Improved Algorithms for Distributed Boosting

Jeff Cooper and **Lev Reyzin**
University of Illinois at Chicago
Chicago, IL 60607
{jcoop8,lreyzin}@uic.edu

Abstract

We introduce two distributed boosting algorithms. Our first algorithm uses the entire dataset to train a classifier and requires a significant amount of communication among the distributed sites. Our second algorithm requires very little communication but uses a subsample of the dataset to train the final classifier. Both of our algorithms improve upon existing distributed algorithms. Further, both are competitive with AdaBoost when it is run with the entire dataset.

Introduction

Both theoretical results and empirical studies indicate that machine learning algorithms can often greatly improve their performance by training on larger and larger datasets. In some situations, training on large datasets is not possible – acquiring labeled training data can be very expensive. Oftentimes, however, there is a plethora of labeled past data – for instance, stocks and their prices over time.

In the cases where data is overabundant, various issues arise. One would ideally like to use all the available data, but training an algorithm on all data can be too time consuming. Moreover, the data may not fit in the working memory of any one machine. Hence, a practical solution would be to distribute the data across several machines.

To reduce training time, one could also use simple classifiers, e.g. linear predictors, and this has proven effective in many applications – the problem of parallelizing the computation still remains interesting (Agarwal et al. 2011), but the theory is more straightforward. However, ideally, one should be able to use state-of-the-art classification methods such as boosting (Freund and Schapire 1997) or SVM (Cortes and Vapnik 1995), without sacrificing on running time or limiting the amount of training data.

We focus on boosting, a state-of-the-art ensemble algorithm for supervised learning (Caruana and Niculescu-Mizil 2006). One effective approach to large-scale ensemble algorithms is to develop distributed algorithms for the weak classifier used in the ensemble; for example, Panda, Herbach, Basu, and Bayardo (2009) and Tyree, Weinberger, and Agrawal (2011) developed distributed algorithms for the decision trees used in ensemble algorithms.

Our goal instead is to develop distributed boosting algorithms which can be used with arbitrary weak learners. Thus our aim is to match the accuracy of standard boosting algorithms when trained on the same datasets. We build on the work of Lazarevic and Obradovic (2001), who presented one of the earliest distributed boosting algorithms. Their algorithm, which improved on work of Fan, Stolfo, and Zhang (1999), performs as well as standard boosting algorithms when constructing small ensembles of classifiers. However, for larger ensembles, their algorithms does not match state-of-the-art boosting algorithms. We will illustrate that this is due to its tendency to over-fit the training data.

In this paper, we present two distributed boosting algorithms. Our first method resembles the method of Lazarevic and Obradovic (2001) but is less prone to overfitting. Our second method is based on subsampling the data and has the advantage that it requires very little communication between machines. Both methods outperform (Lazarevic and Obradovic 2001) on all of the datasets on which we compared them. Moreover, our algorithms match the performance of the well-known boosting algorithm AdaBoost when it is run with all of the available training data.

Previous work

Boosting

AdaBoost (Algorithm 1) is known to be one of the best off-the-shelf machine learning algorithms (Freund and Schapire 1996a; Quinlan 1996). The algorithm is an ensemble method that was originally developed to improve the performance of a single weak learning algorithm. At each iteration of the algorithm, a new weak learner is constructed so that it performs better on training data where the previous weak learner failed. AdaBoost accomplishes this by maintaining a weight distribution over the training data, which the weak learning algorithm can use to emphasize different training points. Unfortunately, this distribution is updated at each step of the algorithm, making the algorithm inherently sequential. It is thus not obvious how to deploy it in a distributed environment.

