

# The Application of AdaBoost for Distributed, Scalable and On-line Learning

Wei Fan, Salvatore J. Stolfo and Junxin Zhang

Department of Computer Science, Columbia University  
1214 Amsterdam Avenue Room 450, New York, NY 10027-7003, USA

## Abstract

We propose to use AdaBoost to efficiently learn classifiers over very large and possibly distributed data sets that cannot fit into main memory, as well as on-line learning where new data become available periodically. We propose two new ways to apply AdaBoost. The first allows the use of a small sample of the weighted training set to compute a weak hypothesis. The second approach involves using AdaBoost as a means to re-weight classifiers in an ensemble, and thus to reuse previously computed classifiers along with new classifier computed on a new increment of data. These two techniques of using AdaBoost provide scalable, distributed and on-line learning. We discuss these methods and their implementation in JAM, an agent-based learning system. Empirical studies on four real world and artificial data sets have shown results that are either comparable to or better than learning classifiers over the complete training set and, in some cases, are comparable to boosting on the complete data set. However, our algorithms use much smaller samples of the training set and require much less memory.

**Keywords:** *Boosting, Distributed, Scalable, On-line*

**Email address of contact author:** wfan@cs.columbia.edu

**Phone number of contact author:** ++ 1 212 939 7078

**Paper ID:** 412

# 1 Introduction

Learning from very large and distributed databases imposes major performance challenges for data mining. Many databases have grown too large to fit into main memory. Learning a monolithic classifier will be very slow since it requires all data be held in main memory. Some databases are inherently distributed and cannot be brought into a single site for a variety of reasons. It is very hard to learn a single classifier under these constraints. One such application is to learn a fraud detection model from different participating credit card companies whose datasets are huge and strictly distributed. Yet there are some databases that have large amounts of new data available periodically or in real time. To re-learn a new model of the whole data set each time an increment is provided is both expensive and inefficient.

Some researchers have proposed incremental learning techniques to solve these problems. This method typically involves direct modification to standard algorithms, such as decision tree learning (Utgoff [23]) and rule learner (Clearwater, Cheng, Hirsh and Buchanan [7]). Other researchers have proposed methods to parallelize learning algorithms, for example, parallel rule induction (Provost and Hennesey [18]), decision tree constructor (Shafer, Agrawal and Mehta [21] and Breiman and Spector [4]) and association rules (Agrawal, Mehta, Shafer and Srikant [1], and Han, Karypis and Kumar [14]). An alternative and general method is to combine multiple classifiers in a “blackbox” manner. In “meta-learning” [5], Chan proposed different methods to compose the predictions of classifiers trained from different partitions of the training set. A JAVA-based agent system implementing meta-learning is available (Stolfo, et al [22]). Breiman [2] has recently proposed a statistics-based “paste vote” method in which he uses simple voting to combine classifiers trained from an “archive” of the original training set. The advantage of combining is that it is algorithm independent, and can be used to scale up many learning algorithms. In this paper, we will study a new technique using AdaBoost [12] to combine classifiers to provide *scalable*, *distributed* and *on-line* inductive learning.

Freund and Schapire’s AdaBoost [12] learns a highly accurate weighted voting ensemble of many “weak” hypotheses whose accuracy is only moderate. Typically, each hypothesis

outputs both a prediction and a confidence for its prediction. Each hypothesis is trained on the “same” data set yet with a different distribution. Different hypotheses are produced in different rounds of boosting. At each round, AdaBoost increases the weight of a wrongly classified training instance and decreases that of a correctly predicted instance. AdaBoost has received extensive theoretical and empirical study [12, 15, 19, 20, 16, 10]. Freund and Schapire (alternatively Schapire, et al [20]) have shown an upper bound on AdaBoost’s generalization error. Breiman [3], Friedman, Hastie and Tibshirani [13] explain boosting from a statistical analysis. Margineantu and Dietterich [16] have studied methods to prune the voted ensemble to increase classification throughput. An empirical study among AdaBoost, Bagging and randomization by Dietterich [10] has shown that AdaBoost increases accuracy more than the other two methods.

There are two reasons that we are interested in applying AdaBoost for scalable, distributed and on-line learning. First, it is a general combining method that can be applied to a wide variety of algorithms and methodologies. Second, the advantage of AdaBoost over other combining techniques is its theoretical foundation and empirically demonstrated utility. Additionally, the strong hypothesis calculated by AdaBoost is a weighted voting ensemble that can freely scale the magnitude of a weak hypothesis by its given weight. This property is desirable since giving a large weight to a stronger hypothesis can correct the wrong classifications of many weaker hypotheses. Simple voting doesn’t have this nice property.

Most prior work on AdaBoost focuses on improving the accuracy of a weak classifier on the same single chunk of data at a central site that is small enough to fit into main memory. There hasn’t been much research on using AdaBoost for scalable, distributed or on-line learning. We study these issues by giving different interpretations of AdaBoost and its weight updating rule for various scenarios. We propose a few applications of AdaBoost for learning under these situations. The major difference of our work from previous research is that each weak classifier is not trained from the same data set at each round but only a small portion of the training set. In one method, the weak classifiers are trained from either random samples or disjoint partitions of the data set, making AdaBoost feasible for scalable or distributed systems. Yet another method for on-line learning reuses old classifiers by re-weighting them using AdaBoost’s weight updating rule. We will show that our methods

can be easily implemented in an agent-based learning system such as JAM [22]. We evaluate the effectiveness of our methods against both publicly available and real world data sets. The surprising results are that even though each weak classifier is trained from only small portions of the training set, the method gives accuracy that is comparable to or better than a global classifier learned over all the data, and sometimes comparable to running AdaBoost using the whole data set at each round. However, the cost to produce a weak classifier from small portions of the data set is very low which makes the method computationally practical for very large databases. And each sample can fit into main memory, reducing memory management overhead. Another interesting finding is that the result is not sensitive to the size of the small partition. This means that in choosing this small portion in practice, we only need to consider memory constraints.

The remainder of this paper is organized as follows. First, we describe the AdaBoost algorithm, and introduce two new ways of using AdaBoost for scalable, distributed and on-line learning. The paper follows with a discussion of the results of the experiment. We conclude with discussions of an implementation of the algorithms in an agent-based architecture and several possible avenues of future research.

## 2 AdaBoost Algorithm

We follow the generalized analysis of AdaBoost by Schapire and Singer [19]. The algorithm is shown in Figure 1. It maintains a distribution  $D_t$  over the training examples. This distribution can be initially uniform. The algorithm proceeds in a series of  $T$  rounds. In each round, the entire weighted training set is given to the weak learner to compute weak hypothesis  $h_t$ . The distribution is updated to give wrong classifications higher weights than correct classifications. The classifier weight  $\alpha_t$  is chosen to minimize a proven upper bound of training error,  $\prod Z_t$ . In summary,  $\alpha_t$  is the root of the following equation:

$$Z'_t = - \sum_i^m D_t(i) u_i e^{-\alpha_t u_i} = 0 \quad \text{where} \quad u_i = y_i h_t(x_i). \quad (1)$$

An approximation method for  $0 \leq |h(x)| \leq 1$  gives a choice of

$$\alpha_t = \frac{1}{2} \ln \frac{1+r}{1-r} \quad \text{where} \quad r = \sum_i^m D_t(i) u_i. \quad (2)$$

- Given:  $\mathcal{S} = \{(x_1, y_1), \dots, (x_m, y_m)\}; x_i \in \mathcal{X}, y_i \in \{-1, +1\}$
- Initialize  $D_1(i)$  (such as  $D_1(i) = \frac{1}{m}$ )
- For  $t = 1, \dots, T$ :
  1. Train weak learner using distribution  $D_t$ .
  2. Compute weak hypothesis  $h_t : \mathcal{X} \rightarrow \mathbb{R}$ .
  3. Choose  $\alpha_t \in \mathbb{R}$ .
  4. Update

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where  $Z_t$  is a normalization factor chosen so that  $D_{t+1}$  will be a distribution.

- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \quad \text{where} \quad f(x) = \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

Figure 1: AdaBoost

In practice, we can use the estimation method to supply a candidate for numerical methods to solve the equation.

### 3 Scalable and Distributed Learning

In AdaBoost, the weak learner is treated as a “blackbox”. We don’t have much control over it except for observing its accuracy and providing it with a different training sample at each round according to its accuracy in previous rounds. Each weak learner may freely choose examples in the sample given to it. Conversely, we can also give the weak learner a sample which is just a small “portion” of the complete training set. This is an important observation that gives us the opportunity for scalable and distributed learning. The weak classifier produced from the small portion of the training set is very likely “weaker” than one generated from the entire weighted training set, but the overall accuracy of the voted ensemble can still be boosted. The interesting questions to answer are: how do we generate each portion of the training set to provide to the weak learner and how do we choose  $\alpha_t$  and update the distribution  $D_t$ .

In order to increase accuracy, the small portion should not be a highly skewed sample of the complete training set. We propose two methods. In  $r$ -sampling (Figure 2,  $r$  is random), a fixed number ( $n$ ) of examples are randomly picked from the weighted training set (without

- Given: sample size  $n$
- Initialize  $D_1(i)$  (such as  $D_1(i) = \frac{1}{m}$ )
- For  $t = 1, \dots, T$ :
  1. Randomly choose  $n$  examples from  $\mathcal{S}$  without replacement.
  2. Re-normalize their weights into a distribution,  $D'_t$ .
  3. Train weak learner using distribution  $D'_t$  for the chosen  $n$  examples.
  4. Compute weak hypothesis  $h_t : \mathcal{X} \rightarrow \mathbb{R}$ .
  5. Update

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}$$

for the complete training set  $\mathcal{S}$ .

- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \quad \text{where} \quad f(x) = \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

Figure 2:  $r$ -sampling AdaBoost

replacement) together with their assigned weights. All examples have equal chance of being selected, and  $D_t$  is not taken into account in this selection. At different rounds, a new  $r$ -sample is picked. In  $d$ -sampling (Figure 3,  $d$  is disjoint), the weighted training set is partitioned into  $p$  disjoint subsets. Each subset is a  $d$ -sample. At each round, a different  $d$ -sample is given to the weak learner. The weights of the chosen examples of both  $r$ - and  $d$ -sampling are re-normalized to make them a distribution,  $D'_t$ . A weak classifier is generated from these examples with distribution  $D'_t$ . The update of weights and calculation of  $\alpha_t$  are performed on the entire data set with distribution  $D_t$ . Both methods can be used for learning over very large data sets, but  $d$ -sampling is more suitable for distributed learning where data at each site cannot be culled together to a single site. The data at each site are taken as a  $d$ -sample. Weak learning at each round is carried out independently of all the local data of each site. On the other hand,  $r$ -sampling would choose data from different sites and transfer that to a single site. This is not possible for some applications and inefficient in general.

When all data are available at a local site, both the distribution  $D_t$  and classifier weight  $\alpha_t$  are calculated exactly as in AdaBoost. We use all the locally available training data in the computation. However, for distributed learning over many sites, in order to calculate  $\alpha_t$ , it requires either the predictions by the weak classifiers at all sites be brought to a single site or possibly several distributed computations of  $Z'_t$  (for a candidate of  $\alpha_t$ ) among different sites

- Given: partition number  $p$
- Initialize  $D_1(i)$  (such as  $D_1(i) = \frac{1}{m}$ )
- Divide  $\mathcal{S}$  into  $p$  partitions  $\{\mathcal{S}_0, \dots, \mathcal{S}_{p-1}\}$ .
- For  $t = 1, \dots, T$ :
  1. Re-normalize the weight of partition  $\mathcal{S}_{t \bmod p}$  to make it a distribution,  $D'_t$ .
  2. Train weak learner using distribution  $D'_t$ .
  3. Compute weak hypothesis  $h_t : \mathcal{X} \rightarrow \mathbb{R}$ .
  4. Update

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}$$

for the complete training set  $\mathcal{S}$ .

- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \quad \text{where} \quad f(x) = \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

Figure 3:  $d$ -sampling AdaBoost

(details are discussed in Section 6). We use the second approach since the load on network traffic is much less.

The extra storage to run both  $r$ -sampling and  $d$ -sampling is a floating point vector  $D_t(i) (i : 1, \dots, m)$  to record the weight of each training example. The size of the vector is linear in the size of the training set, fixed before run time and can be pre-allocated. The memory required to hold this vector is usually much smaller than that of the data set. If main memory is unavailable, this vector can be disk resident. This is still efficient since the access patterns to  $D_t$  are strictly sequential to both update weights and calculate  $\alpha_t$ .

The extra computing power is to label the entire data set at each round of boosting, update weights and calculate  $\alpha_t$ . All these can be done efficiently in main memory. Predicting the data set is sequential and linear in the size of the data set. In network implementations, the extra network traffic to send partial values of  $Z'_t$  (Section 6) is nominal.

## 4 On-line Learning

In on-line learning, there is a steady flow of new instances generated periodically or in real-time that ought to be incorporated to correct or update the model previously learned. We propose an on-line learning scheme using AdaBoost as a means to re-weight classifiers in

- Given:  $\{h_1, \dots, h_{T-1}\}$  and  $\mathcal{S}_T$  ( $|\mathcal{S}_T| = m_T$ )
- Initialize  $D_1(i)$  (such as  $D_1(i) = \frac{1}{m_T}$ )
- For  $t = 1, \dots, T - 1$ :
  1. Choose  $\alpha_t \in \mathbb{R}$ .
  2. Update

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}$$

for the new increment  $\mathcal{S}_T$ .

- Train weak learner using distribution  $D_T$ .
- Compute weak hypothesis  $h_T : \mathcal{X} \rightarrow \mathbb{R}$ .
- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \quad \text{where} \quad f(x) = \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

Figure 4: On-line AdaBoost

an ensemble, and thus to reuse previously computed classifiers along with new classifier computed on a new increment of data.

The basic idea is to reuse previous weak classifiers  $\{h_1, \dots, h_{T-1}\}$  and learn a new classifier from the weighted new data (Figure 4). When the increment  $\mathcal{S}_T$  arrives, we use the weight updating rule to both “re-weight” previous classifiers based on their accuracy on  $\mathcal{S}_T$  and generate a weighted training set. Data items that are not accurately predicted by these hypotheses receive higher weights. A new classifier  $h_T$  is trained from this weighted increment. At the end of this procedure, we have  $\alpha_1, \dots, \alpha_{T-1}$  for  $h_1, \dots, h_{T-1}$ , a new classifier  $h_T$  and its weight  $\alpha_T$  calculated from the weighted training set  $\mathcal{S}_T$  during successive updates of the distribution. The interesting observation is that AdaBoost provides a means of assigning weights to classifiers based on a validation set that these classifiers are not necessarily trained from. It also identifies data items that these classifiers perform poorly on and generates a weighted increment accordingly.

One problem with the above on-line learning method is that we are forced to retain all the classifiers previously learned. This naturally increases our memory requirements and slows the learning and classifying procedures. We may use Margineantu and Dietterich’s pruning method [16] to prune the voted ensemble to speed up classification. We propose to use a “window” of a fixed number of classifiers to solve this problem. Instead of retaining all

classifiers, we only use and store the  $k$  most recent classifiers that reflect the most recent on-line data. The value of  $k$  can either be fixed or change according to the amount of resource and accuracy requirement. The number of new data items accumulated on-line before on-line learning starts can either be fixed or change at runtime as well.

