1-1-2008

Evidence Contrary to the Statistical View of Boosting

David Mease
San Jose State University, dmease@gmail.com

A. Wyner
Wharton School, University of Pennsylvania

Recommended Citation

David Mease and A. Wyner. "Evidence Contrary to the Statistical View of Boosting" *Journal of Machine Learning Research* (2008): 131-156.

Journal of Machine Learning Research 9 (2008) 131-156 Submitted 10/05; Revised 7/07; Published 2/08

Evidence Contrary to the Statistical View of Boosting

David Mease MEASE_D@COB.SJSU.EDU
Department of Marketing and Decision Sciences
College of Business, San Jose State University
San Jose, CA 95192-0069, USA

Abraham Wyner AJW@WHARTON.UPENN.EDU
Department of Statistics
Wharton School, University of Pennsylvania
Philadelphia, PA, 19104-6340, USA

Editor: Yoav Freund

Abstract

The statistical perspective on boosting algorithms focuses on optimization, drawing parallels with maximum likelihood estimation for logistic regression. In this paper we present empirical evidence that raises questions about this view. Although the statistical perspective provides a theoretical framework within which it is possible to derive theorems and create new algorithms in general contexts, we show that there remain many unanswered important questions. Furthermore, we provide examples that reveal crucial flaws in the many practical suggestions and new methods that are derived from the statistical view. We perform carefully designed experiments using simple simulation models to illustrate some of these flaws and their practical consequences.

Keywords: boosting algorithms, LogitBoost, AdaBoost

1. Introduction

As the AdaBoost algorithm of Freund and Schapire (1996) gained popularity in the computer science community because of its surprising success with classification, the statistics community focused its efforts on understanding how and why the algorithm worked. Friedman, Hastie and Tibshirani in 2000 made great strides toward understanding the AdaBoost algorithm by establishing a statistical point of view. Among the many ideas in the Friedman, Hastie and Tibshirani *Annals of Statistics* paper, the authors identified a stagewise optimization in AdaBoost, and they related it to the maximization of the likelihood function in logistic regression. Much work has followed from this paper: extensions of the algorithm to the regression setting (e.g., Buhlmann and Yu, 2003), modification of the loss function (e.g., Hastie et al., 2001), and work on regularization methods for the original AdaBoost algorithm and variants (e.g., Lugosi and Vayatis, 2004). This broad statistical view of boosting is fairly mainstream in the statistics community. In fact, the statistics community has taken to attaching the *boosting* label to *any* classification or regression algorithm that incorporates a stagewise optimization.

Despite the enormous impact of the Friedman, Hastie and Tibshirani paper, there are still questions about the success of AdaBoost that are left unanswered by this statistical view of boosting. Chief among these is the apparent resistance to overfitting observed for the algorithm in countless examples from both simulated and real data sets. This disconnect was noted in some of the discussions published along with the original 2000 *Annals of Statistics* paper. For instance, Freund and Schapire (2000) note that, "one of the main properties of boosting that has made it interesting to statisticians and others is its relative (but not complete) immunity to overfitting," and write that the paper by Friedman, Hastie and Tibshirani "does not address this issue." Also Breiman (2000) writes, "a crucial property of AdaBoost is that it almost never overfits the data no matter how many iterations it is run," and states "unless I am missing something, there is no explanation in the paper."

Various arguments are given in response to the question of why boosting seems to not overfit. A view popular in computer science attributes the lack of overfitting to boosting's ability to achieve a large *margin* separating the two classes, as discussed by Schapire et al. (1998). A number of different opinions exist in the statistics community. Many statisticians simply argue that boosting does in fact overfit and construct examples to prove it (e.g., Ridgeway, 2000). While single examples certainly disprove claims that boosting *never* overfits, they do nothing to help us understand why boosting resists overfitting and performs very well for the large collection of examples that raised the question in the first place. Others argue that boosting will eventually overfit in most all cases if run for enough iterations, but that the number of iterations needed can be quite large since the overfitting is quite slow. Such a notion is difficult to disprove through real examples since any finite number of iterations may not be enough. Furthermore, it is difficult to prove limiting results for an infinite number of iterations without substantially over-simplifying the algorithm. Some evidence supporting the argument that boosting will eventually overfit can be found in Grove and Schuurmans (1998) which has examples for which boosting overfits when run for a very large number of iterations. Another argument often used is that boosting's success is judged with respect to 0/1 misclassification loss, which is a loss function that is not very sensitive to overfitting (e.g., Friedman et al., 2000b). More detailed explanations attribute the lack of overfitting to the stagewise nature of the algorithm (e.g., Buja, 2000). Along this same line, it has also been observed that the repeated iterations of the algorithm give rise to a self-averaging property (e.g., Breiman, 2000). This self-averaging works to reduce overfitting by reducing variance in ways similar to bagging (Breiman, 1996) and Random Forests (Breiman, 2001).

Whatever the explanation for boosting's resistance to overfitting in so many real and important examples, the statistical view of boosting as an optimization does little to account for this. In fact the statistical framework as proposed by Friedman, Hastie and Tibshirani does exactly the opposite; it suggests that *overfitting should be a major concern*. Still, in the final analysis, we do not imply that the statistical view is wrong. Indeed, we agree with Buja (2000) who writes, "There is no single *true* interpretation of anything; interpretation is a vehicle in the service of human comprehension. The value of an interpretation is in enabling others to fruitfully think about an idea." Certainly the paper of Friedman, Hastie and Tibshirani and other related work is quite valuable in this regard. However, any view or theoretical understanding generally gives rise to practical suggestions for implementation. Due to the disconnect between the statistical view and reality, many of these resulting practical suggestions are misguided and empirical performance suffers accordingly. In this paper we focus on illustrating this phenomenon through simulation experiments.

It is important to note that although this paper deals with "the statistical view of boosting", it is an overgeneralization to imply there is only one single view of boosting in the statistical community. All statisticians are not of a single mindset, and much literature has been produced subsequent to the Friedman, Hastie and Tibshirani *Annals of Statistics* paper. Much of what we categorize as the statistical view of boosting can be found in that original paper, but other ideas, especially those in Sections 3.9, 3.10, 4.9 and 4.10, are attributable to other researchers and subsequent publications in the statistics community. For this reason, we are careful to provide references and direct quotations throughout this paper.

The following section describes the general setting for classification and the AdaBoost algorithm. Sections 3 and 4 consider a collection of practical suggestions, commonly held beliefs and modifications to the AdaBoost algorithm based on the statistical view. For each one, a simulation providing contradictory evidence is included. Section 5 mentions a slightly different set of simulations to consider, and finally Section 6 offers practical advice in light of the evidence presented in this paper as well as some concluding remarks.

2. The Classification Problem and Boosting

In this section we will begin by describing the general problem of classification in statistics and machine learning. Next we will describe the AdaBoost algorithm and give details of our implementation.

2.1 Classification

The problem of classification is an instance of what is known as *supervised learning* in machine learning. We are given training data $x_1,\ldots,x_n$ and $y_1,\ldots,y_n$ where each x_i is a $d-$dimensional vector of predictors $(x_i^{(1)},\ldots,x_i^{(d)})$ and $y_i \in \{-1,+1\}$ is the associated observed class label. To justify generalization, it is usually assumed that the training data are *iid* samples of random variables (X,Y) having some unknown distribution. The goal is to learn a rule $\hat{C}(x)$ that assigns a class label in $\{-1,+1\}$ to any new observation x. The performance of this rule is usually measured with respect to misclassification error, or the rate at which new observations drawn from the same population are incorrectly labelled. Formally we can define the misclassification error for a classification rule $\hat{C}(x)$ as $\mathrm{P}(\hat{C}(X) \neq Y)$.

For any given data set misclassification error can be estimated by reserving a fraction of the available data for test data and then computing the percent of incorrect classifications resulting from the classifier trained on the remainder of the data. Various cross-validation techniques improve upon this scheme by averaging over different sets of test data. In this paper we will consider only examples of simulated data so that the joint distribution of X and Y is known. This will enable us to estimate misclassification error as accurately as desired by simply repeatedly simulating training and test data sets and averaging the misclassification errors from the test sets.