DistBoost

The Distributed Boosting Algorithm (Lazarevic and Obradovic 2001), which we refer to as DistBoost, is meant to be run on several machines which can communicate with each other. The data is partitioned across these ma-

Algorithm 1 `AdaBoost`

```
Given: $(x_1, y_1), \ldots, (x_n, y_n)$
where $x_i \in X$, $y_i \in Y = \{-1, +1\}$.
Initialize $D_1(i) = 1/m$.
for $t = 1, \ldots, T$ do
    Train base learner using distribution $D_t$.
    Get base classifier $h_t : X \to \{-1, +1\}$.
    Let $\gamma_t = \sum_i D_t(i) y_i h_t(x_i)$.
    Choose $\alpha_t = \frac{1}{2} \ln \frac{1+\gamma_t}{1-\gamma_t}$.
    Update:
        $$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t},$$
    where $Z_t$ normalizes so that $D_{t+1}$ is a distribution.
end for
Output the final classifier:
    $$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$
```

chines, and each machine stores a weight distribution for its own data. The main idea of the algorithm is to have the concatenation of these local distributions mimic the distribution of `AdaBoost` as if all of the data had been on one machine. A sketch of the algorithm is shown in Algorithm 2. For simplicity, we use the simple majority rule described in (Lazarevic and Obradovic 2001) for creating the ensemble E_t.

Despite its name, it is not hard to see that `DistBoost` is not a true boosting algorithm given a bad split of the data. Consider the set of labeled examples on the real line

$$X = \{(-1, 1), (0, -1), (1, 1)\}.$$

It is not hard to see that X is weakly learnable by decision stumps. However, suppose there are three machines, two of which contain both of the positive examples and one containing the negative example. The two machines with the positive examples will always output a decision stump which classifies -1 and 1 correctly and will therefore misclassify the negative example. Thus the majority vote classifier will never correctly classify the negative example, preventing the creation of a strong learner.

The point of this example is that the algorithm can overfit the training data at one site. More generally, consider a site j. The classifier $h_{t,j}$ is constructed based only on the slice of D_t corresponding to the data at site j. As t grows, this slice becomes increasingly specialized to site j. As a result, $h_{t,j}$ generalizes poorly to the distributions $D_t^{(l)}$, for $l \neq j$. Its contribution to the ensemble E_t thus becomes noise, so E_t fails to effectively reduce the error at any of the machines.

The effect of this overfitting on accuracy can be seen in experiments, where as the number of rounds of boosting grows, `DistBoost` begins to overfit and fails to keep up with `AdaBoost` (when run on the entire dataset) in decreasing the generalization error. This is illustrated in Figure 1, where `DistBoost` fails to match `AdaBoost`'s accuracy after only 25 rounds of boosting.

Algorithm 2 `DistBoost`

```
Given: $K$ machines, $(x_1, y_1), \ldots, (x_{Kn}, y_{Kn})$
where $x_i \in X$, $y_i \in Y = \{-1, +1\}$.
Initialize $D_1(i) = \frac{1}{Kn}$.
for $t = 1, \ldots, T$ do
    for $j = 1, \ldots, K$ (in parallel) do
        Train base learner using data at site $j$ and dist. $D_t$.
        Get base classifier $h_{t,j} : X \to \{-1, +1\}$.
    end for
    Let $E_t(x) = \text{sign}\left(\sum_{j=1}^{K} h_{t,j}(x)\right)$.
    Let $\gamma_t = \sum_i D_t(i) y_i E_t(x_i)$.
    Choose $\alpha_t = \frac{1}{2} \ln \frac{1+\gamma_t}{1-\gamma_t}$.
    Update:
        $$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i E_t(x_i))}{Z_t},$$
    where $Z_t$ normalizes so that $D_{t+1}$ is a distribution.
end for
Output the final classifier:
    $$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$
```

Another drawback of `DistBoost`, which our sampling algorithm addresses, is that the machines communicate with each other at each boosting round. To compute E_t, the weak learners at every site must be broadcast to every other site. Then the sites must each broadcast the local error of E_t so that γ_t can be computed. This reliance on communication among machines can be problematic when hundreds of machines are running and delays or failures become likely.

`Ivote` and `DIvote`

Like `AdaBoost`, `Ivote` (Breiman 1999a) is an ensemble algorithm which tries to focus on harder examples. Rather than maintain a weight distribution, `Ivote` focuses on hard examples at each iteration by sampling from the training data using a variation of bagging (Breiman 1996). Recall that bagging draws a set of samples with replacement and uses them for training. In `Ivote`, a point is drawn at random with replacement, and if the current ensemble correctly classifies it, it is added to the next set of training data. Otherwise, it is discarded with some probability, which is chosen so that roughly half of the sampled data is correctly classified by the current ensemble and half is incorrectly classified. The sampled training data is then used to build a new classifier, which is added to the ensemble. The ensemble uses majority vote to combine the classifiers.

