Boosting with Averaged Weight Vectors

Nikunj C. Oza

Computational Sciences Division
NASA Ames Research Center
Mail Stop 269-3
Moffett Field, CA 94035-1000, USA
oza@email.arc.nasa.gov

Abstract. AdaBoost [5] is a well-known ensemble learning algorithm that constructs its constituent or *base* models in sequence. A key step in AdaBoost is constructing a distribution over the training examples to create each base model. This distribution, represented as a vector, is constructed to be orthogonal to the vector of mistakes made by the previous base model in the sequence [7]. The idea is to make the next base model's errors uncorrelated with those of the previous model. Some researchers have pointed out the intuition that it is probably better to construct a distribution orthogonal to the mistake vectors of *all* the previous base models, but that this is not always possible [7]. We present an algorithm that attempts to come as close as possible to this goal in an efficient manner. We present experimental results demonstrating significant improvement over AdaBoost and the Totally Corrective boosting algorithm [7], which also attempts to satisfy this goal.

1 Introduction

AdaBoost [5] is one of the most well-known and highest-performing ensemble classifier learning algorithms [4]. It constructs a sequence of base models, where each model is constructed based on the performance of the previous model on the training set. In particular, AdaBoost calls the base model learning algorithm with a training set weighted by a distribution.[1] After the base model is created, it is tested on the training set to see how well it learned. We assume that the base model learning algorithm is a *weak learning algorithm* [6]; that is, with high probability, it produces a model whose probability of misclassifying an example is less than 0.5 when that example is drawn from the same distribution that generated the training set. The point is that such a model performs better than random guessing.[2] The weights of the correctly classified examples and

[1] If the base model learning algorithm cannot take a weighted training set as input, then one can create a sample with replacement from the original training set according to the distribution and call the algorithm with that sample.

[2] The version of AdaBoost that we use was designed for two-class classification problems. However, it is often used for a larger number of classes when the base model learning algorithm is strong enough to have an error less than 0.5 in spite of the larger number of classes.

misclassified examples are scaled down and up, respectively, so that the two groups' total weights are 0.5 each. The next base model is generated by calling the learning algorithm with this new weight distribution and the training set. The idea is that, because of the weak learning assumption, at least some of the previously misclassified examples will be correctly classified by the new base model. Previously misclassified examples are more likely to be classified correctly because of their higher weights, which focus more attention on them. Kivinen and Warmuth [7] have shown that AdaBoost scales the distribution with the goal of making the next base model's mistakes uncorrelated with those of the previous base model. It is well-known that ensembles need to have low correlation in their base models' errors in order to perform well [11].

Given this point, we would think, as was pointed out in [7], that AdaBoost would perform better if the next base model's mistakes were uncorrelated with those of *all* the previous base models instead of just the previous one. It is not always possible to construct a distribution consistent with this requirement. However, we can attempt to find a distribution that comes as close as possible to satisfying this requirement. Kivinen and Warmuth [7] devised the *Totally Corrective* algorithm (TCA), which attempts to do this. However, they do not present any empirical results. Also, they hypothesize that this algorithm will overfit and; therefore, not perform well. This paper presents a new algorithm, called AveBoost, which has the same goal as the TCA. In particular, AveBoost calculates the next base model's distribution by first calculating a distribution the same way as in AdaBoost, but then averaging it elementwise with those calculated for the previous base models. In this way, AveBoost takes all the previous base models into account in constructing the next model's distribution. In Section 2, we review AdaBoost and describe the TCA. In Section 3, we state the AveBoost algorithm and describe the sense in which our solution is the best one possible. In Section 4, we present an experimental comparison of AveBoost with AdaBoost and the TCA. Section 5 summarizes this paper and describes ongoing and future work.