2.2 Boosting

AdaBoost (Freund and Schapire, 1996) is one of the first and the most popular boosting algorithms for classification. The algorithm is as follows. First let $F_0(x_i) = 0$ for all x_i and initialize weights $w_i = 1/n$ for $i = 1,\ldots,n$. Then repeat the following for m from 1 to M:

- Fit the classifier g_m to the training data using weights w_i where g_m maps each x_i to -1 or 1.
- Compute the weighted error rate $\varepsilon_m \equiv \sum_{i=1}^{n} w_i \mathrm{I}[y_i \neq g_m(x_i)]$ and half its log-odds, $\alpha_m \equiv \frac{1}{2}\log\frac{1-\varepsilon_m}{\varepsilon_m}$.
- Let $F_m = F_{m-1} + \alpha_m g_m$.
- Replace the weights w_i with $w_i \equiv w_i e^{-\alpha_m g_m(x_i) y_i}$ and then renormalize by replacing each w_i by $w_i/(\sum w_i)$.

The final classifier is 1 if $F_M > 0$ and -1 otherwise. The popularity of this algorithm is due to a vast amount of empirical evidence demonstrating that the algorithm yields very small misclassification error relative to competing methods. Further, the performance is remarkably insensitive to the choice of the total number of iterations M. Usually any sufficiently large value of M works well. For the simulations in this paper we will take $M = 1000$, with the single exception of the simulation in Section 4.7 where it is instructive to consider $M = 5000$.

Many variations of the AdaBoost algorithm now exist. We will visit some of these in Sections 3 and 4 and compare their performance to the original AdaBoost algorithm. Further, these variations as well as AdaBoost itself are very flexible in the sense that the class of classifiers from which each g_m is selected can be quite general. However, the superior performance of AdaBoost is generally in the context of classification trees. For this reason we will use classification trees in our experiments. Specifically, the trees will be fit using the “rpart” function in the “R” statistical software package (http://www.r-project.org/). The R code for all the experiments run in this paper is available on the web page http://www.davemease.com/contraryevidence.

3. Experiments Which Contradict the Statistical View of Boosting

In this section we describe the results of several experiments based on simulations from the model introduced below. Each experiment is meant to illustrate particular inconsistencies between that which is suggested by the statistical view of boosting and what is actually observed in practice.

For the experiments we will consider in this section we will simulate data from the model

$$\mathrm{P}(Y=1|x) = q + (1-2q)\ \mathrm{I}\left[\sum_{j=1}^{J} x^{(j)} > J/2\right].$$

We will take X to be distributed *iid* uniform on the d-dimensional unit cube $[0,1]^d$. The constants n, d, J and q will be set at different values depending on the experiment. Note that q is the Bayes error and $J \leq d$ is the number of effective dimensions. Recall n is the number of observations used to train the classifier. The unconditional probabilities for each of the two classes are always equal since $\mathrm{P}(Y=1) = \mathrm{P}(Y=0) = 1/2$. The only exceptions to this are experiments in which we take $J = 0$ for which the sum (and thus the indicator) is taken to be always zero. In these cases the model reduces to the “pure noise” model $\mathrm{P}(Y=1|x) \equiv q$ for all x.

3.1 Should Stumps Be Used for Additive Bayes Decision Rules?

Additive models are very popular in many situations. Consider the case in which the Bayes decision rule is additive in the space of the predictors $x^{(1)},\ldots,x^{(d)}$. By this we mean that the Bayes decision rule can be written as the sign of $\sum_{i=1}^{d} h_i(x^{(i)})$ for some functions $h_1,\ldots,h_d$. This is true, for example, for our simulation model. The classification rule produced by AdaBoost is itself necessarily additive in the classifiers g_m. Thus when the g_m are functions of only single predictors the AdaBoost classification rule is additive in the predictor space. For this reason it has been suggested that one should use stumps (2-node trees) if one believes the optimal Bayes rule is approximately additive, since stumps are trees which only involve single predictors and thus yield an additive model in the predictor space for AdaBoost. It is believed that using trees of a larger size will lead to overfitting because it introduces higher-level interactions. This argument is made explicit in Hastie et al. (2001) on pages 323-324 and in Friedman et al. (2000a) on pages 360-361.

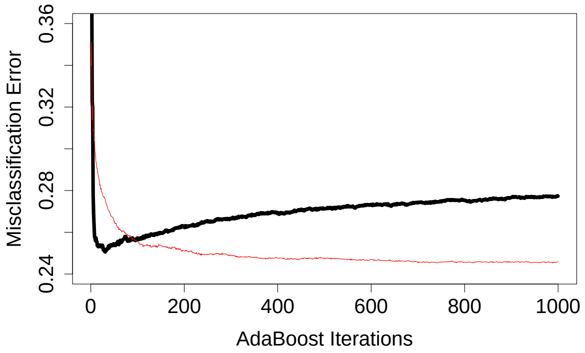

Figure 1: Comparison of AdaBoost with Stumps (Black, Thick) and 8-Node Trees (Red, Thin) for an Additive Bayes Rule

Despite the logic of this argument which is based on the idea that one should use an additive model when fitting an additive function, it can be observed that often, in fact, using larger trees is more effective than using stumps even when the Bayes rule is additive. The reason has to do with the fact that boosting with larger trees actually often overfits *less* than boosting with smaller trees in practice since the larger trees are more orthogonal and a self-averaging process prevents overfitting. We do not endeavor to make this argument rigorous here, but we will provide a compelling example.

For our example we will use our model with a Bayes error rate of $q = 0.1$, a training sample size of $n = 200$ and $d = 20$ dimensions of which $J = 5$ are active. Figure 1 displays the misclassification error of AdaBoost based on hold out samples of size 1000 (also drawn *iid* on $[0,1]^d$) as a function of the iterations. The results are averaged over 100 repetitions of the simulation. While AdaBoost with stumps (thick, black curve) leads to overfitting very early on, AdaBoost with 8-node trees (thin, red curve) does not suffer from overfitting and leads to smaller misclassification error. In fact, the misclassification error by 1000 iterations was smaller for the 8-node trees in 96 of the 100 simulations. The average (paired) difference in misclassification error was 0.031 with a standard error of $0.018/\sqrt{100} = 0.0018$. Also note that both algorithms here perform considerably worse than the Bayes error rate of $q = 0.1$.

The R code for this experiment as well as all others in this paper can be found at http://www.davemease.com/contraryevidence. We encourage the reader to appreciate the reproducibility of the qualitative result by running the code for various values of the parameters q, n, d and J.

It is worth further commenting on the fact that in this simulation AdaBoost with stumps leads to overfitting while AdaBoost with the larger 8-node trees does not, at least by 1000 iterations. This is of special interest since many of the examples other researchers provide to show AdaBoost can in fact overfit often use very small trees such as stumps as the base learner. Some such examples of overfitting can be found in Friedman et al. (2000a), Jiang (2000) and Ridgeway (2000) as well as Leo Breiman's 2002 Wald Lectures on Machine Learning.[1] The belief is that if stumps overfit then so will larger trees since the larger trees are more complex. (Clearly the example presented

[1] Breiman's lecture notes can be found at http://www.stat.berkeley.edu/users/breiman/wald2002-1.pdf.

in this section shows that this is not the case.) To illustrate this viewpoint consider the quote from Jiang (2001) who writes, “all these base systems, even the ones as simple as the ‘stumps’, will unavoidably lead to suboptimal predictions when boosted forever.” Additionally, such examples in which overfitting is observed also often deal with extremely low-dimensional cases such as $d = 2$ or even $d = 1$. By experimenting with the simulation code provided along with this paper one can confirm that in general AdaBoost is much more likely to suffer from overfitting in trivial low-dimensional examples as opposed to high-dimensional situations where it is more often used.

3.2 Should Smaller Trees Be Used When the Bayes Error is Larger?

Similar arguments to those in the previous section suggest that it is necessary to use smaller trees for AdaBoost when the Bayes error is larger. The reasoning is that when the Bayes error is larger, the larger trees lead to a more complex model which is more susceptible to overfitting noise. However, in practice we can often observe the opposite to be true. The higher Bayes error rate actually can favor the larger trees. This counterintuitive result may be explained by the self-averaging which occurs during the boosting iterations as discussed by Krieger et al. (2001). Conversely, the smaller trees often work well for lower Bayes error rates, provided they are rich enough to capture the complexity of the signal.