This on-line learning scheme with a window size of  $k$  is efficient. The extra overhead involves  $k$  classifications of the increment with weight updating and  $k + 1$  computations of  $\alpha$ . The predicting and weight updating procedure is linear in the size of the data increment. The computation of  $\alpha$  has a bounded number of linear computations (to calculate  $Z'$ ).

## 5 Experiment

We evaluate  $r$ - and  $d$ -sampling from a number of perspectives. We compare the error rate of  $r$ - and  $d$ -sampling to the error rate over learning and boosted learning from the entire data set, and calculate the change in error rate over each boosting round and the amount of computation. The sensitivity of error rate to the sample size is crucial since we are constrained to fit the small sample into main memory. Another interesting comparison is to see if we can obtain the same level of error by simply AdaBoost on the same randomly chosen small sample with equal size and number of rounds as used by  $d$ - and  $r$ - sampling.

For on-line learning, it is interesting to know if the on-line algorithm actually performs better than a simple classifier learned over the new increment or the global classifier learned over all the training data including the new increment. It is also important to know the sensitivity of the on-line algorithm to the sample size.

### 5.1 Experimental Set up

Four data sets (summarized in Table 1) are used in this study. ADULT is a population census data downloaded from the UCI machine learning database. WHIRL is an information extraction data set [9] that trains wrappers to correctly extract relevant information from a web page. BOOLEAN is an artificial boolean data set that has 15 boolean variables. (The data item is positive if 4 or 5 variables are true or otherwise negative.) CHASE is a credit card fraud detection data set, courtesy of Chase Bank, provided for our research in fraud

| Data Set | Continuous Feature | Discrete Feature | Class | Training Size | Testing Size |
|----------|--------------------|------------------|-------|---------------|--------------|
| ADULT    | 6                  | 8                | 2     | 32561         | 16281        |
| BOOLEAN  | 0                  | 15               | 2     | 25000         | 7768         |
| WHIRL    | 14                 | 6                | 2     | 50407         | 12602        |
| CHASE    | 14                 | 6                | 2     | 40841         | 41439        |

Table 1: Data Set Summary

and intrusion detection.

The sample sizes for both  $d$ - and  $r$ - sampling are chosen to range from  $\frac{1}{s} = \frac{1}{2}, \frac{1}{4}$  to  $\frac{1}{256}$  of the original training set and the rounds of boosting range over 4, 8 to 512. The significant change (from  $\frac{1}{2}$  to  $\frac{1}{256}$ ) in the sample size will show the sensitivity of the algorithms to the sample size. When the data set size is 50000,  $\frac{1}{256}$  of this amount is only around 200 training items. The bigger sample sizes such as  $\frac{1}{2}$  and  $\frac{1}{4}$  will help us evaluate  $d$ -sampling for distributed learning where, in reality, there are a limited number of data sites with the same data schema.

We don't have any real on-line data set on hand. Instead, we have used a simulated on-line flow of data similar to  $d$ -sampling. Each  $d$ -sample is taken as an increment. The size of the window  $k$  is arbitrarily selected to be 10 and the size of the increment is chosen from  $\frac{1}{s} = \frac{1}{32}$  to  $\frac{1}{256}$  of the entire training set. At the start of the flow of on-line data, we don't have  $k$  classifiers. In this case, we reuse all available classifiers.

We used Cohen's RIPPER [8] as the "weak" learner. RIPPER provides an easy way to change the distribution of the training set (and it is publicly available). Since using the training set alone usually over-estimates the accuracy of a rule set, we used the Laplace estimate to generate the confidence for each rule. We carefully engineered the bisection search algorithm with the first candidate  $\alpha$  calculated by formula (2). The second point was searched with an exponential step increment. To avoid numerical errors introduced when adding a small number to a much bigger number, we used a vector  $v$  to compute  $Z'(\alpha) = -\sum_i D(i)u_i e^{-\alpha u_i}$ . For simplicity, we allow the index of  $v$  to be negative. The  $l$ -th element of the vector sums  $w_i = -D(i)u_i e^{-\alpha u_i}$ , if  $10^{l-1} \leq |w_i| \leq 10^l$ . We add every element in the vector starting from the smallest index (to avoid addition errors) to calculate  $Z'$ .

|               | ADULT      |              |                |              |              | BOOLEAN    |              |                |             |             |
|---------------|------------|--------------|----------------|--------------|--------------|------------|--------------|----------------|-------------|-------------|
| $\frac{1}{s}$ | Base Error | Boosted Base | Boosted Sample | $d$          | $r$          | Base Error | Boosted Base | Boosted Sample | $d$         | $r$         |
| 2             | 14.8       | 14.7         | 14.8           | <b>14.7</b>  | <b>14.7</b>  | 10.7       | 5.98         | <b>8.55</b>    | <b>8.47</b> | <b>8.26</b> |
| 4             | 14.8       | 14.7         | 15.5           | 14.8         | <b>14.7</b>  | 10.7       | 5.98         | <b>7.67</b>    | <b>7.47</b> | <b>6.84</b> |
| 8             | 14.8       | 14.7         | 14.9           | <b>14.7</b>  | <b>14.6</b>  | 10.7       | 5.98         | <b>7.72</b>    | <b>6.49</b> | <b>7.07</b> |
| 16            | 14.8       | 14.7         | 15.4           | <b>14.7</b>  | <b>14.6</b>  | 10.7       | 5.98         | <b>9.14</b>    | <b>7.05</b> | <b>7.49</b> |
| 32            | 14.8       | 14.7         | 16.6           | 14.8         | 14.8         | 10.7       | 5.98         | <b>10.0</b>    | <b>6.68</b> | <b>7.89</b> |
| 64            | 14.8       | 14.7         | 18.6           | 14.8         | 14.9         | 10.7       | 5.98         | <b>10.4</b>    | <b>7.60</b> | <b>8.37</b> |
| 128           | 14.8       | 14.7         | 19.2           | 14.8         | 14.8         | 10.7       | 5.98         | 11.2           | <b>7.58</b> | <b>8.07</b> |
| 256           | 14.8       | 14.7         | 20.8           | 14.9         | 15.1         | 10.7       | 5.98         | 11.1           | <b>7.26</b> | <b>7.83</b> |
|               | WHIRL      |              |                |              |              | CHASE      |              |                |             |             |
| $\frac{1}{s}$ | Base Error | Boosted Base | Boosted Sample | $d$          | $r$          | Base Error | Boosted Base | Boosted Sample | $d$         | $r$         |
| 2             | 1.2        | 0.13         | <b>0.587</b>   | <b>0.270</b> | <b>0.214</b> | 11.6       | 11.3         | <b>11.2</b>    | <b>11.2</b> | <b>11.2</b> |
| 4             | 1.2        | 0.13         | <b>0.270</b>   | <b>0.183</b> | <b>0.153</b> | 11.6       | 11.3         | <b>11.1</b>    | <b>11.2</b> | <b>11.2</b> |
| 8             | 1.2        | 0.13         | <b>0.246</b>   | <b>0.294</b> | <b>0.294</b> | 11.6       | 11.3         | 11.6           | <b>11.1</b> | <b>11.0</b> |
| 16            | 1.2        | 0.13         | <b>0.579</b>   | <b>0.421</b> | <b>0.381</b> | 11.6       | 11.3         | 11.7           | 11.6        | 11.6        |
| 32            | 1.2        | 0.13         | <b>1.10</b>    | <b>0.674</b> | <b>0.579</b> | 11.6       | 11.3         | 13.5           | 12.5        | 11.8        |
| 64            | 1.2        | 0.13         | 2.13           | <b>0.762</b> | <b>0.674</b> | 11.6       | 11.3         | 12.4           | 11.9        | 12.1        |
| 128           | 1.2        | 0.13         | 2.48           | <b>0.823</b> | <b>0.762</b> | 11.6       | 11.3         | 15.1           | 12.6        | 12.7        |
| 256           | 1.2        | 0.13         | 4.52           | <b>0.786</b> | <b>0.786</b> | 11.6       | 11.3         | 14.1           | 12.4        | 12.4        |