`DIvote` is a distributed version of `Ivote` (Chawla et al. 2004). The data is first partitioned across machines. At each machine, an ensemble is built using `Ivote`, and the ensembles are then combined to create one large majority-vote ensemble. As each individual classifier is trained using roughly half correctly classified and half incorrectly classified examples (by the machine's current ensemble), the learners avoid

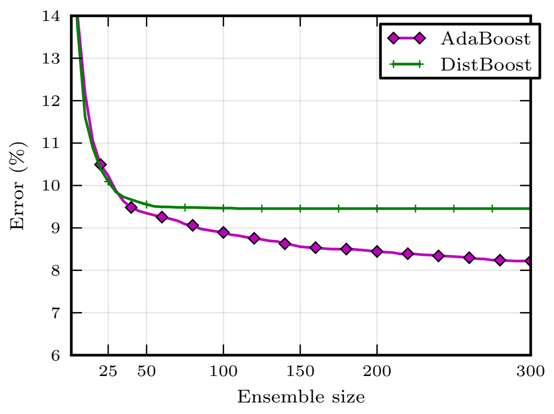

Figure 1: AdaBoost and DistBoost run on the UCI particle dataset.

overfitting on hard examples. However, the algorithm is not a boosting algorithm and is unable to drive down the training error as well as AdaBoost (the example from Section also works for DIvote). Our experiments show this results in a loss of accuracy compared to AdaBoost.

Our approach

We propose two algorithms for distributed boosting, PreWeak and AdaSampling. PreWeak addresses the overfitting problem of DistBoost but still requires a large amount of communication among sites. AdaSampling is a different approach, which tries to select the most difficult examples from each machine and use them to train one classifier. FilterBoost (Bradley and Schapire 2007) is a somewhat related approach, which samples a small amount of data from the training set at every round of boosting.

Avoiding overfitting

The main idea of PreWeak is to pre-build a large set of weak learners that can be tested on the global distribution. The algorithm has two stages. During the first stage, AdaBoost is run at each of K sites, resulting in a set of T weak learners at each site. These classifiers are then broadcast to each other so that each site has its own copy of the set $\mathcal{H} = \{h_1, \ldots, h_{TK}\}$ of all classifiers. Finally, each site j computes a $TK \times n$ error matrix E_j, where

$$E_j(i,m) = \begin{cases} 1, & h_i(x_m, y_m) \neq y_m \\ 0, & h_i(x_m, y_m) = y_m. \end{cases}$$

Stage two consists of a master server running a slight variant of AdaBoost for T rounds. The algorithm maintains a single distribution D_t over the set of all examples. Since this distribution may not fit in the main memory of one machine, PreWeak accomplishes this by having each site maintain the slice of the distribution corresponding to its training data (this is also how DistBoost maintains the global distribution). For each site j, we refer to this slice as $D_t^{(j)}$.

However, rather than using D_t to construct a classifier at each iteration, PreWeak uses D_t to select a classifier from