We illustrate this phenomenon by re-running the experiment in the previous section, this time using $q = 0$, which implies the Bayes error is zero. The average misclassification error over the 100 hold out samples is displayed in the top panel of Figure 2. It can now be observed that AdaBoost with stumps performs better than AdaBoost with 8-node trees. In fact, this was the case in 81 out of the 100 simulations (as opposed to only 4 of the 100 for $q = 0.1$ from before). The mean difference in misclassification error after 1000 iterations was 0.009 with a standard error of $0.011/\sqrt{100} = 0.0011$. The bottom panel of Figure 2 confirms that AdaBoost with stumps outperforms AdaBoost with 8-node tress only for very small values of q with this simulation model.

3.3 Should LogitBoost Be Used Instead of AdaBoost for Noisy Data?

The LogitBoost algorithm was introduced by Friedman et al. (2000a). The algorithm is similar to AdaBoost, with the main difference being that LogitBoost performs stagewise minimization of the negative binomial log likelihood while AdaBoost performs stagewise minimization of the exponential loss. By virtue of using the binomial log likelihood instead of the exponential loss, the LogitBoost algorithm was believed to be more “gentle” and consequently likely to perform better than AdaBoost for classification problems in which the Bayes error is substantially larger than zero. For instance, on page 309 Hastie et al. (2001) write, “it is therefore far more robust in noisy settings where the Bayes error rate is not close to zero, and especially in situations where there is misspecification of the class labels in the training data.”

Despite such claims, we often observe the opposite behavior. That is, when the Bayes error is not zero, LogitBoost often overfits while AdaBoost does not. As an example, we consider the performance of AdaBoost and LogitBoost on the simulation from Section 3.1 in which the Bayes error was $q = 0.1$. The base learners used are 8-node trees. Figure 3 displays the performance averaged over 100 hold out samples. It is clear that LogitBoost (blue, thick) begins to overfit after about 200 iterations while AdaBoost (red, thin) continues to improve. After 1000 iterations the mean difference was 0.031 with a standard error of $0.017/\sqrt{100}$=0.0017. The misclassification error for LogitBoost at 1000 iterations was larger than that of AdaBoost in all but 4 of the 100 simulations.

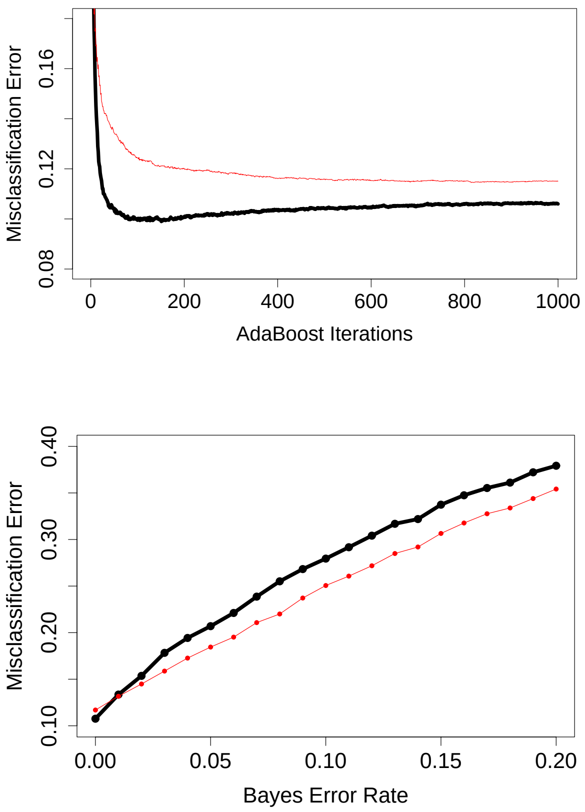

Figure 2: Comparison of AdaBoost with Stumps (Black, Thick) and 8-Node Trees (Red, Thin) for an Additive Bayes Rule. Top Panel: Misclassification Error for Zero Bayes Error as a Function of the Iterations. Bottom Panel: Misclassification Error at 1000 Iterations as a Function of the Bayes Error Rate *q*.

Other examples of this phenomenon of LogitBoost overfitting noisy data when AdaBoost does not can be found in Mease et al. (2007).

The R code used for LogitBoost was written by Marcel Dettling and Peter Buhlmann and can be found at http://stat.ethz.ch/∼dettling/boosting.html. Two small modifications were made to the code in order to fit 8-node trees, as the original code was written for stumps.

It should be noted that LogitBoost differs from AdaBoost not only in the loss function which it minimizes, but also in the Newton style minimization that it employs to carry out the minimization. For this reason it would be of interest to examine the performance of the algorithm in Collins et al. (2000) which minimizes the negative binomial log likelihood in a manner more analogous to AdaBoost. We do not consider that algorithm in this paper since our focus is mainly on the work of Friedman et al. (2000a) and the implications in the statistical community.

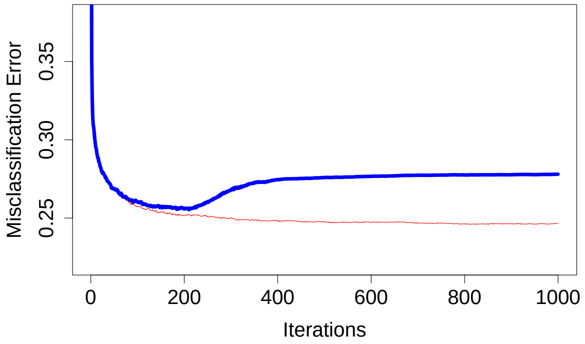

Figure 3: Comparison of AdaBoost (Red, Thin) and LogitBoost (Blue, Thick) with 8-Node Trees

3.4 Should Early Stopping Be Used to Prevent Overfitting?

In order to prevent overfitting, one popular regularization technique is to stop boosting algorithms after a very small number of iterations, such as 10 or 100. The statistics community has put a lot of emphasis on early stopping as evidenced by the large number of papers on this topic. For example, the paper "Boosting with Early Stopping: Convergence and Consistency" by Zhang and Yu (2005) tells readers that "boosting forever can overfit the data" and that "therefore in order to achieve consistency, it is necessary to stop the boosting procedure early." Standard implementations of boosting such as the popular gbm package for R by Ridgeway (2005) implement data-derived early stopping rules.

The reasoning behind early stopping is that after enough iterations have occurred so that the complexity of the algorithm is equal to the complexity of the underlying true signal, then any additional iterations will lead to overfitting and consequently larger misclassification error. However, in practice we can often observe that additional iterations beyond the number necessary to match the complexity of the underlying true signal actually reduce the overfitting that has already occurred rather than causing additional overfitting. This is likely due to the self-averaging property of AdaBoost to which we eluded earlier.

To illustrate this we use a somewhat absurd example. We take $J = 0$ in our simulation model, so that there is no signal at all, only noise. We have $\mathrm{P}(Y = 1|x) \equiv q$ so that Y does not depend on x in any way. We take a larger sample size of $n = 5000$ this time, and also use larger $2^8 = 256$-node trees. The experiment is again averaged over 100 repetitions, each time drawing the $n = 5000$ x values from $[0,1]^d$ with $d = 20$. The 100 hold out samples are also drawn from $[0,1]^{20}$ each time. The Bayes error rate is $q = 0.2$.

Since there is no signal to be learned, we can observe directly the effect of AdaBoost's iterations on the noise. We see in Figure 4 that early on there is some overfitting, but this quickly goes away and the misclassification error decreases and appears to asymptote very near the Bayes error rate of $q = 0.2$. In fact, the final average after 1000 iterations (to three decimals accuracy) is 0.200 with a standard error of $0.013/\sqrt{100}$=0.0013. Even more interesting, the misclassification error after 1000 iterations is actually less than that after the first iteration (i.e., the misclassification error for a single 2^8-node tree). The mean difference between the misclassification error after one iteration and that after 1000 iterations was 0.012 with a standard error of $0.005/\sqrt{100}$=0.0005. The difference was

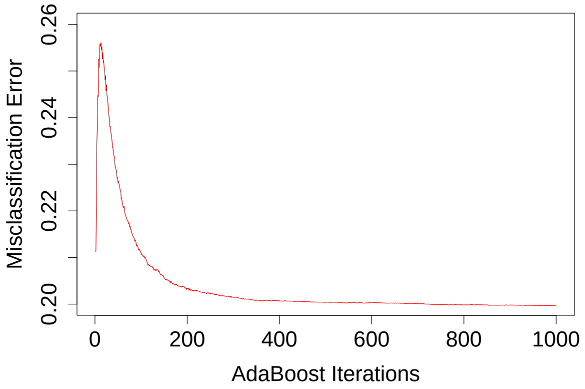

Figure 4: AdaBoost on 20% Pure Noise

positive in 99 of the 100 repetitions. Thus we see that not only does AdaBoost resist overfitting the noise, it actually fits a classification rule that is less overfit than its own 2^8-node tree base classifier.