Table 2:  $d$ - and  $r$ - Sampling AdaBoost Error Rate Summary (%)

## 5.2 Results for Scalable and Distributed Learning

The results for  $d$ - and  $r$ -sampling are summarized in Table 2. The detailed plots (Figures 5 through 8) show the change in error rate with respect to sample size and boosting rounds. For each data set and each chosen sample size, the last two two columns of Table 2 (under “ $d$ ” and “ $r$ ”) lists the error rate for  $d$ - and  $r$ - sampling. As a comparison, the table also lists, from the first to third columns, the *baseline error rate* of a single RIPPER classifier learned from all the available data ( under “Baseline Error”), the *boosted baseline error rate* of 10 rounds AdaBoost RIPPER on all available data (under “Boosted Base”) and the *boosted sample error rate* of running AdaBoost RIPPER on the “same” small randomly chosen sample with equal size and number of rounds as used by  $d$ - and  $r$ - sampling (under column “Boosted Sample”). In each detailed plot, the x-axis is the rounds of boosting and the y-axis is the error rate. For all data sets, we draw a plot for every chosen sample size. There are five curves in each detailed plot. Two horizontal lines are the baseline error rate and the boosted baseline error rate (lower one). The other three lines display the results for  $d$ -sampling (drawn with lines points),  $r$ -sampling and boosted sample (both drawn with lines). In most cases, the curves

of  $d$ - and  $r$ -sampling are twisted together and hard to separate. The boosted sample curve is mostly high above  $d$ - and  $r$  sampling lines and becomes very flat with increasing boosting rounds.

From the curves and the summary table, we can see that  $d$ - and  $r$ -sampling AdaBoost reduce the error rate significantly. We observe that the error rates achieved by  $d$ - and  $r$ -sampling are either comparable to or even much lower than the baseline error rate of the global classifier. In many cases, their error rates are comparable to that of AdaBoost RIPPER on the complete training set. In Table 2, the results lower than the baseline are highlighted in bold font. This result applies to all data sets with all sample sizes ( $\frac{1}{2}$  to  $\frac{1}{256}$ ) under study. The error rates achieved by the different sample sizes for the same data set are in comparable levels, but bigger sample sizes exhibit slightly lower error rates. These observations suggest that both  $d$ - and  $r$ - sampling are quite robust to the change in sample size. In real world applications, we are mainly memory-constrained. These results show that we can overcome these constraints and use AdaBoost  $d$ - and  $r$ - sampling to compute accurate classifiers.

We compare the performance of  $d$ - and  $r$ -sampling with simply applying AdaBoost RIPPER to the same sample. From the curves, we can see that when the sample size is “big” ( $\frac{1}{2}$  to  $\frac{1}{16}$ ), boosting the same sample can reduce the error rate for BOOLEAN and WHIRL. However the level of reduction is not as much as either  $d$ - or  $r$ -sampling. For ADULT and CHASE, simply boosting the same sample increases the error rate for bigger sample sizes. The possible reason for reducing the error rate for “big” samples is that the sample is big enough for effective learning. When the sample size is small ( $\frac{1}{32}$  to  $\frac{1}{256}$ ), we observe a trend in error reduction early but it quickly flattens, yet the final resultant error rate is still significantly higher than the error rates attained by  $d$ - and  $r$ - sampling.

The speed of error reduction is very fast. By looking at the curves in Figures 5 to 8, we find that the quickest error reduction usually happens in the first 5% to 40% of the total number of boosting rounds. This is especially true when the sample sizes are small. In practice, it means that we can stop the learning process quite early without losing much accuracy. We also see that the error rate is still slowly decreasing at the last rounds of boosting. If we had allowed more time to compute, we could have obtained even lower error rates.