Algorithm 3 PreWeak Algorithm

```
Given: $K$ machines, $(x_1, y_1), \ldots, (x_{Kn}, y_{Kn})$
where $x_i \in X$, $y_i \in Y = \{-1, +1\}$.
for $j = 1, \ldots, K$ (in parallel) do
    Run AdaBoost for $T$ rounds using data at site $j$
    Save collection of weak learners $h_{j,1} \ldots, h_{j,T}$.
end for
Initialize $D_1(i) = \frac{1}{Kn}$.
for $t = 1, \ldots, T$ do
    Choose $h_t$ from collection
        $\{h_{j,i} : 1 \leq j \leq K, 1 \leq i \leq T\}$
    that minimizes error with respect to $D_t$.
    Let $\gamma_t = \sum_i D_t(i) y_i h_t(x_i)$.
    Choose $\alpha_t = \frac{1}{2} \ln \frac{1+\gamma_t}{1-\gamma_t}$.
    Update:
        $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$,
    where $Z_t$ normalizes so that $D_{t+1}$ is a distribution.
end for
Output the final classifier:
    $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

the set $\mathcal{H}$ that was constructed in the previous stage. More precisely, PreWeak selects the $h_i \in \mathcal{H}$ that minimizes

$$\sum_{j=1}^{K} \sum_{m=1}^{n} E_j(i,m) D_t^{(j)}(m),$$

which the master can compute after receiving the inner sum from each site. The master then tells each site j which classifier was selected so that the site may compute $D_{t+1}^{(j)}$. Note that the normalization factor Z_t can be computed with an additional round of communication with the master.

DistBoost and PreWeak differ primarily in how the classifier is selected at each round. DistBoost constructs a classifier at site j using the weak learning algorithm and the distribution $D_t^{(j)}$, so it has available to it a larger set of candidate classifiers than PreWeak. However, the classifier is evaluated based only on the local slice $D_t^{(j)}$ of the global distribution. As Section discusses, after a few rounds of boosting, this causes the constructed weak learner to overfit to its local training data and so it has trouble classifying higher weighted examples at other machines.

PreWeak is only able to choose one of TK weak learners, but it gets to evaluate all of these weak learners against the global distribution. Thus the chosen weak learner is less likely to overfit to the training data at any single site. Furthermore, since the candidate set of weak learners was constructed using AdaBoost, we expect that at each round at least one of the weak learners will perform well on the global distribution. We thus expect PreWeak to more successfully reduce the training error than DistBoost.

Figure 2 demonstrates DistBoost's tendency to overfit. After each iteration of DistBoost, we measured the error of the weak learner constructed at site i on the training data at all sites $j \neq i$ (using the current round's distribution). As Figure 2 shows, the weak learners did not perform significantly better than random guessing. On the other hand, PreWeak did not suffer from this problem. Figure 2 also shows the error of the weak learner selected by PreWeak when tested on the sites where it was not built. This error still approaches 50%, but it does so slowly enough that it is able to continue to decrease the test error.

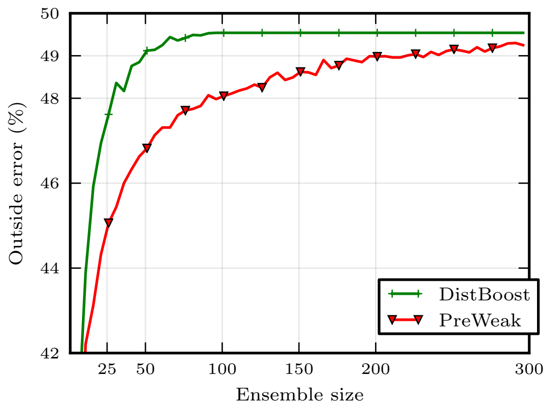

Figure 2: Global training error of weak learners selected by DistBoost and PreWeak on the UCI particle dataset.

Unfortunately, PreWeak requires slightly more communication among sites than DistBoost. Both algorithms broadcast a total of TK weak learners: PreWeak broadcasts these during stage one, while DistBoost broadcasts K weak learners during each of the T rounds of boosting. However, during each round of boosting, PreWeak requires that each site broadcast the error of the TK classifiers on its training data. Each site thus broadcasts T^2K messages that DistBoost avoids. Depending on the size of the weak learner, these additional messages may be negligible compared to the broadcasting of the weak learners.

An advantage of PreWeak is that the weak learner built at a particular site depends only on the data at that site. If PreWeak is later run on a larger dataset with additional machines, PreWeak can save time by reusing the weak learners from the old machines, whereas DistBoost would need to rebuild weak learners at every site.

Passing informative examples

While PreWeak works well in our experiments, it requires significant communication. We introduce a second algorithm, AdaSampling, which requires no communication between sites. In AdaSampling, a small number of examples are selected at each site and then sent to a master machine. This machine then runs AdaBoost with these examples to obtain the final classifier. Since the final classifier is not trained on all examples, AdaSampling must be careful in how each machine chooses its small set of examples.

Algorithm 4 AdaSampling Algorithm