2 AdaBoost and Totally Corrective Algorithm

Figure 1 shows AdaBoost's pseudocode. AdaBoost constructs a sequence of base models h_t for $t \in \{1, 2, \ldots, T\}$, where each one is constructed based on the performance of the previous base model on the training set. In particular, AdaBoost maintains a distribution over the m training examples. The distribution $\mathbf{d}_1$ used in creating the first base model gives equal weight to each example ($d_{1,i} = 1/m$ for all $i \in \{1, 2, \ldots, m\}$). AdaBoost now enters the loop, where the base model learning algorithm L_b is called with the training set and $\mathbf{d}_1$.[3] The returned model h_1 is then tested on the training set to see how well it learned. Training examples misclassified by the current base model have their weights increased

[3] As mentioned earlier, if L_b cannot take a weighted training set as input, then we can give it a sample drawn with replacement from the original training set according to the distribution $\mathbf{d}$ induced by the weights.

```
$\mathbf{AdaBoost}(\{(x_1, y_1), \ldots, (x_m, y_m)\}, L_b, T)$
  Initialize $d_{1,i} = 1/m$ for all $i \in \{1, 2, \ldots, m\}$.
  For $t = 1, 2, \ldots, T$:
      $h_t = L_b(\{(x_1, y_1), \ldots, (x_m, y_m)\}, \mathbf{d}_t)$.
      Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} d_{t,i}$.
      If $\epsilon_t \geq 1/2$ then,
          set $T = t - 1$ and abort this loop.
      Calculate distribution $\mathbf{d}_{t+1}$:
          $d_{t+1,i} = d_{t,i} \times \begin{cases} \frac{1}{2(1-\epsilon_t)} & \text{if } h_t(x_i) = y_i \\ \frac{1}{2\epsilon_t} & \text{otherwise.} \end{cases}$
  Output the final hypothesis:
      $h_{fin}(x) = \text{argmax}_{y \in Y} \sum_{t:h_t(x)=y} log \frac{1-\epsilon_t}{\epsilon_t}$.
```

Fig. 1. AdaBoost algorithm: $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ is the training set, L_b is the base model learning algorithm, and T is the maximum allowed number of base models.

```
$\mathbf{Totally\ Corrective\ AdaBoost}(\{(x_1, y_1), \ldots, (x_m, y_m)\}, L_b, T)$
  Initialize $d_{1,i} = 1/m$ for all $i \in \{1, 2, \ldots, m\}$.
  For $t = 1, 2, \ldots, T$:
      $h_t = L_b(\{(x_1, y_1), \ldots, (x_m, y_m)\}, \mathbf{d}_t)$.
      Calculate the mistake vector $\mathbf{u_t}$:
          $u_{t,i} = \begin{cases} 1 & \text{if } h_t(x_i) = y_i \\ -1 & \text{otherwise.} \end{cases}$
      If $\mathbf{d_t} \cdot \mathbf{u_t} \leq 0$ then,
          set $T = t - 1$ and abort this loop.
      Calculate distribution $\mathbf{d_{t+1}}$:
          Initialize $\hat{\mathbf{d}}_\mathbf{1} = \mathbf{d_1}$.
          For $j = 1, 2, \ldots$:
              $q_j = \text{argmax}_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}|$.
              $\hat{\alpha}_j = \ln\left(\frac{1+\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}}{1-\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}}\right)$.
              For all $i \in \{1, 2, \ldots, m\}$,
                  $\hat{d}_{j+1,i} = \frac{1}{\hat{Z}_j} \hat{d}_{j,i} exp(-\hat{\alpha}_j u_{q_j,i})$,
                  where $\hat{Z}_j = \sum_{i=1}^{m} \hat{d}_{j,i} exp(-\hat{\alpha}_j u_{q_j,i})$.
  Output the final hypothesis:
      $h_{fin}(x) = \text{argmax}_{y \in Y} \sum_{t:h_t(x)=y} log \frac{1-\epsilon_t}{\epsilon_t}$.
```

Fig. 2. Totally Corrective Boosting algorithm: $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ is the training set, L_b is the base model learning algorithm, and T is the maximum allowed number of base models.