3.5 Should Regularization Be Based on the Loss Function?

Since the statistical view of boosting centers on the stagewise minimization of a certain loss function on the training data, a common suggestion is that regularization should be based on the behavior of that loss function on a hold out or cross-validation sample. For example, the implementation of the AdaBoost algorithm in the gbm package (Ridgeway, 2005) uses the exponential loss $\sum_{i=1}^{n} e^{-y_i F_m(x_i)}$ to estimate the optimal stopping time. Indeed, if early stopping is to be used as regularization, the statistical view would suggest stopping when this exponential loss function begins to increase on a hold out sample. However, in practice the misclassification error often has little to do with the behavior of the exponential loss on a hold out sample. To illustrate this, we return to the experiment in Section 3.1. If we examine the exponential loss on hold out samples for AdaBoost with the 8-node trees, it can be seen that this loss function is exponentially increasing throughout the 1000 iterations. This is illustrated in Figure 5 which shows the linear behavior of the log of the exponential loss for a single repetition from this experiment on a hold out sample of size 1000. Thus, early stopping regularization based on the loss function would suggest stopping after just one iteration, when in fact Figure 1 shows we do best to run the 8-node trees for the full 1000 iterations. This behavior has also been noted for LogitBoost as well (with respect to the negative log likelihood loss) in Mease et al. (2007) and in Dettling and Buhlmann (2003). In the latter reference the authors estimated a stopping parameter for the number of iterations using cross-validation but observed that they “could not exploit significant advantages of estimated stopping parameters” over allowing the algorithm to run for the full number of iterations (100 in their case).

3.6 Should the Collection of Basis Functions Be Restricted to Prevent Overfitting?

Another popular misconception about boosting is that one needs to restrict the class of trees in order to prevent overfitting. The idea is that if AdaBoost is allowed to use *all* 8-node trees for instance, then the function class becomes too rich giving the algorithm too much flexibility which

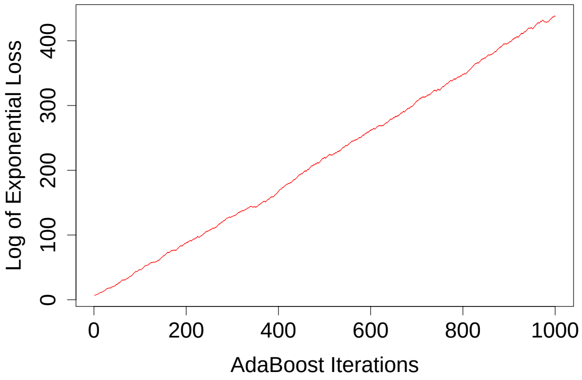

Figure 5: The Log of the Exponential Loss for AdaBoost on a Hold Out Sample

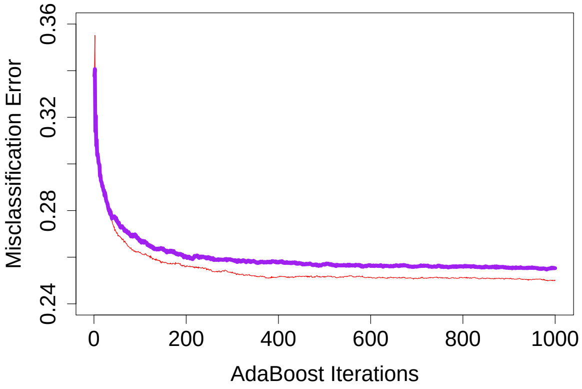

Figure 6: Comparison of AdaBoost with 8-Node Trees (Red, Thin) to AdaBoost with 8-Node Trees Restricted to Have at Least 15 Observations in Each Terminal Node (Purple, Thick)

leads to overfitting. This line of thinking gives rise to various methods for restricting or regularizing the individual trees themselves as a method of regularizing the AdaBoost algorithm. For instance, the implementation of AdaBoost in the gbm code (Ridgeway, 2005) has a parameter called "n.minobsinnode" which is literally the minimum number of observations in the terminal nodes of the trees. The default of this value is not 1, but 10.

In spite of this belief, it can be observed that the practice of limiting the number of observations in the terminal nodes will often degrade the performance of AdaBoost. It is unclear why this happens; however, we note that related tree ensemble algorithms such as PERT (Cutler and Zhao, 2001) have demonstrated success by growing the trees until only a single observation remains in each terminal node.

As an example of this performance degradation, we again revisit the simulation in Section 3.1 and compare the (unrestricted) 8-node trees used there to 8-node trees restricted to have at least 15

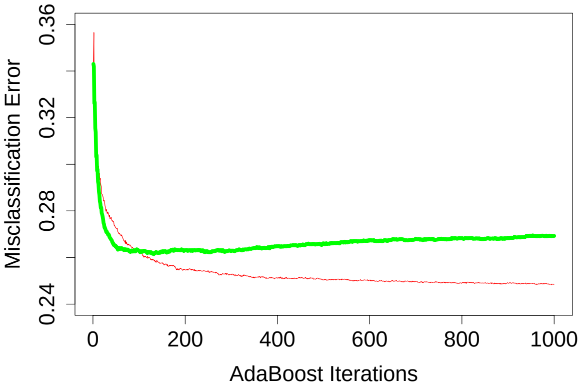

Figure 7: Comparison of AdaBoost (Red, Thin) and AdaBoost with Shrinkage (Green, Thick)

observations in each terminal node. (This is done in R by using the option “minbucket=15” in the “rpart.control” syntax.) Figure 6 shows the results with the unrestricted 8-node trees given by the red (thin) curve and the restricted 8-node trees given by the purple (thick) curve. The degradation in performance is evident, although not extremely large. The mean difference in misclassification error at 1000 iterations was 0.005 with a standard error of $0.010/\sqrt{100}$=0.001. AdaBoost with unrestricted 8-node trees gave a lower misclassification error in 67 of the 100 repetitions.

3.7 Should Shrinkage Be Used to Prevent Overfitting?

Shrinkage is yet another form of regularization that is often used for boosting algorithms. In the context of AdaBoost, shrinkage corresponds to replacing the α_m in the update formula $F_m = F_{m-1} + \alpha_m g_m$ by $\nu\alpha_m$ where ν is any positive constant less than one. The value $\nu = 0.1$ is popular. In the statistical view of boosting, shrinkage is thought to be extremely important. It is believed to not only reduce overfitting but also to increase the maximum accuracy (i.e., the minimum misclassification error) over the iterations. For instance, Friedman et al. (2000b) write, “the evidence so far indicates that the smaller the value of ν, the higher the overall accuracy, as long as there are enough iterations.”

Despite such claims, it can be observed that shrinkage often does not improve performance and instead can actually cause AdaBoost to overfit in situations where it otherwise would not. To understand why this happens one needs to appreciate that it is the suboptimal nature of the stagewise fitting of AdaBoost that helps it to resist overfitting. Using shrinkage can destroy this resistance. For an example, we again revisit the simulation in Section 3.1 using the 8-node trees. In Figure 7 the red (thin) curve corresponds to the misclassification error for the 8-node trees just as in Section 3.1 and the green (thick) curve now shows the effect of using a shrinkage value of $\nu = 0.1$. It is clear that the shrinkage causes overfitting in this simulation. By 1000 iterations shrinkage gave a larger misclassification error in 95 of the 100 repetitions. The mean difference in misclassification error at 1000 iterations was 0.021 with a standard error of $0.012/\sqrt{100}$=0.0012.

3.8 Is Boosting Estimating Probabilities?

The idea that boosting produces probability estimates follows directly from the statistical view through the stagewise minimization of the loss function. Specifically, the exponential loss $\sum_{i=1}^{n} e^{-y_i F_m(x_i)}$, which is minimized at each stage by AdaBoost, achieves its minimum when the function $F_m(x)$ relates to the true conditional class probabilities $p(x) \equiv \mathrm{P}(Y=1|x)$ by the formula $F_m(x) = \frac{1}{2}\log\frac{p(x)}{1-p(x)}$. This leads to the estimator of $p(x)$ after m iterations given by

$$\hat{p}_m(x) = 1/(1+e^{-2F_m(x)}).$$

This relationship between the score function F_m in AdaBoost and conditional class probabilities is given explicitly in Friedman et al. (2000a). An analogous formula is also given for obtaining probability estimates from LogitBoost. Standard implementations of boosting such as Dettling and Buhlmann's LogitBoost code at http://stat.ethz.ch/∼dettling/boosting.html as well as the gbm LogitBoost code by Ridgeway (2005) output conditional class probabilities estimates directly.