```
Given: $K$ machines, $(x_1, y_1), \dots, (x_{Kn}, y_{Kn})$
where $x_i \in X$, $y_i \in Y = \{-1, +1\}$.
for $j = 1, \dots, K$ (in parallel) do
    Run AdaBoost for $T$ rounds using data at site $j$
    Sort examples by decreasing value of $\sum_{t=1}^{T} D_t^j(i)/t$
    Broadcast $n/K$ consecutive examples with lowest local test error
end for
Run AdaBoost with training set of the $n$ broadcasted examples.
Output classifier returned by AdaBoost
```

AdaSampling is based on a connection between AdaBoost and game theory. Consider a two-person game, where the row player is given a set $\mathcal{H}$ of weak learners, and the column player is given a set X of training data. In each round of the game, the row player picks a weak learner $h \in \mathcal{H}$, while the column player picks a training example $(x, y) \in X$. The row player receives a payout of 1 if $h(x) = y$ and 0 if $h(x) \neq y$.

The solution to this game is a pair of probability distributions (P^*, Q^*), where $P^* \in \mathbb{R}^{|\mathcal{H}|}$ and $Q^* \in \mathbb{R}^{|X|}$, which tells each player how to randomly select its choice at each round. Since the column player is motivated to choose examples which are likely to be misclassified, we would intuitively expect their corresponding weights in Q^* to be higher than the weights of examples which are easier to classify.

Recall that AdaBoost maintains a distribution D_t over its training set at each iteration of the algorithm. When a constant value of γ_t is used in AdaBoost, (Freund and Schapire 1996b) showed that the average of the distributions D_t converges (as $t \to \infty$) to the strategy Q^*. We may thus view the examples with highest average weight as the hardest to classify.

AdaSampling uses this view to select examples on which to train a final classifier. Consider one of the K sites. Its goal is to select n/K examples to send to the final classifier for training. The site runs AdaBoost for T iterations. It then sorts the examples in decreasing order of $\sum_{t=1}^{T} D_t(i)$, giving us a list $(x_1, y_1), \dots, (x_n, y_n)$ of training examples at the site. We know that the top examples in this list can be viewed as the hardest examples, so we might then be tempted to select examples $(x_1, y_1), \dots, (x_{n/K}, y_{n/K})$. However, we only know that they are hard to classify, not that they are particularly helpful in training a classifier.

We instead select the n/K consecutive examples which provide the most accurate AdaBoost classifier at the local machine. To find them, for each i, we train a classifier with examples $(x_{i+1}, y_{i+1}), \dots, (x_{i+n/K}, y_{i+n/K})$ and compute the test error on the remaining examples. We then send the n/K examples with the lowest test error. These examples may not form the most effective training set at the local machine, but our sorting scheme allows us to select a training set from only $n - n/K$ candidate sets rather than all possible training sets. In our experiments, we further reduced the number of candidate training sets by incrementing i by a step size of $n/4K$ and stopping when i reached $n/2$. Thus we only trained an additional $2K$ classifiers, each with a relatively small training set of size n/K.