for the purpose of creating the next base model, while correctly-classified training examples have their weights decreased. More specifically, if h_t misclassifies the ith training example, then its new weight $d_{t+1,i}$ is set to be its old weight $d_{t,i}$ multiplied by $\frac{1}{2\epsilon_t}$, where ϵ_t is the sum of the weights of the examples that h_t misclassifies. AdaBoost assumes that L_b is a *weak learner*, i.e., $\epsilon_t < \frac{1}{2}$ with high probability. Under this assumption, $\frac{1}{2\epsilon_t} > 1$, so the ith example's weight increases $(d_{t+1,i} > d_{t,i})$. On the other hand, if h_t correctly classifies the ith example, then $d_{t+1,i}$ is set to $d_{t,i}$ multiplied by $\frac{1}{2(1-\epsilon_t)}$, which is less than one by the weak learning assumption; therefore, example i's weight is decreased. Note that $\mathbf{d}_{t+1}$ is already normalized:

$$\begin{aligned}\sum_{i=1}^{m} d_{t+1,i} &= \frac{1}{2\epsilon_t}\sum_{i=1}^{m} d_{t,i} I(h_t(x_i) \neq y_i) + \frac{1}{2(1-\epsilon_t)}\sum_{i=1}^{m} d_{t,i} I(h_t(x_i) = y_i) \\ &= \frac{1}{2\epsilon_t}\epsilon_t + \frac{1}{2(1-\epsilon_t)}(1-\epsilon_t) = 1.\end{aligned}$$

Under distribution $\mathbf{d}_{t+1}$, the total weight of the examples misclassified by h_t and those correctly classified by h_t become 0.5 each. This is done so that, by the weak learning assumption, h_{t+1} will classify at least some of the previously misclassified examples correctly. As shown in [1], this weight update scheme is equivalent to the usual scheme [5] but is intuitively more clear. The loop continues, creating the T base models in the ensemble. The final ensemble returns, for a new example, the one class in the set of classes Y that gets the highest weighted vote from the base models.

Construct a vector $\mathbf{u}_t \in [-1, 1]^m$ such that the ith element $u_{t,i} = 1$ if h_t classifies the ith training example correctly $(h_t(x_i) = y_i)$ and $u_{t,i} = -1$ otherwise. Kivinen and Warmuth [7] pointed out that AdaBoost finds $\mathbf{d}_{t+1}$ by minimizing $\Delta(\mathbf{d}_{t+1}, \mathbf{d}_t)$ subject to $\mathbf{d}_{t+1} \cdot \mathbf{u}_t = 0$, where $\Delta(a, b)$ is a distance measure such as relative entropy. That is, the new distribution is created to be orthogonal to the mistake vector of h_t, which can be intuitively described as wanting the new base model's mistakes to be uncorrelated with those of the previous model. This naturally leads to the question of whether one can improve upon AdaBoost by constructing $\mathbf{d}_{t+1}$ to be orthogonal to the mistake vectors of *all* the previous base models $h_1, h_2, \ldots, h_t$ (i.e., $\mathbf{d}_{t+1} \cdot \mathbf{u}_q = 0$ for all $q \in \{1, 2, \ldots, t\}$). However, there is no guarantee that a *probability distribution* $\mathbf{d}_{t+1}$ exists that satisfies all the constraints. Even if a solution exists, finding it appears to be a very difficult optimization problem [7]. The Totally Corrective Algorithm (figure 2) attempts to solve this problem using an iterative method. The initial parts of the algorithm are similar to AdaBoost: it uses the same $\mathbf{d_1}$ as AdaBoost in creating the first base model and the next statement checks that the base model error is less than 0.5. The difference is in the method of calculating the weight distribution for the next base model. The TCA starts with some initial distribution such as $\mathbf{d_1}$. It then repeatedly finds the $q_j \in \{1, 2, \ldots, t\}$ yielding the highest $|\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u}_{\mathbf{q_j}}|$, and then projects the current distribution onto the hyperplane defined by $\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u}_{\mathbf{q_j}} = 0$. This is similar to so-called row action optimization

```
**AveBoost**$(\{(x_1, y_1), \ldots, (x_m, y_m)\}, L_b, T)$
    Initialize $d_{1,i} = 1/m$ for all $i \in \{1, 2, \ldots, m\}$.
    For $t = 1, 2, \ldots, T$:
        $h_t = L_b(\{(x_1, y_1), \ldots, (x_m, y_m)\}, \mathbf{d}_t)$.
        Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} d_{t,i}$.
        If $\epsilon_t \geq 1/2$ then,
            set $T = t - 1$ and abort this loop.
        Calculate orthogonal distribution:
            For $i = 1, 2, \ldots, m$:

                $c_{t,i} = d_{t,i} \times \begin{cases} \frac{1}{2(1-\epsilon_t)} & \text{if } h_t(x_i) = y_i \\ \frac{1}{2\epsilon_t} & \text{otherwise} \end{cases}$
                $d_{t+1,i} = \frac{t d_{t,i} + c_{t,i}}{t+1}$

    **Output** the final hypothesis:
        $h_{fin}(x) = \text{argmax}_{y \in Y} \sum_{t:h_t(x)=y} log \frac{1-\epsilon_t}{\epsilon_t}$.
```