Despite the belief that boosting is estimating probabilities, the estimator $\hat{p}_m(x)$ given above is often extremely overfit in many cases in which the classification rule from AdaBoost shows no signs of overfitting and performs quite well. An example is given by the experiment in Section 3.1. In Figure 1 we saw that the classification rule using 8-node trees performed well and did not overfit even by 1000 iterations. Conversely, the probability estimates are severely overfit early on. This is evidenced by the plot of the exponential loss in Figure 5. In this context the exponential loss can be thought of as an estimate of a *probability scoring rule* which quantifies the average disagreement between a true probability p and an estimate $\hat{p}$ using only binary data (Buja et al., 2006). For the exponential loss the scoring rule is $p\sqrt{(1-\hat{p})/\hat{p}} + (1-p)\sqrt{\hat{p}/(1-\hat{p})}$. The fact that the plot in Figure 5 is increasing shows that the probabilities become worse with each iteration as judged by this scoring rule. Similar behavior can be seen using other scoring rules such as the squared loss $(p-\hat{p})^2$ and the log loss $-p\log\hat{p} - (1-p)\log(1-\hat{p})$ as shown in Mease et al. (2007). This reference also shows the same behavior for the probability estimates from LogitBoost, despite the fact that efficient probability estimation is the main motivation for the LogitBoost algorithm.

The reason for the overfitting of these probability estimators is that as more and more iterations are added to achieve a good classification rule, the value of $|F_m|$ at any point is increasing quickly. The classification rule only depends on the sign of F_m and thus is not affected by this. However, this increasing tendency of $|F_m|$ impacts the probability estimates by causing them to quickly diverge to 0 and 1. Figure 8 shows the probability estimates $\hat{p}_m(x_i) = 1/(1+e^{-2F_m(x_i)})$ for AdaBoost from a single repetition of the experiment in Section 3.1 using 8-node trees on a hold out sample of size 1000. The top histogram corresponds to $m = 10$ iterations and the bottom histogram shows $m = 1000$ iterations. The histograms each have 100 equal width bins. It can be seen that after only 10 iterations almost all of the probability estimates are greater than 0.99 or less than 0.01, and even more so by 1000 iterations. This indicates a poor fit since we know all of the true probabilities are either 0.1 or 0.9.

Other researchers have also noted this type of overfitting with boosting and have used it as an argument in favor of regularization techniques. For instance, it is possible that using a regularization technique such as shrinkage or the restriction to stumps as the base learners in this situation could produce better probability estimates. However, from what we have seen of some regularization techniques in this paper, we know that regularization techniques often severely degenerate the classification performance of the algorithm. Furthermore, some are not effective at all in many

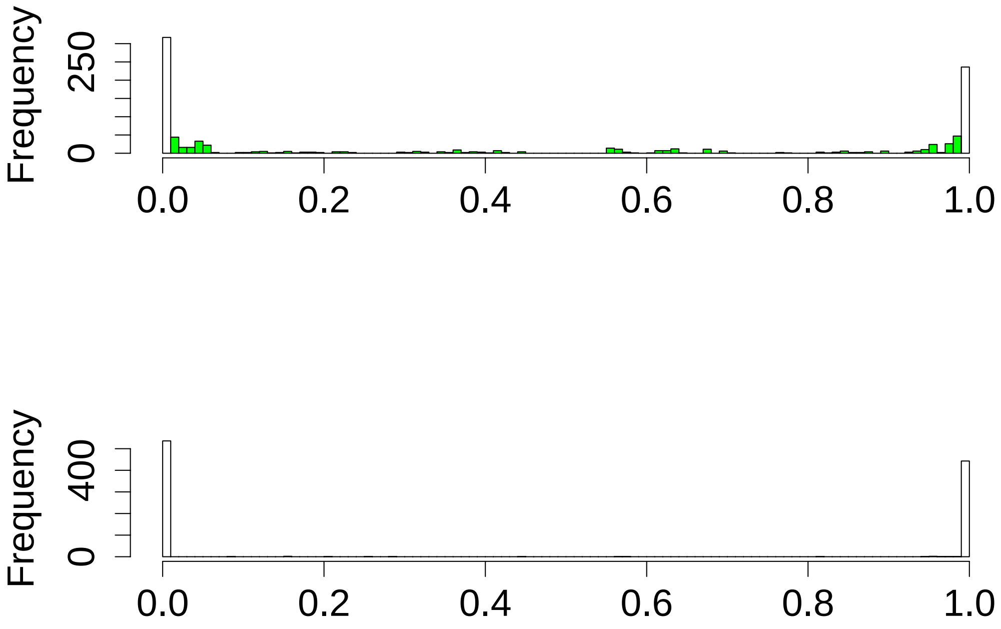

Figure 8: Probability Estimates From AdaBoost at $m = 10$ Iterations (Top) and $m = 1000$ Iterations (Bottom)

situations. For instance, early stopping, one of the most popular regularization techniques, is of little help when the probabilities overfit from the outset as in Figure 5. For a technique that achieves conditional probability estimation using AdaBoost without modification or regularization the reader should see Mease et al. (2007).

3.9 Is Boosting Similar to the One Nearest Neighbor Classifier?

In all the experiments considered in this paper, AdaBoost achieves zero misclassification error on the training data. This characteristic is quite typical of AdaBoost and has led some researchers to draw parallels to the (one) nearest neighbor classifier, a classifier which necessarily also yields zero misclassification error on the training data. This characteristic has also been suggested as a reason why AdaBoost will overfit when the Bayes error is not zero.

The belief in a similarity between boosting and the nearest neighbor classifier was not expressed in the original paper of Friedman et al. (2000a), but rather has been expressed more recently in the statistics literature on boosting by authors such as Wenxin Jiang in papers such as Jiang (2000), Jiang (2001) and Jiang (2002). In Jiang (2000), the equivalence between AdaBoost and the nearest neighbor classifier is established only for the case of $d = 1$ dimension. In the $d = 1$ case, the equivalence is merely a consequence of fitting the training data perfectly (and following Jiang's convention of using midpoints of the training data for the classification tree splits). However, as we will see from the experiment in this section, the behavior of AdaBoost even in $d = 2$ dimensions is radically different from the nearest neighbor rule.

Despite this difference, Jiang goes on to suggest that the performance of AdaBoost in higher dimensions might be similar to the case of $d = 1$ dimension. For instance in “Is Regularization Unnecessary for Boosting?” Jiang (2001) writes, “it is, however, plausible to conjecture that even in the case of higher dimensional data running AdaBoost forever can still lead to a suboptimal prediction which does not perform much better than the nearest neighbor rule.” Further, Jiang (2002) writes, “the fit will be perfect for almost all sample realizations and agree with the nearest neighbor rule at all the data points as well as in some of their neighborhoods” and that “the limiting prediction presumably cannot perform much better than the nearest neighbor rule.”

To understand why equivalent behavior on the training data (or “data points” using Jiang’s terminology above) does not imply similar performance for classification rules for $d > 1$, it is important to remember that in the case of continuous data the training data has measure zero. Thus the behavior on the training data says very little about the performance with respect to the population. This is well illustrated by the pure noise example from Section 3.4. For any point in the training data for which the observed class differs from the class given by the Bayes rule, both AdaBoost and nearest neighbor will classify this point as the observed class and thus disagree with the Bayes rule. However, the volume of the affected neighborhood surrounding that point can be arbitrarily small with AdaBoost, but will necessarily be close to $1/n$ of the total volume with nearest neighbor.