Note that our sampling scheme is reminiscent of Karmaker and Kwek's (2006) method of using AdaBoost to ignore noisy examples.

Experiments

We compare four distributed algorithms: DistBoost, DIvote, PreWeak, and AdaSampling. In all of the algorithms, we assume that the training data is distributed across 10 sites. We also compare our results to AdaBoost when trained with the full training set and AdaBoost when trained with 1/10 of the training set (1/10). This is meant to simulate AdaBoost running on a single site where it would not be able to take advantage of additional training data.

Datasets

We first experiment on six two-class datasets: ocr17, ocr49, forestcover12, particle, ringnorm, and twonorm. None of these datasets approach terascale sizes (Agarwal et al. 2011), but are comparable in size to those of Lazarevic and Obradovic (2001). In addition, we experimented on a larger dataset provided by Yahoo!

Dataset	Training	Testing	Features
ocr17	5000	1000	196
ocr49	5000	1000	196
forestcover12	250,000	245,141	54
particle	80,000	50,064	50
ringnorm	50,000	50,000	21
twonorm	50,000	50,000	21
Yahoo!	1,500,000	790,224	10

Table 1: Datasets used in experiments

The training size, test size, and number of attributes for each dataset are shown in Table 1. All of the datasets' features are scaled to be integer-valued. Datasets ocr17 and ocr49 are a subset of the NIST database and consist of handwritten images of 1s and 7s and of 4s and 9s, respectively. Each image is represented by 196 integer intensity values in the range $[0, 3]$. We experiment on these small datasets to see how robust our algorithms were to small datasets.

The forestcover12 dataset consists of classes 1 and 2 from the 7-class Covertype dataset in the UCI repository. The particle dataset, from the UCI repository, consists of two classes, signal and background. This data is from the MiniBooNE experiment and is used to distinguish electron neutrinos (signal) from muon neutrions (background). Ringnorm and twonorm are both synthetic datasets, which we generated using the twonormgen and ringnormgen scripts made available by Breiman (1999b).

The Yahoo! dataset (Yahoo! 2014) contains anonymized user click logs for news articles displayed on Yahoo! The dataset contains 10 features per click, and the label determines whether or not the user clicked the displayed article. We trimmed the dataset to contain 1,042,974 positive examples (clicks) and 1,237,251 negative examples (non-clicks).

Results

Our results on the Yahoo! dataset are averaged over three experiments. The results on the remaining datasets are averaged over 15 experiments. In each experiment, we randomly partition the data into training and testing sets. We compare both ensembles of decision stumps and of depth 3 decision trees with no pruning, with each algorithm performing 300 boosting rounds, a number large enough to elucidate the trends in the error rates. DIvote built a 300 tree ensemble at each site, resulting in a total ensemble of 3000 trees.