Fig. 3. AveBoost algorithm: $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ is the training set, L_b is the base model learning algorithm, and T is the maximum allowed number of base models.

methods [3]. Kivinen and Warmuth show that, if there is a distribution that satisfies all the constraints, then at most $\frac{2 \ln m}{\gamma^2}$ iterations are needed so that $\max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u}_{\mathbf{q}_\mathbf{j}}| \leq \gamma$ for any $\gamma > 0$. Of course, as mentioned earlier, we cannot generally assume that there is a distribution that satisfies all the constraints, in which case the bound is invalid. In fact, we are not even guaranteed to reduce $\max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u}_{\mathbf{q}_\mathbf{j}}|$ at each iteration. To make the TCA usable for our experiments, we have added two stopping criteria not present in the original algorithm. Define $v_{t,j} = \max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u}_{\mathbf{q}_\mathbf{j}}|$. The algorithm stops if either $|v_{t,j} - v_{t,j-1}| < 0.0001$ or both $j > m$ and $v_{t,j} > v_{t,j-1}$. The first constraint requires that the maximum dot product change by some minimum amount between consecutive iterations. The second constraint requires that, after iterating at least as many times as the number of training examples, the maximum dot product not increase. These are heuristic criteria devised based on our experiments with this algorithm.

3 AveBoost algorithm

Figure 3 shows our new algorithm, AveBoost. Just as in AdaBoost, AveBoost initializes $d_{1,i} = 1/m$ for all $i \in \{1, 2, \ldots, m\}$. Then it goes inside the loop, where it calls the base model learning algorithm L_b with the training set and distribution $\mathbf{d}_1$ and calculates the error of the resulting base model h_1. It then calculates $\mathbf{c}_1$, which is the distribution that AdaBoost would use to construct the next base model. However, AveBoost averages this with $\mathbf{d}_1$ to get $\mathbf{d}_2$, and uses this $\mathbf{d}_2$ instead. Note that the $\mathbf{c}_t$'s for all $t \in \{1, 2, \ldots, T\}$ do not need to be normalized because they are calculated the same way as the $\mathbf{d}_t$'s in AdaBoost, which we showed to be distributions in Section 2. Showing that the $\mathbf{d}_t$'s in AveBoost are distributions is a trivial proof by induction. For the base case, $\mathbf{d}_1$ is constructed to be a distribution. For the inductive part, if $\mathbf{d}_t$ is a distribution, then $\mathbf{d}_{t+1}$ is a distribution because it is a convex combination of $\mathbf{d}_t$ and $\mathbf{c}_t$.

Returning to the algorithm, the loop continues for a total of T iterations. Then the base models are combined using the same weighted averaging used in AdaBoost. The vector $\mathbf{d}_{t+1}$ is a running average of $\mathbf{d}_1$ and the vectors $\mathbf{c_q}$ for $q \in \{1, 2, \ldots, t\}$, which are orthogonal to the mistake vectors of the previous t base models ($\mathbf{u_q}$ for $q \in \{1, 2, \ldots, t\}$), respectively. For ease of exposition, define $\mathbf{c}_0 \triangleq \mathbf{d}_1$ and $\mathbf{u}_0$ to be any vector in $[-1, 1]^m$ such that its elements sum to zero. Then we have $\mathbf{d}_{t+1} = \frac{1}{t+1} \sum_{q=0}^{t} \mathbf{c}_q$. This $\mathbf{d}_{t+1}$ has the least average Euclidian distance to the vectors $\mathbf{c}_q$ for $q \in \{0, 1, \ldots, t\}$. That is, $\mathbf{d}_{t+1}$ is the solution $\mathbf{d}$ that minimizes the least-squares error $\sum_{q=0}^{t} \sum_{i=1}^{m} (c_{q,i} - d_i)^2$. In this sense, AveBoost finds a solution that does the best job of balancing among the t constraints $\mathbf{c_q} \cdot \mathbf{u_q} = 0$ with much less computational cost than an optimization method such as that used in the TCA.