To help the reader visualize this, we consider a $d = 2$-dimensional version of the pure noise example from Section 3.4. We again use a Bayes error rate of $q = 0.2$ but now take only $n = 200$ points spread out evenly according to a Latin hypercube design. The left plot in Figure 9 shows the resulting classification of AdaBoost using 8-node trees after 1000 iterations and the right plot shows the rule for nearest neighbor. The training points with $Y = -1$ are colored black and those with $Y = +1$ are colored yellow. Regions classified as -1 are colored purple and those classified as $+1$ are colored light blue. Since the Bayes rule is to classify the entire area as -1, we can measure the overfitting of the rules by the fraction of the total area colored as light blue. The nearest neighbor classifier has 20% of the region colored as light blue (as expected), while AdaBoost has only 16%. The two classifiers agree “at all the [training] data points as well as in some of their neighborhoods” as stated by Jiang, but the “some” here is relatively small.

In higher dimensions this effect is even more pronounced. For the $d = 20$-dimensional example from Section 3.4 the area (volume) of the light blue region would be essentially zero for AdaBoost (as evidenced by its misclassification error rate matching almost exactly that of the Bayes error), while for nearest neighbor it remains at 20% as expected. Thus we see that the nearest neighbor classifier differs from the Bayes rule for 20% of the points in both the training data and the population while AdaBoost differs from the Bayes rule for 20% of the points in the training data but virtually none in the population.

The differences between the nearest neighbor classifier and AdaBoost are obvious in the other experiments in this paper as well. For instance, for the experiment in Section 3.1 the nearest neighbor classifier had an average misclassification error rate of 0.376 versus 0.246 for AdaBoost with the 8-node trees.

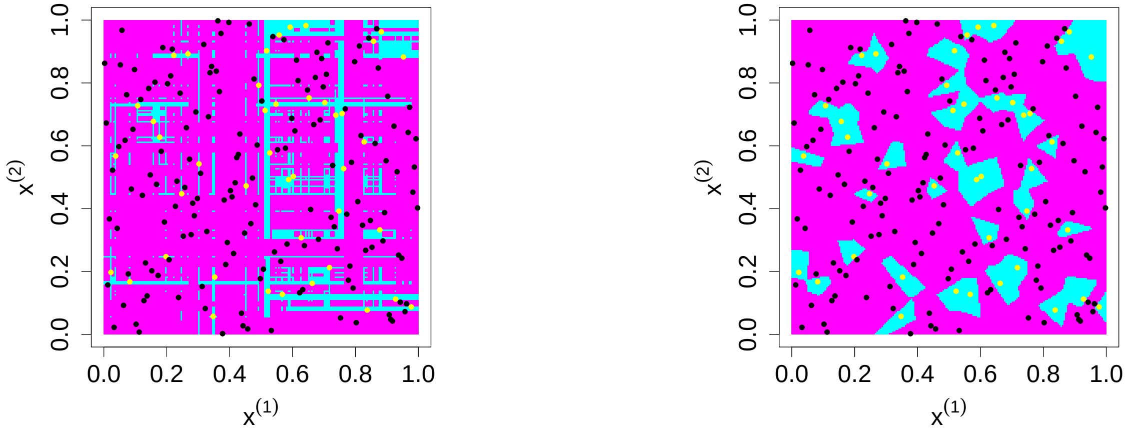

Figure 9: Comparison of AdaBoost (Left) and Nearest Neighbor (Right) on 20% Pure Noise

3.10 Is Boosting Consistent?

An important question to ask about any estimator is whether or not it is consistent. A consistent estimator is defined to be any estimator for which the estimated quantity converges in probability to the true quantity. In our context, to ask if AdaBoost is a consistent estimator is to ask if its classification rule converges in probability to the Bayes rule. If it is consistent, then with a sufficiently large training sample size n its misclassification error will come arbitrarily close to the Bayes error.

The belief in the statistics community is that AdaBoost is not consistent unless regularization is employed. The main argument given is that if AdaBoost is left unregularized it will eventually fit all the data thus making consistency impossible as with the nearest neighbor classifier. Consequently, all work on the consistency of boosting deals with regularized techniques. While we have noted in Section 3.9 that it is characteristic of AdaBoost to achieve zero misclassification error on the training data, we have also discussed the fact that this in no way determines its performance in general, as the training data has measure zero in the case of continuous data. In fact in Section 3.4 we observed that with a sample size of $n = 5000$ AdaBoost with 2^8-node trees achieved the Bayes error rate to three decimals on a 20% pure noise example despite fitting all the training data.

In this section we consider a simulation with this same sample size and again 2^8-node trees but we now include a signal in addition to the noise. We take $J = 1$ and use $d = 5$ dimensions and fix the Bayes error rate at $q = 0.1$. The resulting misclassification error rate averaged over 100 repetitions each with a hold out sample of size 1000 is shown in Figure 10. As before, AdaBoost fits all the training data early on, but the misclassification error after 1000 iterations averages only 0.105 with a standard error of $0.010/\sqrt{100}$=0.001. This is quite close to the Bayes error rate $q = 0.1$ and can be observed to come even closer by increasing the sample size. It should also be noted that this error rate is much below the limit of $2q(1-q) = 0.18$ that holds for the nearest neighbor classifier in this case.

The belief that unregularized AdaBoost can not be consistent is promoted by Wenxin Jiang's work mentioned in Section 3.9 connecting the performance of AdaBoost and the nearest neighbor classifier. His result for $d = 1$ rules out consistency in that case since the nearest neighbor rule is

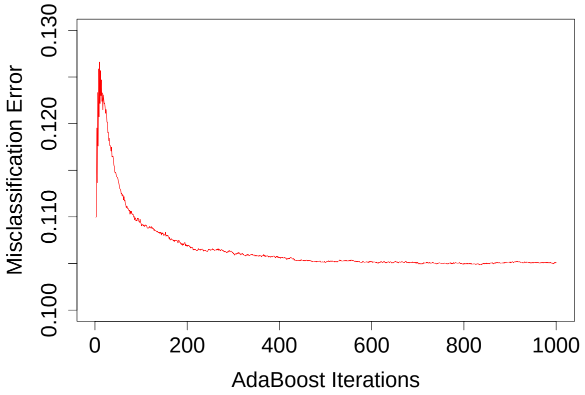

Figure 10: Performance of AdaBoost for a Simulation with a Bayes Error of 0.1

not consistent, but nothing is established for $d > 1$ with regard to AdaBoost. Jiang (2002) admits this when he writes, "what about boosting forever with a higher dimensional random continuous predictor x with $\dim(x) > 1$? We do not have theoretical results on this so far."

4. More Experiments Which Contradict the Statistical View of Boosting

In this section we revisit the experiments from Section 3 using a different simulation model. The purpose here is to show that the results are reproducible and do not depend on a particular simulation model. We also encourage readers to experiment with other simulation models by modifying the code provided on the web page.

The simulations in this section will use the model

$$P(Y=1|x) = \begin{cases} q & x^{(1)} \in [0,0.1) \cup [0.2,0.3) \cup [0.4,0.5) \cup [0.6,0.7) \cup [0.8,0.9) \\ 1-q & x^{(1)} \in [0.1,0.2) \cup [0.3,0.4) \cup [0.5,0.6) \cup [0.7,0.8) \cup [0.9,1]. \end{cases}$$

We will rerun each experiment from Section 3 using this model. Throughout this section we will use $d = 20$ dimensions and take X to be distributed *iid* uniform on the 20-dimensional unit cube $[0,1]^{20}$. For each experiment we will use twice the sample size of the analogous experiment in Section 3 and the same Bayes error q. The single exception will be the experiment in Section 4.9 in which we use a Bayes error of $q = 0.1$ and $d = 2$ dimensions for visualization purposes.

Note that while the experiments in Section 3 had $J \leq d$ effective dimensions, the experiments in this section will all have only one effective dimension as a result of this simulation model. The plots in Figure 19 are useful for visualizing this model in $d = 2$ dimensions.

4.1 Should Stumps Be Used for Additive Bayes Decision Rules?

As in Section 3.1 we use a Bayes error rate of $q = 0.1$ and $d = 20$ dimensions. We use the new simulation model with a training sample size of $n = 400$. Figure 11 displays the misclassification error of AdaBoost based on hold out samples of size 1000 (also drawn *iid* on $[0,1]^{20}$) as a function of the iterations. The results are again averaged over 100 repetitions of the simulation.

