represents various fonts of each letter using 16 features normalized to the range $[0, 1]$. For the 20 Newsgroups data set (Rennie 2003), we represented each post as a binary vector of the 100 most discriminating words determined by Weka (Witten and Frank 2000). We use only 5% of the original data sets to challenge the transfer methods.

For each domain, we generated a set of tasks (Table 1) to distinguish one class from a set of negative classes, ensuring that each task had unique negative examples and equal class priors. 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, using the first newsgroup in each category as negative examples for the tasks given by the 13 remaining[2] newsgroups.

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.

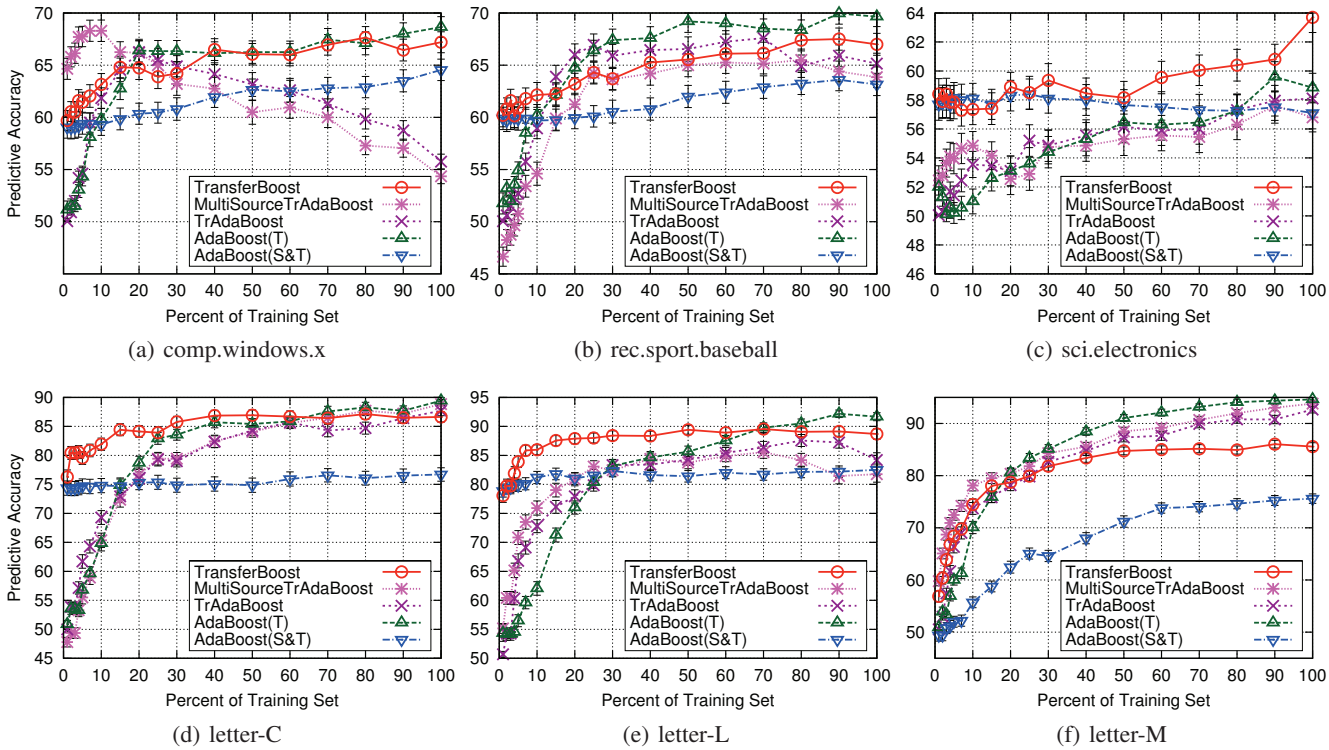
Table 2 compares the predictive accuracy of the algorithms when trained on 1%, 10%, and 25% of the target training data. Statistical significance against TransferBoost was assessed using a paired t-test with $\alpha = 0.05$. For several example tasks, Figure 2 shows their full learning curves.

The results indicate that TransferBoost clearly outperforms TrAdaBoost and MultiSourceTrAdaBoost when given little target training data. The strong performance of TransferBoost in scenarios with little training data is especially

[2]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.

Target Task	AdaBoost(T)			AdaBoost(S&T)			TrAdaBoost			MultiSrcTrAdaBoost			TransferBoost		
	1%	10%	25%	1%	10%	25%	1%	10%	25%	1%	10%	25%	1%	10%	25%
comp.sys.ibm.pc.hw	52.8−	59.0−	66.4−	68.0	69.4	71.2	50.0−	61.4−	67.2−	**73.5+**	**71.8+**	68.1−	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	**64.7+**	**68.3+**	65.0	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+**	46.7−	54.6−	64.3	**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−	52.5−	54.9	52.9−	**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	45.6−	51.7−	61.5	**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	49.4	52.1−	55.3−	**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	41.2−	50.5	55.0	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	43.2−	71.2−	80.9	**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	53.8−	66.3	**75.4+**	**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−	47.8−	65.4−	79.4−	**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	51.6−	65.5	75.8+	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	51.4−	65.5−	76.7	**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+	**50.4+**	**65.1+**	73.9+	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−	53.9−	69.4−	82.6	**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+	50.6−	**59.9**	60.1	**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−	55.1−	78.2−	89.2	**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−	52.7−	69.0−	74.7	**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−	51.7−	62.3−	71.0−	**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−	55.2−	75.9−	83.1−	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	**59.8+**	**78.1+**	81.8	56.9	74.5	79.9

Table 2: A comparison of the algorithms' predictive accuracies 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.