AveBoost can be seen as a relaxed version of AdaBoost. When training examples are noisy and therefore difficult to fit, AdaBoost is known to increase the weights on those examples to excess and overfit them [4] because many consecutive base models may not learn them properly. AveBoost tends to mitigate this overfitting by virtue of its averaging process. For this reason, we expect the range of training example weights to be narrower for AveBoost than AdaBoost. AveBoost's averaging process limits the range of training set distributions that are explored, which we expect will increase the average correlations among the base models. However, we expect their average accuracies to go up. We also hypothesize that AveBoost will tend to show greater advantage over AdaBoost for small numbers of base models. When AveBoost creates a large ensemble, the last few training set distributions cannot be too different from each other because they are prepared by averaging over many previous distributions.

4 Experimental Results

In this section, we compare AdaBoost, the TCA, and AveBoost on the nine UCI datasets [2] described in Table 1. We ran all three algorithms with three different values of T, which is the maximum number of base models that the algorithm is allowed to construct: 10, 50, and 100. Each result reported is the average over 50 results obtained by performing 10 runs of 5-fold cross-validation. Table 1 shows the sizes of the training and test sets for the cross-validation runs.

Table 2 shows how often AveBoost significantly outperformed, performed comparably with, and significantly underperformed AdaBoost and the TCA.

Table 1. The datasets used in the experiments

Data Set	Training Set	Test Set	Inputs	Classes
Promoters	84	22	57	2
Balance	500	125	4	3
Breast Cancer	559	140	9	2
German Credit	800	200	20	2
Car Evaluation	1382	346	6	4
Chess	2556	640	36	2
Mushroom	6499	1625	22	2
Nursery	10368	2592	8	5
Connect4	54045	13512	42	3

Table 2. Performance of AveBoost

Compared to	Base Model	10	50	100
AdaBoost	Naive Bayes	+6=1-2	+4=3-2	+4=2-3
Totally Corrective	Naive Bayes	+6=2-1	+6=2-1	+6=2-1
AdaBoost	Decision Trees	+2=7-0	+2=5-2	+2=5-2
AdaBoost	Decision Stumps	+2=6-1	+2=4-3	+2=3-4

For example, with 10 Naive Bayes base models, AveBoost significantly outperformed[4] AdaBoost on six of the datasets, performed comparably on one dataset, and performed significantly worse on two, which is written as "+6=1-2." Figure 4 compares the error rates of AdaBoost and AveBoost with Naive Bayes base models. In all the plots presented in this paper, each point marks the error rates of two algorithms when run with the number of base models indicated in the legend and a particular dataset. The diagonal line in the plots contain points at which the two algorithms have equal error. Therefore, points below/above the line correspond to the error of the algorithm indicated on the y-axis being less than/greater than the error of the algorithm indicated on the x-axis, respectively. The scales for each plot were adapted for the error rates shown; therefore, they are different across the plots. We can see that, for Naive Bayes base models, AveBoost performs much better than AdaBoost overall. Figure 5 shows that AveBoost performs substantially better than the TCA. We examined the runs of the TCA in more detail and often found the overfitting that Kivinen and Warmuth thought would happen. Due to this poor performance, we did not continue experimenting with the TCA for the rest of this paper.

We compare AdaBoost and AveBoost using decision tree and decision stump base models in figures 6 and 7, respectively. With decision trees, AveBoost per-

[4] We use a t-test with $\alpha = 0.05$ to compare all the classifiers in this paper.