As in Section 3.1, Adaboost with 8-node trees (thin, red curve) does not show any signs of overfitting while AdaBoost with stumps (thick, black curve) leads to overfitting. The overfitting

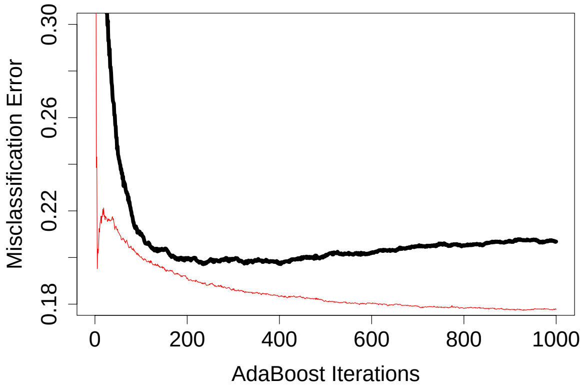

Figure 11: Comparison of AdaBoost with Stumps (Black, Thick) and 8-Node Trees (Red, Thin) for an Additive Bayes Rule

is evident in this experiment after about 400 iterations. Furthermore, AdaBoost with 8-node trees outperforms AdaBoost with stumps throughout the entire 1000 iterations. The misclassification error by 1000 iterations was smaller for the 8-node trees in 93 of the 100 simulations. The average (paired) difference in misclassification error after 1000 iterations was 0.029 with a standard error of $0.018/\sqrt{100} = 0.0018$. As before, since the simulation model used here has an additive Bayes decision rule, this evidence is directly at odds with the recommendation in Hastie et al. (2001) and Friedman et al. (2000a) that stumps are preferable for additive Bayes decision rules.

4.2 Should Smaller Trees Be Used When the Bayes Error is Larger?

As in Section 3.2, we observe that when we decrease the Bayes error rate from $q = 0.1$ to $q = 0$, the 8-node trees no longer have an advantage over the stumps. Figure 12 displays the results of the simulation in Section 4.1 using a Bayes error rate of $q = 0$. We see that the advantage of the 8-node trees has completely disappeared, and now the 8-node trees and stumps are indistinguishable. By 1000 iterations the misclassification errors for both are identical in all of the 100 repetitions.

Thus we see that the advantage of the larger trees in Section 4.1 is a result of the non-zero Bayes error, again suggesting that larger trees are in some way better at handling noisy data. This directly contradicts the conventional wisdom that boosting with larger trees is more likely to overfit on noisy data than boosting with smaller trees.

4.3 Should LogitBoost Be Used Instead of AdaBoost for Noisy Data?

We now rerun the experiment in Section 4.1 using AdaBoost and LogitBoost both with 8-node trees. Figure 13 displays the results with AdaBoost in red (thin) and LogitBoost in blue (thick). While LogitBoost performs better early on, it eventually suffers from overfitting near 400 iterations while AdaBoost shows no overfitting. Furthermore, the misclassification error for AdaBoost after 1000 iterations is (slightly) lower than the minimum misclassification error achieved by LogitBoost. After 1000 iterations the mean difference in misclassification error between LogitBoost and AdaBoost

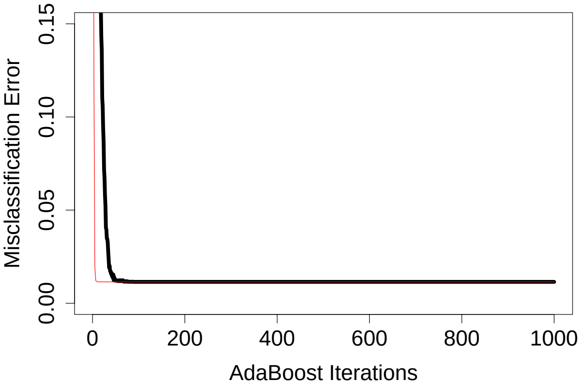

Figure 12: Comparison of AdaBoost with Stumps (Black, Thick) and 8-Node Trees (Red, Thin) for an Additive Bayes Rule with Zero Bayes Error

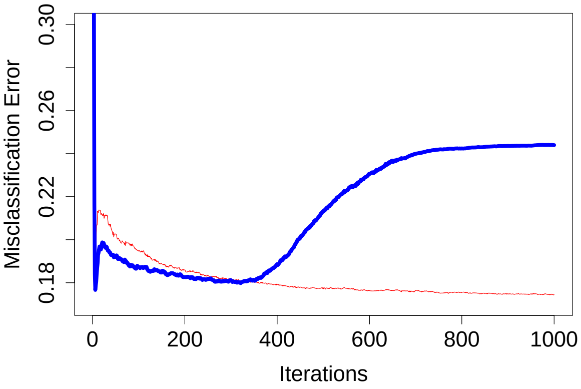

Figure 13: Comparison of AdaBoost (Red, Thin) and LogitBoost (Blue, Thick) with 8-Node Trees

was 0.069 with a standard error of $0.021/\sqrt{100}$=0.0021. The misclassification error for LogitBoost at 1000 iterations was larger than that of AdaBoost in all of the 100 repetitions.

Thus we again see that although LogitBoost was invented to perform better than AdaBoost for data with non-zero Bayes error, LogitBoost actually overfits the data while AdaBoost does not.

4.4 Should Early Stopping Be Used to Prevent Overfitting?

In this section we repeat the simulation from Section 3.4 using the new simulation model. Just as in Section 3.4 we use large $2^8 = 256$-node trees, a Bayes error rate of $q = 0.2$ and $d = 20$ dimensions. We now take twice the training sample size of Section 3.4 so that we have $n = 10,000$ points.

Figure 14 shows the resulting misclassification error averaged over 100 repetitions for hold out samples of size 1000. Although there is overfitting early on, the best performance is again achieved

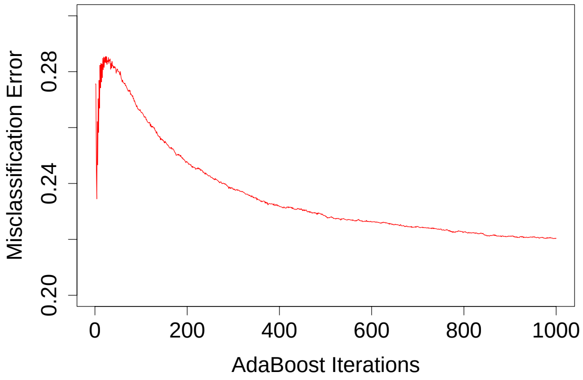

Figure 14: AdaBoost with 20% Bayes Error Using 256-Node Trees

by running the algorithm for the full 1000 iterations. We note that conventional early stopping rules here would be especially harmful since they would stop the algorithm after only a few iterations when the overfitting first takes place. Consequently any such early stopping rule would miss the optimal rule of running for the full 1000 iterations.

It should also be noted that the $2^8 = 256$-node trees used here are much richer than needed to fit the simple one-dimensional Bayes decision rule for this simulation model. Despite this, the misclassification error after 1000 iterations was lower than the misclassification error after the first iteration in all 100 of the reptitions. Thus it is again the self-averaging property of boosting that improves the performance as more and more iterations are run. Early stopping in this example would destroy the benefits of this property.

4.5 Should Regularization Be Based on the Loss Function?

As discussed in Section 3.5, regularization techniques for boosting such as early stopping are often based on minimizing a loss function such as the exponential loss in the case of AdaBoost. However, the performance of AdaBoost with regard to misclassification loss often has very little to do with the exponential loss function in practice.

In this section we examine the exponential loss for the experiment in Section 4.1 using 8-node trees. Figure 15 shows the increasing linear behavior for the log of the exponential loss for a single repetition of this experiment with a hold out sample of size 1000. Thus, just as in Section 3.5, the exponential loss increases exponentially as more iterations are run, while the misclassification error continues to decrease. Choosing regularization to minimize the exponential loss is again not useful for minimizing the misclassification error.

4.6 Should the Collection of Basis Functions Be Restricted to Prevent Overfitting?

In Section 3.6 we saw that restricting the number of observations in the terminal nodes of the trees to be at least 15 degraded the performance of AdaBoost, despite the common belief that such restrictions should be beneficial. In this section we rerun the experiment in Section 4.1 but again consider this same restriction.