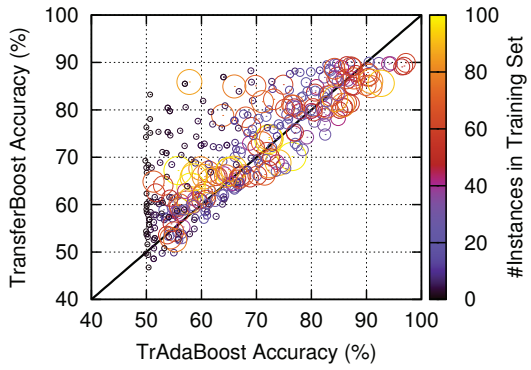
(a) comp.windows.x (b) rec.sport.baseball (c) sci.electronics

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

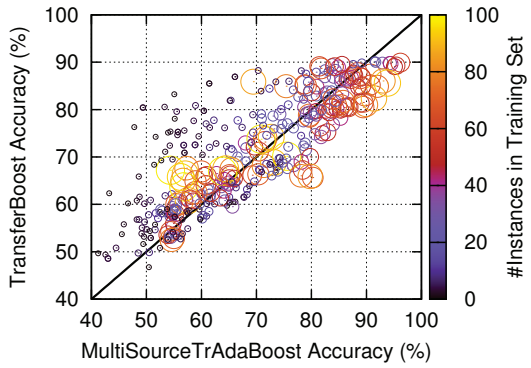
Figure 2: Learning curves on example target tasks, with error bars denoting the standard error of the mean.

apparent in Figure 3. This improvement can be attributed to the ability of TransferBoost to boost all instances of the relevant source tasks, rather than just individual instances as in TrAdaBoost and its variant. We also experimented with a modified TrAdaBoost that used the full ensemble to correct for the problem of using only the last $\lceil \frac{K}{2} \rceil$ ensemble members. Although the modification improved TrAdaBoost's accuracy overall, it had qualitatively similar performance to the original algorithm in comparison with TransferBoost.

As the amount of target data increases, the performance differences between the transfer algorithms become less apparent, with TransferBoost still retaining a slight edge over



(a) TransferBoost vs TrAdaBoost



(b) TransferBoost vs MultiSourceTrAdaBoost

Figure 3: Comparison of TransferBoost's performance with TrAdaBoost and MultiSourceTrAdaBoost over all tasks. Each circle depicts an average of 200 experiments on a single task, with a given amount of target training data denoted by the size and color of the circle.

TrAdaBoost overall. For large amounts of data, the best algorithm seems largely dependent on the task, with TransferBoost being best for some tasks and MultiSourceTrAdaBoost being best for others. In some cases, TrAdaBoost and its variant suffer from declining performance as the amount of target data increases (e.g., Figure 2(a)); TransferBoost does not seem to suffer from this problem, which is most likely prevented by boosting at the task level.

Theoretically, transfer will not improve learning when given enough target data to learn a model with high performance.[3] This shift is indicated in the learning curves by the best performance shifting from TransferBoost to AdaBoost(T). The crossover indicates the point where 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.

Conclusion and Future Work

Task-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.

TransferBoost uses transferability to estimate the reweighting factor for each source task S_i. 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. We have also seen indications 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, are left to future work.

[3]For this reason, we restrict the size of each learning task.

Acknowledgments

This work was supported by ONR award #N00014-11-1-0139 and 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

Asuncion, A., and Newman, D. 2007. UCI repository. Available at http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent Dirichlet allocation. *J. Machine Learning Research* 3:993–1022.

Dai, W.; Yang, Q.; Xue, G.-R.; and Yu, Y. 2007. Boosting for transfer learning. *ICML*, 193–200.

Daumé III, H., and Marcu, D. 2006. Domain adaptation for statistical classifiers. *J. Artificial Intelligence Research* 26:101–126.

Eaton, E.; desJardins, M.; and Lane, T. 2008. Modeling transfer relationships between learning tasks for improved inductive transfer. *ECML*, 317–332.

Freund, Y., and Schapire, R. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences* 55(1):119–139.

Gao, J.; Fan, W.; Jiang, J.; and Han, J. 2008. Knowledge transfer via multiple model local structure mapping. *SIGKDD*, 283–291.

Kaski, S., and Peltonen, J. 2007. Learning from relevant tasks only. *ECML*, 608–615.

Littlestone, N., and Warmuth, M. K. 1994. The weighted majority algorithm. *Information and Computation* 108(2):212–261.

Raina, R.; Ng, A. Y.; and Koller, D. 2006. Constructing informative priors using transfer learning. *ICML*, 713–720.

Rennie, J. 2003. 20 Newsgroups data set, sorted by date. Available online at http://www.ai.mit.edu/∼jrennie/20Newsgroups/.

Reyzin, L., and Schapire, R. 2006. How boosting the margin can also boost classifier complexity. *ICML*, 753–760.

Rosenstein, M.; Marx, Z.; Kaelbling, L.; and Dietterich, T. 2005. To transfer or not to transfer. In *NIPS Inductive Transfer Workshop*.

Schapire, R., and Singer, Y. 1998. Improved boosting algorithms using confidence-rated predictions. *COLT*, 80–91.

Shi, X.; Fan, W.; and Ren, J. 2008. Actively transfer domain knowledge. *ECML*, 342–357.

Witten, I. H., and Frank, E. 2000. *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann.

Wu, P., and Dietterich, T. G. 2004. Improving SVM accuracy by training on auxiliary data sources. *ICML*, 871–878.

Yao, Y., and Doretto, G. 2010. Boosting for transfer learning with multiple sources. *CVPR*, 1855–1862.