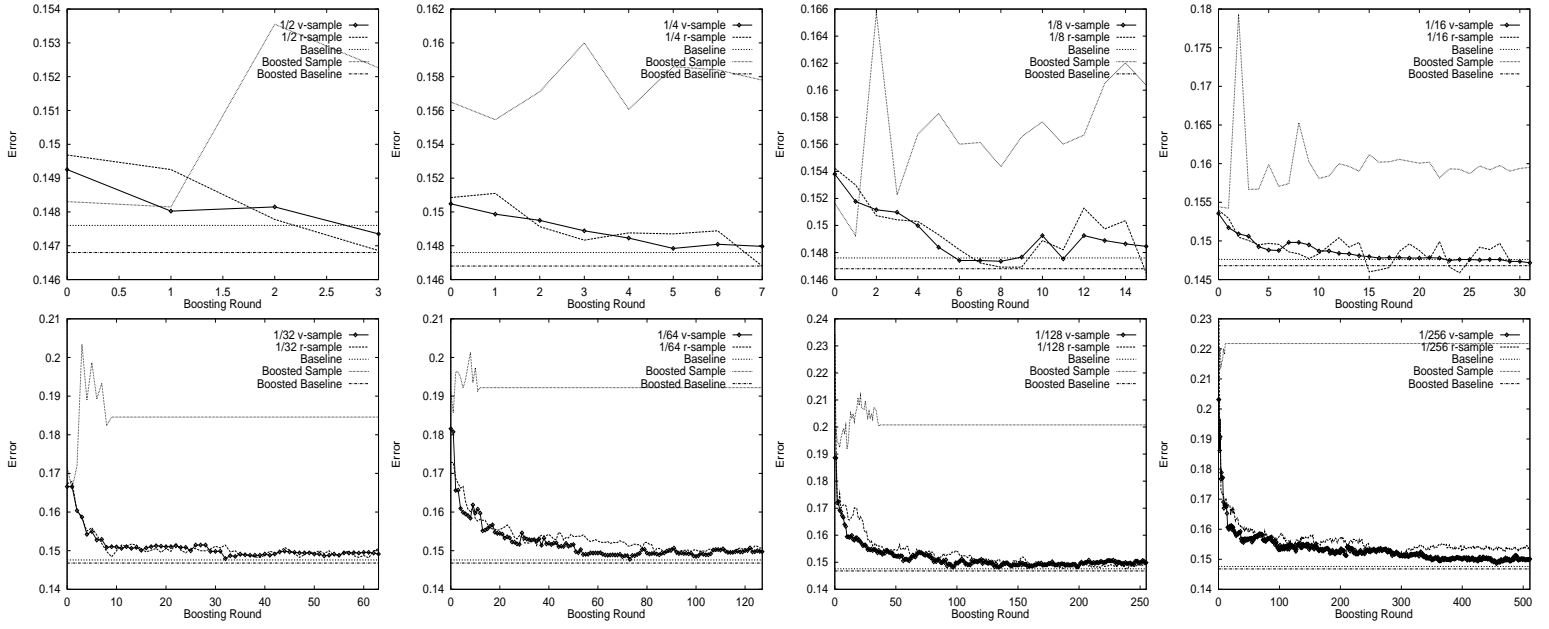


Figure 5: ADULT  $d$ - and  $r$ - sampling Results

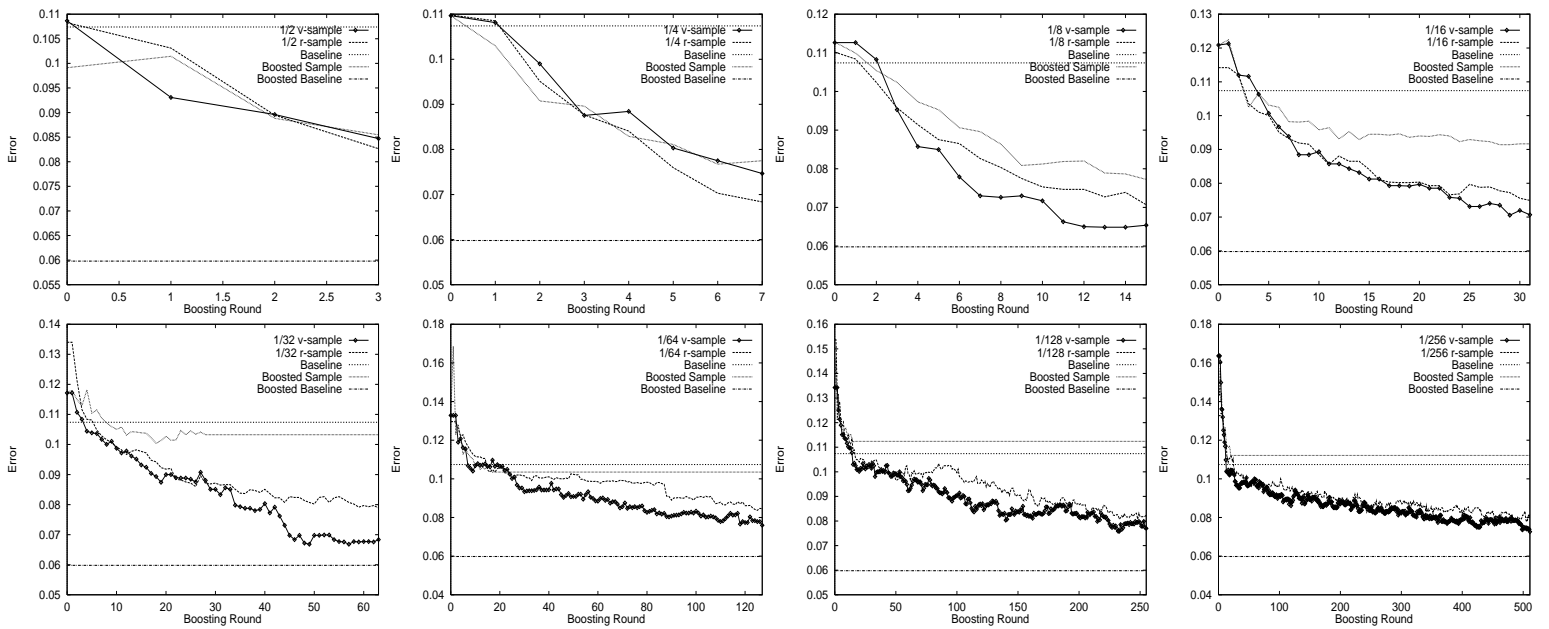


Figure 6: BOOLEAN  $d$ - and  $r$ - sampling Results

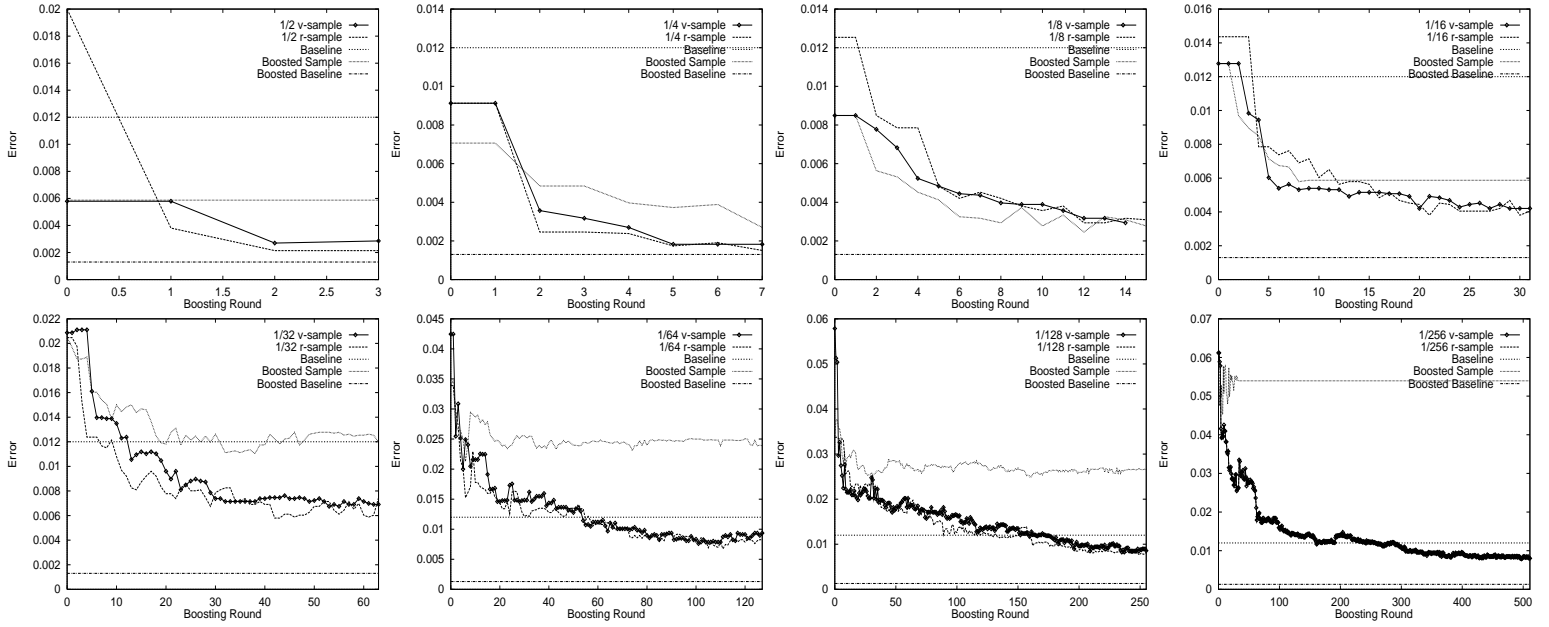


Figure 7: WHIRL  $d$ - and  $r$ - sampling Results

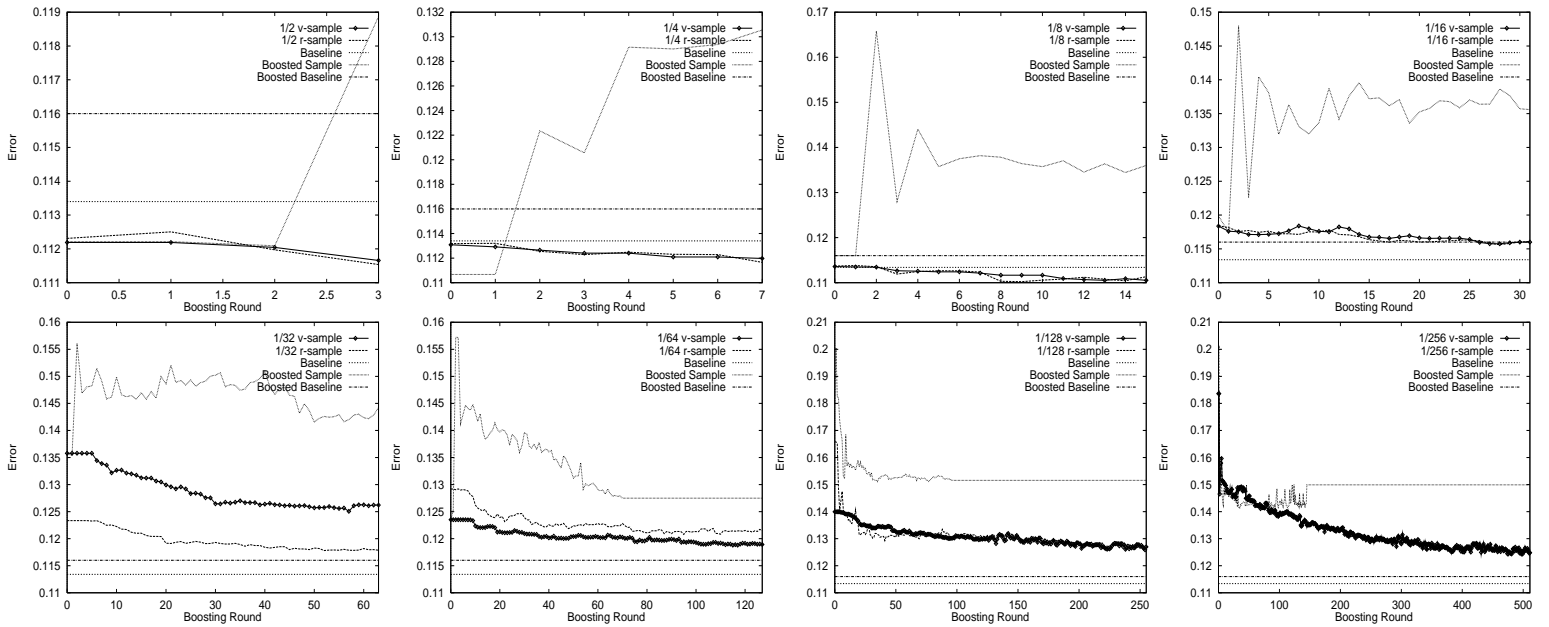


Figure 8: CHASE  $d$ - and  $r$ - sampling Results

|               | ADULT    |        |      |         |      |      | BOOLEAN  |        |      |         |      |      |
|---------------|----------|--------|------|---------|------|------|----------|--------|------|---------|------|------|
| $\frac{1}{s}$ | Baseline | Sample | Avg  | Perfect | Good | Bad  | Baseline | Sample | Avg  | Perfect | Good | Bad  |
| 32            | 14.8     | 16.8   | 15.9 | 18.8    | 53.1 | 28.1 | 10.7     | 13.5   | 10.0 | 81.3    | 18.8 | 0    |
| 64            | 14.8     | 19.3   | 15.8 | 28.1    | 68.8 | 3.1  | 10.7     | 13.6   | 10.9 | 40.6    | 56.2 | 3.1  |
| 128           | 14.8     | 22     | 15.3 | 42.2    | 56.3 | 1.56 | 10.7     | 14.4   | 12.2 | 26.6    | 56.3 | 17.2 |
| 256           | 14.8     | 22.5   | 15.7 | 38.7    | 58.6 | 2.73 | 10.7     | 17.5   | 13.7 | 22.3    | 60.5 | 17.2 |
|               | WHIRL    |        |      |         |      |      | CHASE    |        |      |         |      |      |
| $\frac{1}{s}$ | Baseline | Sample | Avg  | Perfect | Good | Bad  | Baseline | Sample | Avg  | Perfect | Good | Bad  |
| 32            | 1.2      | 2.45   | 1.22 | 53.1    | 43.8 | 3.13 | 11.6     | 13.2   | 11.4 | 56.3    | 40.6 | 3.13 |
| 64            | 1.2      | 4.1    | 1.30 | 56.3    | 43.8 | 0    | 11.6     | 14.6   | 11.4 | 57.8    | 39.1 | 3.13 |
| 128           | 1.2      | 4.6    | 2.04 | 11.7    | 86.7 | 1.56 | 11.6     | 15     | 11.0 | 59.4    | 36.7 | 3.91 |
