Boosting with Averaged Weight Vectors

Nikunj C. Oza
Computational Sciences Division
NASA Ames Research Center
Mail Stop 269-3
Moffett Field, CA 94035-1000
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 [6]. 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 that is orthogonal to the mistake vectors of all the previous base models, but that this is not always possible [6]. 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 [6], which also attempts to satisfy this goal.

1 Introduction

Ensemble learning algorithms are machine learning algorithms that, given a training set, construct a combination of *base models* drawn from a designated hypothesis class. AdaBoost [5] is one of the most well-known and high-performing ensemble learning algorithms. 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*; that is, its error is less than 0.5, so that

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

it performs better than random guessing.[2] The weights of the correctly classified examples and 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 weight, which focuses more attention on them. Kivinen and Warmuth [6] 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 [1, 9].

Given this point, we would think, as was pointed out in [6], 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 turns out that it is not always possible to construct a distribution consistent with this requirement. However, we can attempt to get as close as possible to such a distribution. That is, we may attempt to find a distribution such that the resulting base model's mistakes are as close as possible to being uncorrelated to all the past base models' mistakes. Kivinen and Warmuth [6] devised the *totally corrective* boosting algorithm, 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 Averaging AdaBoost, which has the same goal as the totally corrective algorithm. In particular, our algorithm 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, our algorithm attempts to take all the previous base models into account in constructing the next model's distribution. In Section 2, we review how AdaBoost works and, in particular, how it constructs its distributions over the training set for each base model. We also describe the totally corrective algorithm here. In Section 3, we state our algorithm and describe the sense in which our solution is the best one possible. In Section 4, we present an experimental comparison of our algorithm with AdaBoost and the totally corrective algorithm. Section 5 summarizes this paper and describes ongoing and future work.

2 AdaBoost

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 $i \in \{1, 2, \ldots\ m\}$). The base model learning algorithm L_b is called with the

[2]The version of AdaBoost that we use was designed for two-class classification problems. However, it is routinely 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.

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

Figure 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.

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 $\hat{\mathbf{d}}_1 = \mathbf{d_1}$.
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 $\hat{Z}_j = \sum_{i=1}^{m} \hat{d}_{j,i} exp(-\hat{\alpha}_j u_{q_j,i})$ is the normalizing factor.
Output the final hypothesis:
$h_{fin}(x) = \text{argmax}_{y \in Y} \sum_{t:h_t(x)=y} log \frac{1-\epsilon_t}{\epsilon_t}$.

Figure 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.

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 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. 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.

For all the base models h_t ($t \in \{1, 2, \ldots, T\}$) and the m training examples, 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 [6] pointed out that AdaBoost calculates $\mathbf{d_{t+1}}$ from $\mathbf{d_t}$ such that $\mathbf{d_{t+1}} \cdot \mathbf{u_t} = 0$. 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 to reduce a suitable loss function in a direction orthogonal to what the previous base model did, so that the new base model's mistakes are 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 base hypotheses $h_1, h_2, \ldots, h_t$ (i.e., $\mathbf{d_{t+1}} \cdot \mathbf{u_q} = 0$ for all $q \in \{1, 2, \ldots, t\}$). Constructing such a $\mathbf{d_{t+1}}$ is not always possible. In particular, if $m > t$, then the system of equations just given is overspecified, so that there may not be a solution. Kivinen and Warmuth's totally corrective algorithm (figure 2) attempts to solve this problem using an iterative method. The initial parts of the algorithm are similar to AdaBoost. That is, the totally corrective algorithm 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 totally corrective algorithm repeatedly finds the one among the t constraints that is most strongly violated, i.e., finds the value q_j having the highest value of $|\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}|$, and then projects the current distribution onto the hyperplane defined by that violated constraint. This is similar to so-called row action optimization methods [3, 4]. Kivinen and Warmuth show that, if there is a distribution that satisfies all the constraints, then there is an upper bound of $\frac{2 \ln m}{\gamma^2}$ on the number of iterations needed so that $\max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}| \leq \gamma$ 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 there is no such bound on the number of iterations. In fact, we are not even guaranteed to reduce $\max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}|$ at each iteration. To make the totally corrective algorithm usable for our experiments, we have added two stopping criteria not present in the original algorithm. Define $v_{t,j} = \max_{q_j \in \{1,2,\ldots,t\}} |\hat{\mathbf{d}}_\mathbf{j} \cdot \mathbf{u_{q_j}}|$. The algorithm stops if

[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 distribution $\mathbf{d}$.

Averaging 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 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:
$h_{fin}(x) = \text{argmax}_{y \in Y} \sum_{t:h_t(x)=y} log\frac{1-\epsilon_t}{\epsilon_t}$.

Figure 3: Our Averaging 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.

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 decrease by some minimum amount between consecutive iterations. The second constraint leaves the loop if, after iterating at least as many times as the number of training examples, the maximum dot product increases. These are heuristic criteria devised on the basis of observations of some of our experiments with this algorithm.

In the next section, we describe our algorithm.

3 Our algorithm

Figure 3 shows our new algorithm. Just as in AdaBoost, our algorithm 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, our algorithm averages this with $\mathbf{d_1}$ to get $\mathbf{d_2}$, and uses this $\mathbf{d_2}$ instead. The loop continues for a total of T iterations. The vector $\mathbf{d_{t+1}}$ is a running average of 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.