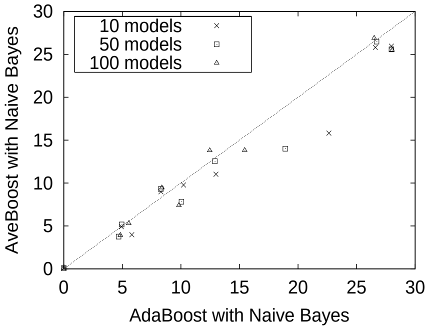

Fig. 4. Test set error rates of AdaBoost vs. AveBoost (Naive Bayes)

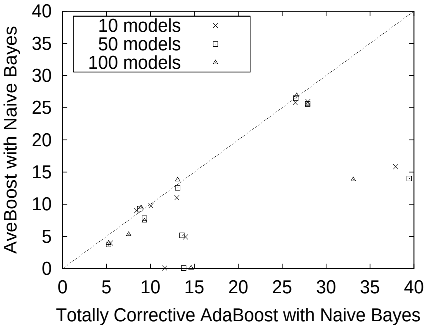

Fig. 5. Test set error rates of Totally Corrective Boosting vs. AveBoost (Naive Bayes)

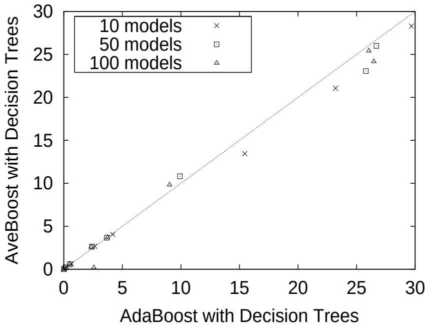

Fig. 6. Test set error rates of AdaBoost vs. AveBoost (Decision Trees)

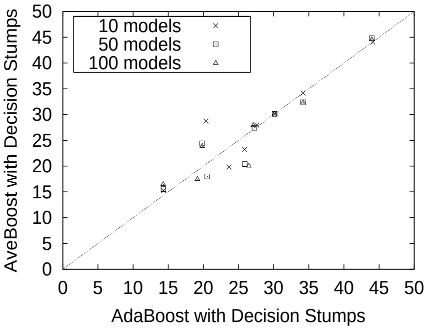

Fig. 7. Test set error rates of AdaBoost vs. AveBoost (Decision Stumps)

forms somewhat better than AdaBoost. With decision stumps, the differences in error rates vary much more, with AveBoost sometimes performing worse than AdaBoost.

We now analyze the performances of AdaBoost and AveBoost in more depth. Due to space limitations, we discuss our results with only Naive Bayes base models and for only two datasets: Promoters and Breast Cancer. We chose these two because the performances of AveBoost relative to AdaBoost are very different for these two datasets. On Promoters, AveBoost significantly outperformed AdaBoost, while on Breast Cancer, they performed comparably for 10 and 100 base models and AveBoost performed worse for 50 base models. Table 3 gives the results of our comparison. The top half of the table gives the Promoters dataset results while the bottom half gives the Breast Cancer results. For each dataset, the first row states the maximum allowed number of base models (this is T in Figures 1-3). The second row gives the average number of base models actually constructed over the 50 runs (recall that each algorithm has a stopping crite-

Table 3. Detailed comparison of AveBoost and AdaBoost on Promoters and Breast Cancer

PROMOTERS	AdaBoost NB			AveBoost NB		
Max. Base Models	10	50	100	10	50	100
Avg. Base Models	10	50	100	9.9	43.32	72.22
Ens. Train Perf.	0.9993	1.0	1.0	0.9998	1.0	1.0
Ens. Test Perf.	0.7736	0.8109	0.8455	0.8418	0.8600	0.8618
Avg. Corr. Train	0.3319	0.2901	0.2813	0.6229	0.5511	0.4277
Avg. Corr. Test	0.1877	0.1422	0.1395	0.4541	0.5159	0.5245
Base Train Perf.	0.7878	0.7597	0.7555	0.8917	0.9052	0.9085
Base Test Perf.	0.6514	0.6204	0.6210	0.7490	0.7613	0.7640
Min. Example Wt.	5.85e-05	3.40e-08	7.67e-10	0.0046	0.0024	0.0019
Max. Example Wt.	0.1748	0.2528	0.2821	0.0644	0.0670	0.0671
Sdev. Example Wt.	0.0177	0.0212	0.0217	0.0089	0.0098	0.0100