| 256           | 1.2      | 6.5    | 3.00 | 6.25    | 92.2 | 1.56 | 11.6     | 18     | 11.2 | 59.0    | 39.1 | 1.95 |

Table 3: On-line AdaBoost with window size  $k=10$  Error Rate Summary (%)

There isn’t much difference in performance between  $d$ -sampling and  $r$ -sampling for the data sets under study. The curves are mostly “twisted” with each other. In training sets with highly skewed distributions, we may see some difference between these two methods.

For WHIRL and CHASE with sampling sizes of  $\frac{1}{2}$  and  $\frac{1}{4}$ , the error rate of the classifier learned from the sample before boosting is even lower than that of the global classifier. This is probably because  $\frac{1}{2}$  and  $\frac{1}{4}$  are large enough for effective learning, and overfitting easily occurs with more data.

### 5.3 Results for On-line Learning

The results are summarized in Table 3. Detailed scatter plots are shown in Figures 9 to 12. In each scatter plot, we draw two lines for comparison: baseline error rate of the global classifier learned over all available data and *average sample error rate*. Sample error is the error of the single classifier learned on the increment itself. For each data set under study, Table 3 shows the baseline error rate (under “Baseline”), average sample error rate (under “Sample”), average error rate (under “Avg”) of on-line AdaBoost for the flow of increments, and three categories to distinguish on-line AdaBoost performance. An on-line AdaBoost error rate is *perfect* if it is lower or equal to the baseline error rate. It is *bad* if it is higher than the error rate of the monolithic classifier learned on the on-line increment. Otherwise, it is in the *good* category.