Graphs comparing error rate to the number of trees in each algorithm's ensemble are shown in Figure 3 and Figure 4. In every case, DistBoost stops boosting after a small number of rounds, while PreWeak boosts at the same rate as AdaBoost. These results support our conclusion that DistBoost's weak learners overfit the training data at each site, while PreWeak is able to continually drive down the training error. We noticed that on the ocr17 and ocr49 datasets, PreWeak and AdaBoost both had 0% training error when using depth three trees.

The results with decision stumps are shown in Table 2. PreWeak outperformed AdaBoost on every dataset except ringnorm. On ringnorm, PreWeak was still competitive with AdaBoost and drastically outperformed both DistBoost and DIvote. AdaSampling was competitive with AdaBoost on every dataset except ocr49 and ringnorm. Further, it outperformed DIvote and DistBoost on every dataset.

Table 3 shows the results with depth three decision trees. Except for twonorm, the classifiers are all more accurate with depth 3 trees. On twonorm, AdaBoost, PreWeak, and AdaSampling all performed worse with depth 3 trees. Further, PreWeak and AdaSampling (and DistBoost) lost to 1/10, the version of AdaBoost trained using 1/10 of the data. PreWeak and AdaSampling also lost to DIvote on ringnorm, the other synthetic dataset. However, PreWeak and AdaSampling were more accurate than DIvote on the remaining four datasets. In addition, AdaSampling with decision stumps was just as accurate as DIvote with depth three trees.

Discussion

We presented two new algorithms for distributed boosting. Both of our algorithms are competitive with AdaBoost when it is trained with the entire dataset, and both algorithms outperform DistBoost on every dataset on which we experimented. Further, PreWeak was able to boost its accuracy at the same rate as AdaBoost.

Like DIvote, AdaSampling requires no communication between sites yet outperformed it on several datasets. AdaSampling, however, was substantially worse than AdaBoost on two of the datasets. It remains open to create a boosting algorithm that is competitive in accuracy to AdaBoost on all six datasets yet requires as little communication as DIvote.

	AdaBoost	DistBoost	DIvote	PreWeak	AdaSampling	1/10
ocr17	.69 ± .23	2.14 ± .61	1.76 ± .38	**.63** ± .23	.68 ± .25	1.41 ± .28
ocr49	5.17 ± .66	8.60 ± 1.31	6.21 ± .89	**4.55** ± .62	5.93 ± .86	6.94 ± .58
forestcover12	22.66 ± .11	23.15 ± .40	25.61 ± .06	22.46 ± .09	**22.20** ± .11	22.73 ± .15
particle	8.22 ± .10	9.46 ± .25	11.16 ± .17	**7.98** ± .10	8.08 ± .07	8.57 ± .12
ringnorm	**2.53** ± .07	46.9 ± .89	27.1 ± 10.8	2.69 ± .08	4.59 ± .25	2.80 ± .10
twonorm	2.89 ± .07	4.86 ± .61	2.74 ± .07	2.86 ± .06	**2.58** ± .06	3.06 ± .07
Yahoo!	36.57 ± .05	37.56 ± .39	40.31 ± .05	**36.50** ± .27	37.20 ± .06	36.89 ± .21

Table 2: Test error (%) with 300 decision stumps

	AdaBoost	DistBoost	DIvote	PreWeak	AdaSampling	1/10
ocr17	**.39** ± .16	2.16 ± .49	.94 ± .29	.41 ± .16	.40 ± .18	.98 ± .52
ocr49	**1.61** ± .36	3.97 ± .72	2.50 ± .52	1.76 ± .42	2.79 ± .47	3.10 ± .30
forestcover12	20.86 ± .44	20.96 ± .66	22.13 ± .06	**18.72** ± .24	19.55 ± .17	20.99 ± .46