BREAST CANCER	AdaBoost NB			AveBoost NB		
Max. Base Models	10	50	100	10	50	100
Avg. Base Models	10	50	99.52	10	48.46	93.48
Ens. Train Perf.	0.9973	0.9998	1.0	0.9892	0.9985	0.9991
Ens. Test Perf.	0.9509	0.9506	0.9445	0.9509	0.9483	0.9470
Avg. Corr. Train	0.6409	0.4955	0.4638	0.9146	0.8688	0.7864
Avg. Corr. Test	0.6338	0.4960	0.4650	0.9210	0.9033	0.8958
Base Train Perf.	0.8918	0.8465	0.8340	0.9723	0.9727	0.9715
Base Test Perf.	0.8639	0.8209	0.8091	0.9429	0.9351	0.9324
Min. Example Wt.	7.06e-06	8.66e-15	4.96e-24	6.62e-04	3.25e-04	2.40e-04
Max. Example wt.	0.0934	0.1413	0.1643	0.0360	0.0434	0.0446
Sdev. Example Wt.	0.0073	0.0087	0.091	0.0039	0.0053	0.0056

rion that can result in fewer base models than what the user selects). AveBoost uses much fewer base models than AdaBoost on the Promoters dataset, but we found no correlation between number of base models and performance. The next two rows state the average ensemble performances on the training and test sets. Next are the average correlations of the outputs of all pairs of base models on the training and test sets. As expected, the correlations are higher for AveBoost than AdaBoost. However, the average training and test accuracies of the base models are also higher, as shown by the next two rows for each dataset. We then calculated, for each training example, the minimum and maximum weights ever assigned to it, and the standard deviation of all the weights assigned to it. The next three rows give the average, over all the training examples, of these minima, maxima, and standard deviations. As anticipated, the ranges of the weights for AveBoost are much lower than for AdaBoost.

5 Conclusions

We presented AveBoost, a boosting algorithm that trains each base model using a training example weight vector that is based on the performances of all the previous base models rather than just the previous one. We discussed the theoretical motivation for this algorithm and demonstrated empirical results that are superior overall to AdaBoost and the TCA that has the same goal as AveBoost.

We are currently analyzing AveBoost theoretically. The algorithmic stability-based framework [9] intuitively seems the most promising because of AveBoost's averaging process. We plan to empirically analyze the algorithms presented here for all the datasets in the style of Table 3 for a longer version of this paper. We also plan to devise synthetic datasets with various levels of noise to observe if AveBoost is more robust to noise than AdaBoost as we have hypothesized. Additionally, it has been pointed out [8, 10] that ensembles work best when they are somewhat anti-correlated. We attempted to exploit this by implementing several boosting algorithms that, at each iteration, change the base model weights so that the correctly classified examples' weights add up to slightly less than 0.5. This scheme occasionally performed better and occasionally performed worse than AdaBoost. Depending on the available running time, it may be possible to create classifiers using several of these weight adjustment schemes and use the ones that look most promising for the dataset under consideration.

References

1. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, Sep. 1999.
2. C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases, 1999. (URL: http://www.ics.uci.edu/∼mlearn/MLRepository.html).
3. Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34(3):321–353, 1981.
4. Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–158, Aug. 2000.
5. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann.
6. Michael J. Kearns and Umesh V. Vazirani. *Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1994.
7. Jyrki Kivinen and Manfred K. Warmuth. Boosting as entropy projection. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 134–144, 1999.
8. A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems-7*, pages 231–238. M.I.T. Press, 1995.
9. Samuel Kutin and Partha Niyogi. The interaction of stability and weakness in adaboost. Technical Report TR-2001-30, University of Chicago, October 2001.
10. Nikunj C. Oza. *Online Ensemble Learning.* PhD thesis, The University of California, Berkeley, CA, Dec 2001.
11. K. Tumer and J. Ghosh. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348, February 1996.