In most of the cases (96+% for WHIRL and CHASE, 80+% for ADULT and BOOLEAN), the

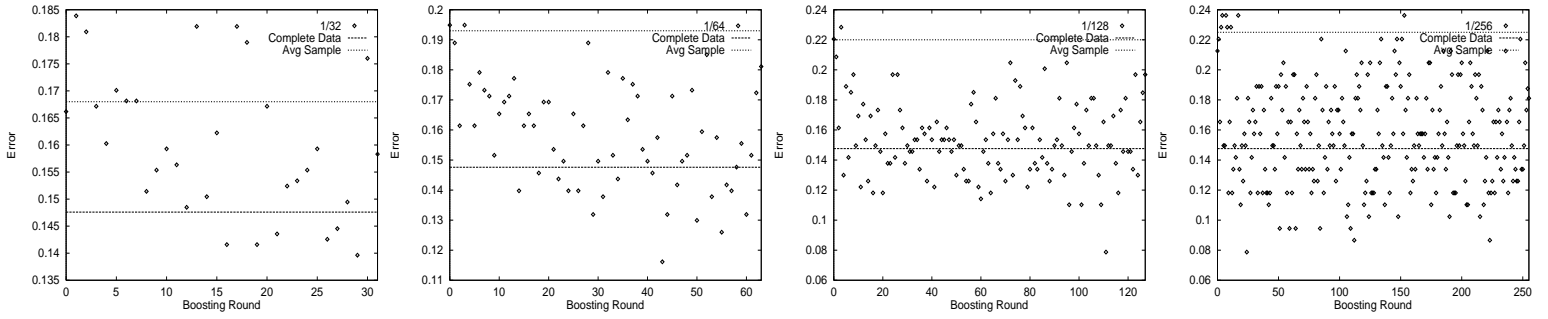


Figure 9: ADULT On-line Results

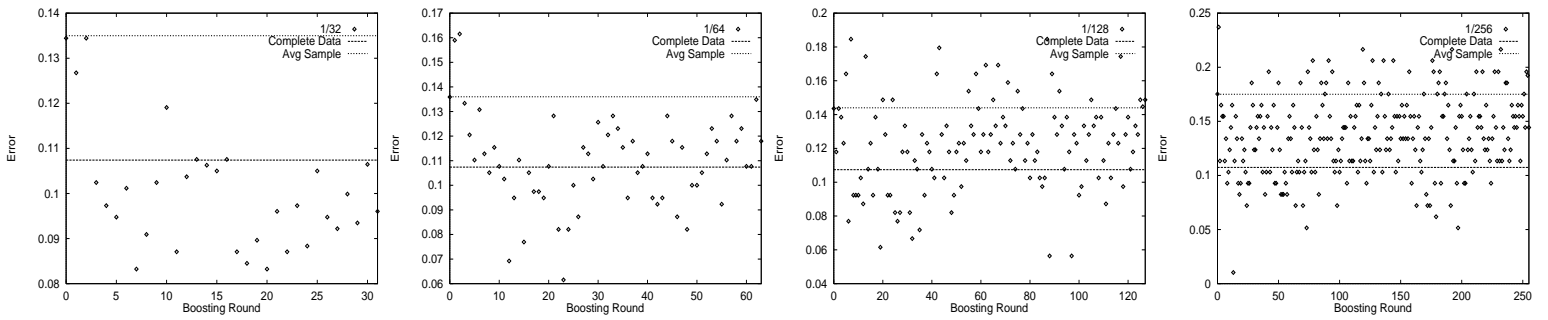


Figure 10: BOOLEAN On-line Results

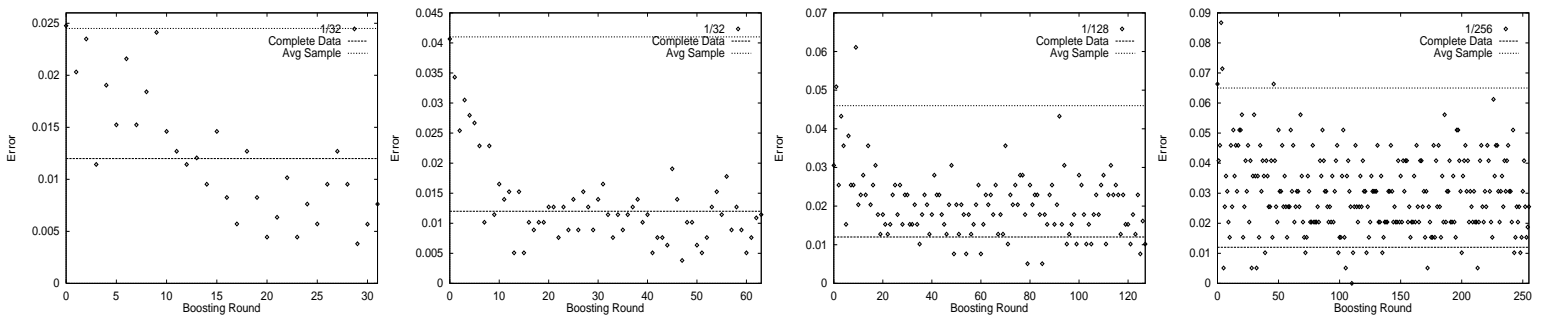


Figure 11: WHIRL On-line Results

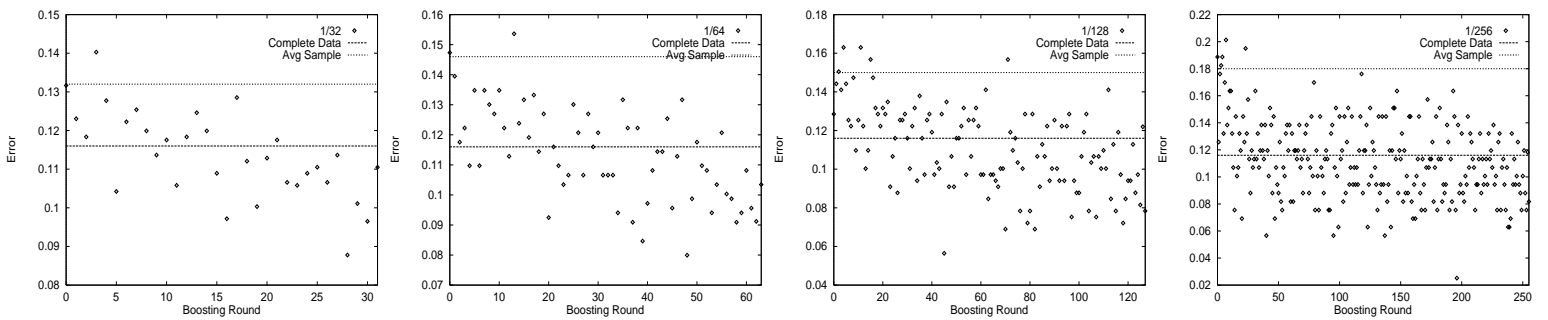


Figure 12: CHASE On-line Results

on-line error rate are in either good or perfect categories, implying that in an overwhelming majority of the cases, on-line learning has better performance than learning a single classifier from the on-line increment. In many cases (from 20% to 60%), the on-line error rate is even lower than the error rate of a global classifier learned over all available training data. For the CHASE data set, there are many points where the error rate is much lower than the boosted base line error rate.

Comparing the change in performance for the same data set with different sample sizes ( $\frac{1}{32}$  to  $\frac{1}{256}$ ), we find that the results are quite insensitive to the size of the sample. Except for WHIRL, there is an insignificant increase in average error rate when the size decreases. The percentage of results in the bad category remains almost the same for all experiments.

## 6 Implementation Scheme

In previous work [22], Stolfo, et al, implemented the JAM system, a distributed, scalable and portable data mining system that employs “meta-learning” for scalable data mining applications. JAM provides a set of learning agents and classifier agents, implemented as either JAVA applets or applications, that compute models and label data stored at a site. It uses a special distribution mechanism which allows the migration of the derived classifier agents to other remote sites. The boosting algorithms for scalable and distributed learning can be easily incorporated into the existing agent architecture (as shown in Figure 13).

The learning and classifier agents are used as the weak learners and weak hypotheses. Each classifier agent at each round of boosting (corresponding to  $h_t$ ) is sent to every site  $j$  to label its local data and produce  $u_{j:i}$  (defined as  $u_{j:i} = y_{j:i}h_t(x_{j:i})$ ,  $j : i$  refers the  $i$ -th example at the  $j$ -th site). The configuration file manager [22] specifies a site to run an  $\alpha$ -agent to compute  $\alpha_t$  at each round. Each site runs an  $\alpha$ -server agent. There are two ways to calculate  $\alpha_t$  as discussed in Section 3. In the first, the server agents of all sites will send their local predictions  $u_{j:i}$  to the  $\alpha$ -agent. In the second method, each server site participates in the calculation. The  $\alpha$ -agent sends them a candidate of  $\alpha_t$  denoted as  $\alpha_t^c$ . Each server-agent will return a partial sum of  $Z'$ . The partial sum at site  $j$  is given by  $Z'_j = -\sum D_t(j : i)u_{j:i}\exp(-\alpha_t^c u_{j:i})$ . The  $\alpha$ -agent will sum up all partial sums from