particle	6.75 ± .09	8.39 ± .18	7.35 ± .08	7.09 ± .09	**6.25** ± .13	7.21 ± .09
ringnorm	**1.81** ± .05	5.97 ± .99	2.06 ± .05	2.13 ± .07	2.15 ± .06	2.25 ± .07
twonorm	3.09 ± .09	4.10 ± .15	**2.58** ± .05	3.28 ± .08	3.14 ± .06	3.08 ± .07
Yahoo!	**34.99** ± .08	35.61 ± .10	37.40 ± .02	35.12 ± .01	35.31 ± .04	35.56 ± .03

Table 3: Test error (%) using 300 depth three decision trees

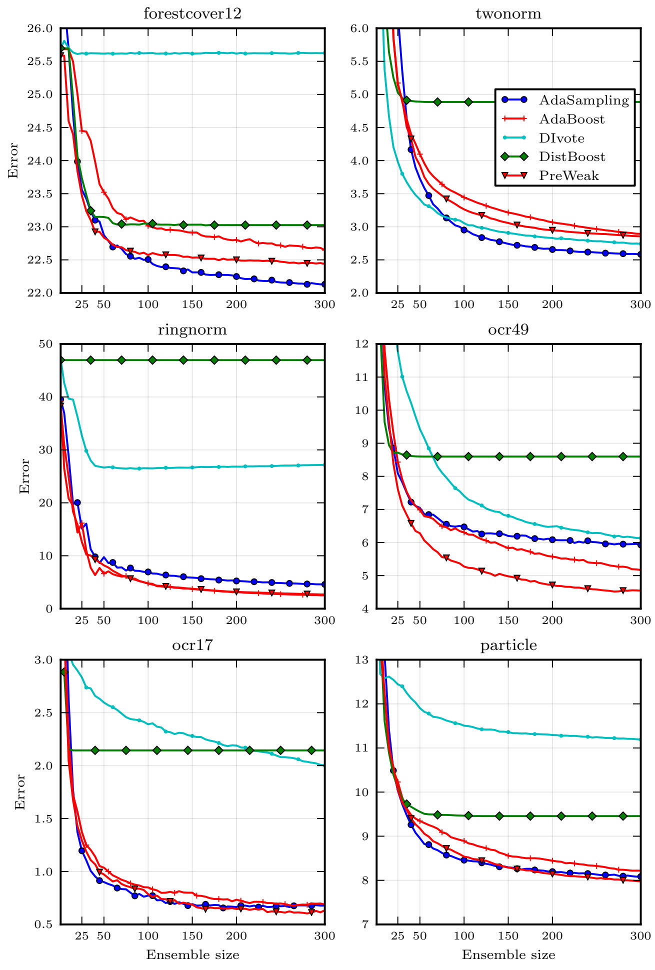

Figure 3: Test error (%) with ensembles of decision stumps

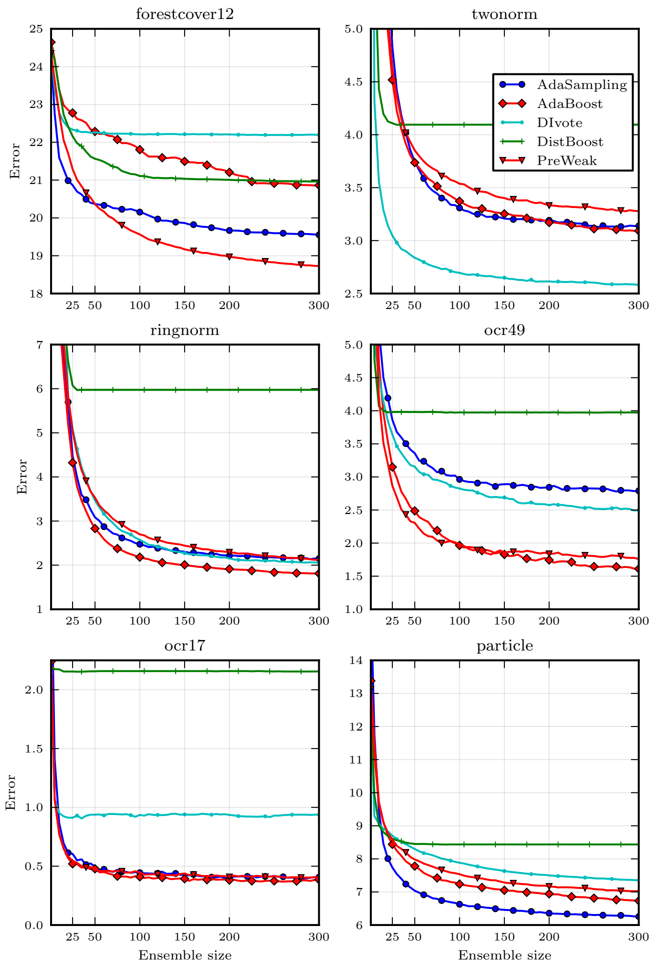

Figure 4: Test error (%) with ensembles of depth three decision trees

References

Agarwal, A.; Chapelle, O.; Dudík, M.; and Langford, J. 2011. A reliable effective terascale linear learning system. *CoRR* abs/1110.4198.

Bradley, J. K., and Schapire, R. E. 2007. FilterBoost: Regression and classification on large datasets. In *Advances in Neural Information Processing Systems*, volume 20.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Breiman, L. 1999a. Pasting small votes for classification in large databases and on-line. *Machine Learning* 36(1-2):85–103.

Breiman, L. 1999b. Prediction games and arcing algorithms. *Neural Computation* 11(7):1493–1517.

Caruana, R., and Niculescu-Mizil, A. 2006. An empirical comparison of supervised learning algorithms. In Cohen, W. W., and Moore, A., eds., *ICML*, volume 148 of *ACM International Conference Proceeding Series*, 161–168. ACM.

Chawla, N. V.; Hall, L. O.; Bowyer, K. W.; and Kegelmeyer, W. P. 2004. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research* 5:421–451.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

Fan, W.; Stolfo, S. J.; and Zhang, J. 1999. The application of adaboost for distributed, scalable and online learning. In *Pages 362366 of: SIGKDD Conference on Knowledge and Data Mining (KDD*.

Freund, Y., and Schapire, R. E. 1996a. Experiments with a new boosting algorithm. In Saitta, L., ed., *ICML*, 148–156. Morgan Kaufmann.

Freund, Y., and Schapire, R. E. 1996b. Game theory, on-line prediction and boosting. In *In Proceedings of the Ninth Annual Conference on Computational Learning Theory*, 325–332. ACM Press.

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

Karmaker, A., and Kwek, S. 2006. A boosting approach to remove class label noise. *Int. J. Hybrid Intell. Syst.* 3(3):169–177.

Lazarevic, A., and Obradovic, Z. 2001. The distributed boosting algorithm. In Lee, D.; Schkolnick, M.; Provost, F. J.; and Srikant, R., eds., *KDD*, 311–316. ACM.

Panda, B.; Herbach, J. S.; Basu, S.; and Bayardo, R. J. 2009. Planet: Massively parallel learning of tree ensembles with mapreduce. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009).*

Quinlan, J. R. 1996. Bagging, boosting, and c4.5. In Clancey, W. J., and Weld, D. S., eds., *AAAI/IAAI, Vol. 1*, 725–730. AAAI Press / The MIT Press.

Tyree, S.; Weinberger, K. Q.; and Agrawal, K. 2011. Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th International World Wide Web Conference (WWW-2011).*

Yahoo! 2014. Webscope dataset ydata-frontpage-todaymodule-clicks-v1_0. http://labs.yahoo.com/Academic_Relations.