It is well-known that this $\mathbf{d_{t+1}}$ has the least average Euclidian distance to the vectors $\mathbf{c_q}$ for $q \in \{1, 2, \ldots, t\}$. In this sense, our algorithm finds a solution that does the best job of balancing among the t constraints $\mathbf{c_q} \cdot \mathbf{u_q} = \mathbf{0}$ without the computational cost of a convex optimization method. It is easy to prove that $\mathbf{d_{t+1}}$ is already a distribution (i.e., normalization is unnecessary), but space precludes us

Table 1: The datasets used in our 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

from doing so here

We now demonstrate the experimental usefulness of this algorithm.

4 Experimental Results

In this section, we compare AdaBoost, the totally corrective algorithm, and our averaging algorithm on 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.

Figure 4 compares the error rates of AdaBoost and our averaging algorithm 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 algorithm indicated on the y-axis being less than/greater than the error of the algorithm indicated on the x-axis, respectively. We can see that, for Naive Bayes base models, our averaging algorithm performs much better than AdaBoost overall. Table 2 shows how often our averaging algorithm significantly outperformed, performed comparably with, and significantly underperformed AdaBoost and the Totally Corrective Algorithm. In particular, for 10 base models, averaging 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" in the table. Figure 5 shows that our averaging algorithm performs substantially better than the Totally Corrective algorithm with our averaging algorithm. We examined the runs of the Totally Corrective algorithm 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 totally corrective algorithm for the rest of this paper.

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

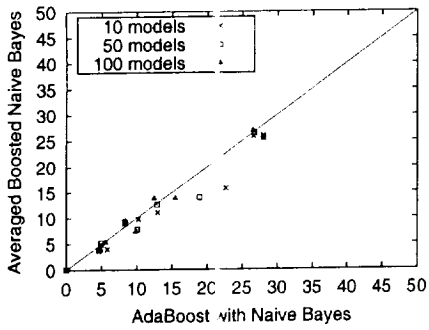

Figure 4: AdaBoost vs. Averaged Boosting (Naive Bayes)

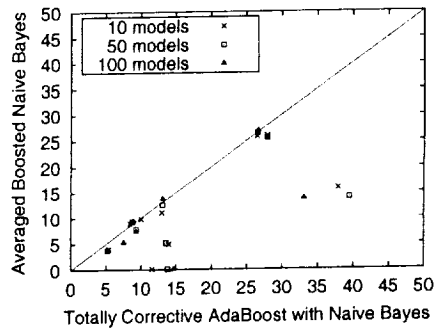

Figure 5: Totally Corrective Boosting vs. Averaged Boosting (Naive Bayes)

Table 2: Performance of Averaged Boosting

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

We compare AdaBoost and our averaging algorithm using decision tree and decision stump base models in figures 6 and 7, respectively. With decision trees, the averaging algorithm performs somewhat better than AdaBoost. With decision stumps, the differences in error rates vary much more, with averaging sometimes performing worse than AdaBoost.

5 Conclusions

We presented 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

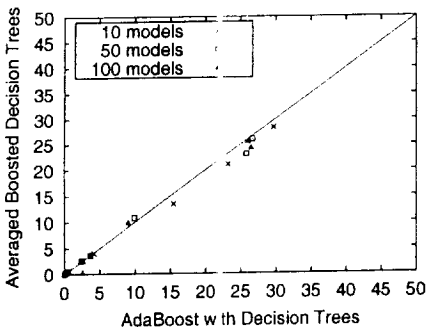

Figure 6: AdaBoost vs. Averaged Boosting (Decision Trees)

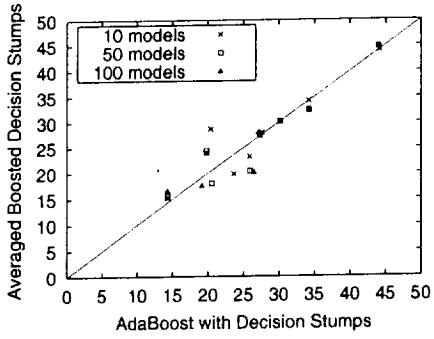

Figure 7: AdaBoost vs. Averaged Boosting (Decision Stumps)

models rather than just the previous one. We discuss the theoretical motivation for this algorithm and demonstrate empirical results that are superior overall relative to AdaBoost and the Totally Corrective algorithm that has the same goal as our algorithm.

Space precluded a detailed analysis of the performances of the base models and their correlations, as is often done in a detailed study of ensemble methods. We plan to do this for a longer version of this paper in order to compare our algorithm to AdaBoost and the Totally Corrective algorithm in more detail. This analysis may help to explain why Averaging AdaBoost's improvement over AdaBoost was greater for smaller numbers of base models. Additionally, it has been pointed out [7, 8] 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 not to 0.5, but 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 combine all of them or a subset of them in an ensemble, or perhaps cease using certain weight adjustment schemes if they do not look promising for the dataset under consideration.

References

[1] K. M. Ali and M. J. Pazzani. On the link between error correlation and error reduction in decision tree ensembles. Technical Report 95-38, Department of Information and Computer Science, University of California, Irvine, 1995.

[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] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Physics*, 7:200–217, 1967.

[4] 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.

[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] 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.

[7] 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.

[8] Nikunj C. Oza. *Online Ensemble Learning*. PhD thesis, The University of California, Berkeley, CA, Dec 2001.

[9] K. Tumer and J. Ghosh. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348, February 1996.