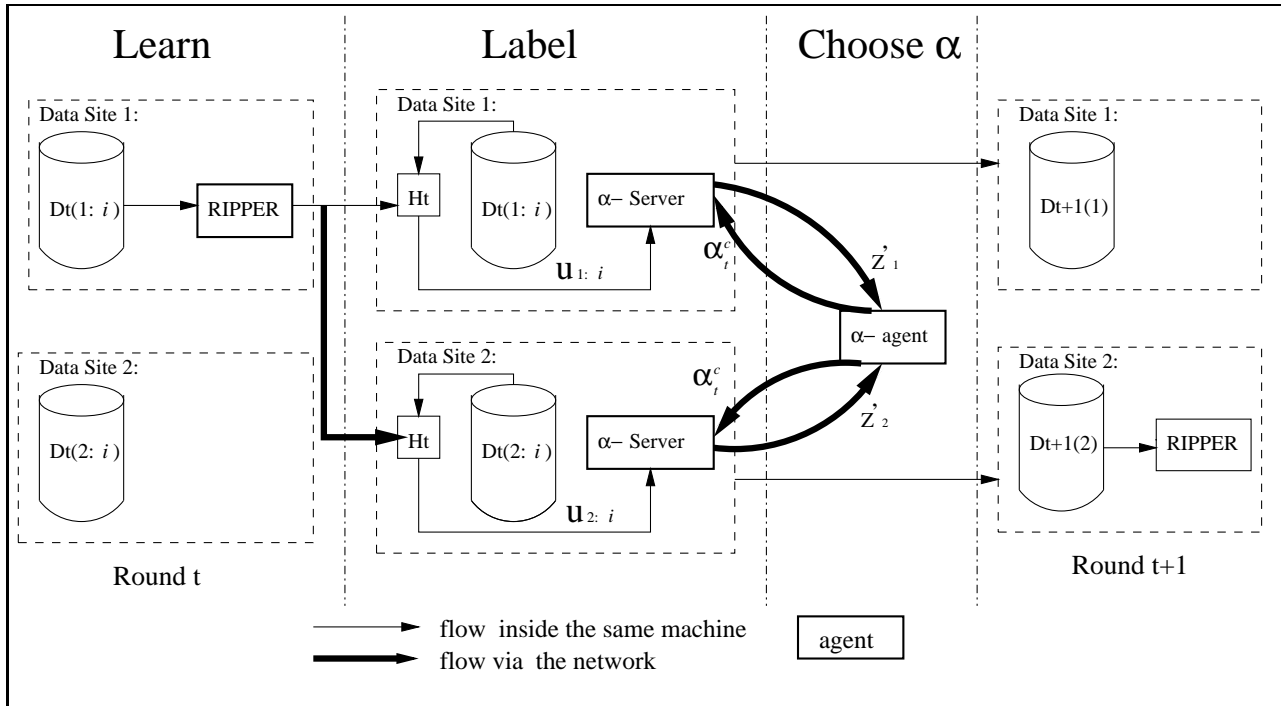


Figure 13: Boosted Agents

each server agent,  $Z' = \sum Z'_j$ . The numerical calculation typically takes several iterations. The final choice of  $\alpha_t$  is broadcast to every server agent in order to update  $D_t$  for its local data. The *boosting classifying agent* keeps every  $\alpha_t$  and a reference to every classifier agent produced during boosting. Both  $\alpha$  and  $\alpha$ -server agents expire at the end of learning. During classification, the boosting classifying agent launches all learned classifier agents and sums their weighted predictions.

## 7 Discussion

One drawback of combining methods for scalable learning is that a large number of classifiers may possibly be learned from many samples. In this situation, post-learning pruning is necessary to increase system throughput. Margineantu and Dietterich [16] have used several techniques to prune the final ensemble of AdaBoost. One of their approaches, “Reduce-error”, can still maintain about 90% of the original accuracy after 40% of pruning and preserve about 80% after 60% of pruning. To recover some of the lost accuracy by pruning, one possible avenue of research is to use the weight updating rule to re-weight these

un-pruned classifiers and learn an additional classifier on portions of the training set where these un-pruned classifiers perform poorly. This is similar to the scheme to reuse classifiers. The re-weighted ensemble may be more accurate than the simply pruned ensemble. An alternative method is to handle redundancy in the voted ensemble and only keep some important components. Merz and Pazzani [17] have used principle component analysis for this task and it may be applicable to the scenarios of improving classification efficiency. An additional method is to learn a single classifier that has similar accuracy to the voted ensemble. Domingos [11] provides a meta-learning method, “CMM”, to learn correlations of classifiers in a combined ensemble and re-learn a single classifier that has similar decision boundaries. Although, his method is originally proposed to solve comprehensibility of combined models, it may be used to increase throughput as well.

We have experimented with the idea of using the weight updating rule as a way of reusing old classifiers. The same method can also be used for “foreign knowledge import”. Chan [6] used meta-learning to import classifiers from foreign sites to improve accuracy on the local site. The on-line AdaBoost algorithm can be used in this situation, where the imported hypotheses can be taken as the previously learned hypotheses in on-line learning.

## 8 Conclusion

This paper has raised an interesting question: can we use AdaBoost for scalable, distributed and on-line learning. In this study, we have given two new ways to apply AdaBoost. In the first case, we only choose samples from the complete weighted training set. Each weak classifier is expected to be “weaker” than one trained from the complete training set. However, boosting can still increase the overall accuracy of the voted ensemble after many rounds. This approach allows us to use AdaBoost for scalable and distributed learning. In the second case, we regard the AdaBoost weight updating formula as a way of assigning weights to classifiers in a weighted voting ensemble. With this approach, we have an opportunity to reuse old classifiers on new data sets and learn a new classifier to concentrate on portions of the new data where the old classifiers perform poorly. We have experimented with  $d$ - and  $r$ -sampling as two alternatives for scalable and distributed learning. We tested them on four

real world and artificial data sets. The results are in most cases comparable to or better than learning a global classifier from the complete training set and in many cases comparable to boosting the global classifier on the complete data set. However, the cost of learning and the requirements for memory are significantly lower. We also tested an on-line AdaBoost with window size of 10 on the same data sets. The results suggest significant improvement from learning a single classifier on the new increment of data itself and in many cases even better than learning the global classifier where all data participates in learning. But its cost is similar to learning over the new increment data. The storage overhead for all these methods is bounded and can be pre-allocated before runtime. The computation overhead is also limited.

## References

- [1] R. Agrawal, M. Mehta, J. Shafer and R. Srikant. The Quest Data Mining System. In *Proc. KDD-96*, Portland, Oregon, pp.244-249.
- [2] L. Breiman. Pasting Small Votes for Classification Large Databases and On-line. will appear in *Machine Learning*
- [3] L. Breiman. Arcing Classifiers. In *The Annals of Statistics*, 26(3):801-849, 1998
- [4] L. Breiman. and P. Spector. Parallelizing CART using a workstation network. In *Proc Annual American Statistical Association Meeting 1994*
- [5] P. Chan. An Extensible Meta-learning Approach for Scalable and Accurate Inductive Learning. PhD Thesis, Department of Computer Science, Columbia University, 1997.
- [6] P. Chan and S. Stolfo. Sharing Learned Models Among Remote Database Partitions by Meta-learning. In *Proc. KDD-96*, Portland, Oregon.
- [7] S. Clearwater, T.P. Cheng, H. Hirsh and B.G. Buchanan. Incremental Batch Learning. not known of its publication source.
- [8] W. Cohen. Fast Effective Rule Induction. In *Proc. Twelfth International Conference on Machine Learning*, pp. 115-123, Morgan Kaufman.
- [9] W. Cohen and W. Fan. Learning Page-independent Heuristics for Extracting Data from the Web. accepted for WWW-99, Toronto, Canada.
- [10] T. Dietterich. An experimental comparison of three methods for construction ensembles of decision trees: Bagging, boosting and randomization. submitted
- [11] P. Domingos. Knowledge Acquisition from Examples Via Multiple Models. In *Proc. of ICML-97*

- [12] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119-139,1997.
- [13] J. Friedman, T. Hastie and R. Tibshirani. Additive Logistic Regression: a statistical view of boosting. *Technical Report*,
- [14] E. Han,G. Karypis and V. Kuma. Scalable Parallel Data Mining for Association Rules. *Proc ACM-SIGMOD-97*.
- [15] G. Karakoulas and J. Shawe-Taylor. Optimizing classifiers for imbalanced training sets. In *NIPS-1998*.
- [16] D, Margineantu and T. Dietterich. Pruning Adaptive Boosting. In *Proc of ICML-97*.
- [17] C. Merz and M. Pazzani. Handling Redundancy in Ensemble of Learned Models Using Principal Components. In *Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms WorkShop, AAAI-96*.
- [18] F.J. Provost and D. Hennesey. Scaling Up: Distributed Machine Learning with Cooperation. In *Proc of AAAI-96*.
- [19] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 1998.
- [20] R. Schapire, Y. Freund, P. Bartlett and W.S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, to appear.
- [21] J. Shafer, R. Agrawal and M. Mehta. Fast Serial and Parallel Classification of Very Large Databases. In *VLDB-96*.
- [22] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan and P. Chan. JAM: Java Agents for Meta-learning over Distributed Databases. In *Prod. Third Intl. Conf. Knowledge Discovery and Data Mining*, 1997.
- [23] P. Utgoff. An improved algorithm for incremental induction of decision. In *Proc. 11th Intl. Conf. Machine Learning*, 1994, pp318-325.