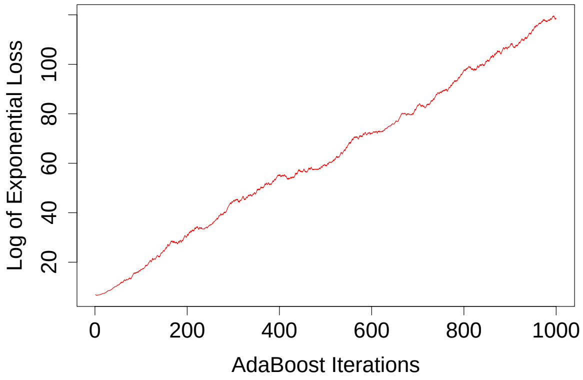

Figure 15: The Log of the Exponential Loss for AdaBoost on a Hold Out Sample

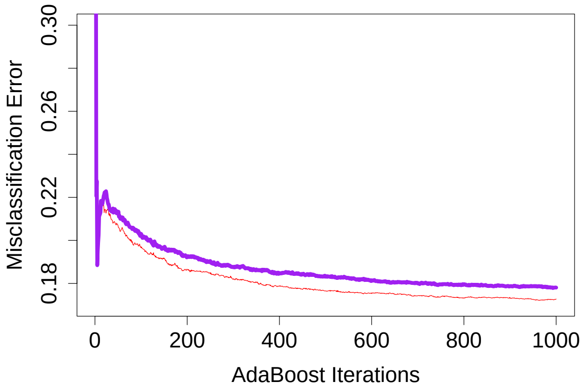

Figure 16: Comparison of AdaBoost with 8-Node Trees (Red, Thin) to AdaBoost with 8-Node Trees Restricted to Have at Least 15 Observations in the Terminal Nodes (Purple, Thick)

Figure 16 shows the results with the unrestricted 8-node trees given by the red (thin) curve and the 8-node trees restricted to have at least 15 observations in the terminal nodes given by the purple (thick) curve. As in Section 3.6, degradation in performance is evident. The mean difference in misclassification error at 1000 iterations was 0.005 with a standard error of $0.010/\sqrt{100}$=0.001. AdaBoost with unrestricted 8-node trees gave a lower misclassification error at 1000 iterations in 65 of the 100 repetitions for this simulation model.

4.7 Should Shrinkage Be Used to Prevent Overfitting?

In Section 3.7 we saw that shrinkage actually caused AdaBoost to overfit in a situation where it otherwise would not have, in spite of the popular belief that shrinkage prevents overfitting. In this section we rerun the experiment in Section 4.1 with 8-node trees again using a shrinkage value of

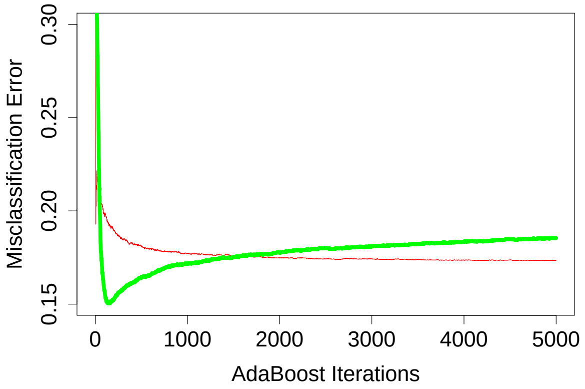

Figure 17: Comparison of AdaBoost (Red, Thin) and AdaBoost with Shrinkage (Green, Thick)

$\nu = 0.1$. Figure 17 shows the results with the red (thin) curve corresponding to no shrinkage and the green (thick) curve showing the results for shrinkage. The plot shows that again shrinkage causes overfitting.

It is interesting to note that in this simulation, unlike the simulation in Section 3.7, shrinkage has the beneficial effect of producing a lower misclassification error very early on in the process, despite causing the eventual overfitting. This suggests that a stopping rule which could accurately estimate the optimal number of iterations combined with shrinkage may prove very effective for this particular simulation. As a result of the good performance early on, the shrinkage actually gives a lower misclassification error at our chosen stopping point of 1000 iterations than without the shrinkage. However, if we run for enough iterations (the plot shows 5000 iterations) the overfitting caused by the shrinkage eventually overwhelms this advantage. By 5000 iterations the shrinkage leads to a larger misclassification error in 87 of the 100 repetitions. The mean difference in misclassification error at 5000 iterations was 0.012 with a standard error of $0.012/\sqrt{100}$=0.0012.

4.8 Is Boosting Estimating Probabilities?

In Section 3.8 we saw that the probability estimates suggested by Friedman et al. (2000a) for AdaBoost diverge quickly to 0 and 1 and consequently perform very poorly even for cases where the AdaBoost classification rule performs well. In this section we examine the probability estimates for a single repetition of the experiment in Section 4.1 on a hold out sample of size 1000.

The two histograms in Figure 18 show the resulting probability estimates for $m = 10$ iterations and $m = 1000$ iterations respectively using 8-node trees. Both histograms have 100 equal width bins. At 10 iterations the estimates have not yet diverged, but by 1000 iterations almost all of the probability estimates are greater than 0.99 or less than 0.01, just as we saw in Section 3.8. As before, this indicates a poor fit since with this simulation model all of the true probabilities are either 0.1 or 0.9.

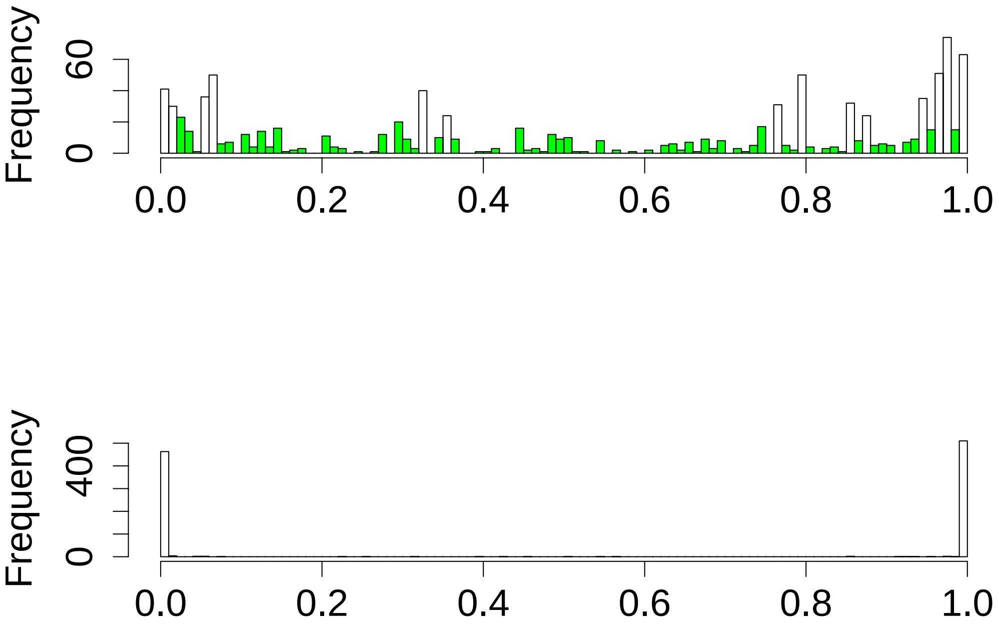

Figure 18: The Probability Estimates From AdaBoost at $m = 10$ Iterations (Top) and $m = 1000$ Iterations (Bottom)

4.9 Is Boosting Similar to the One Nearest Neighbor Classifier?

In Section 3.9 we saw that despite the fact that boosting agrees with the nearest neighbor classifier on all the training data, its performance elsewhere is quite different for $d > 1$ dimensions. For AdaBoost, areas surrounding points in the training data for which the observed class differs from that of the Bayes rule are classified according to the Bayes rule more often than they would be using the nearest neighbor rule.

We illustrate this again using $d = 2$ dimensions for visualization purposes. We use a Bayes error rate of $q = 0.1$ and take $n = 400$ points spread out evenly according to a Latin hypercube design. The plot on the left in Figure 19 shows the resulting classification rule of AdaBoost with 8-node trees at 1000 iterations for a single repetition using the new simulation model. The plot on the right shows the nearest neighbor rule. Both plots use the same color scheme as Figure 9. For the nearest neighbor rule, 21% of the points in the hold out sample disagree with the Bayes rule. This number is only 6% for AdaBoost, despite the fact that both classifiers classify every point in the training data according to the observed class label.

The difference between AdaBoost and the nearest neighbor rule is also well illustrated by other experiments in Section 4. For instance, in Section 4.1 the misclassification error for the nearest neighbor classifier was 0.499 but only 0.178 for AdaBoost with the 8-node trees.

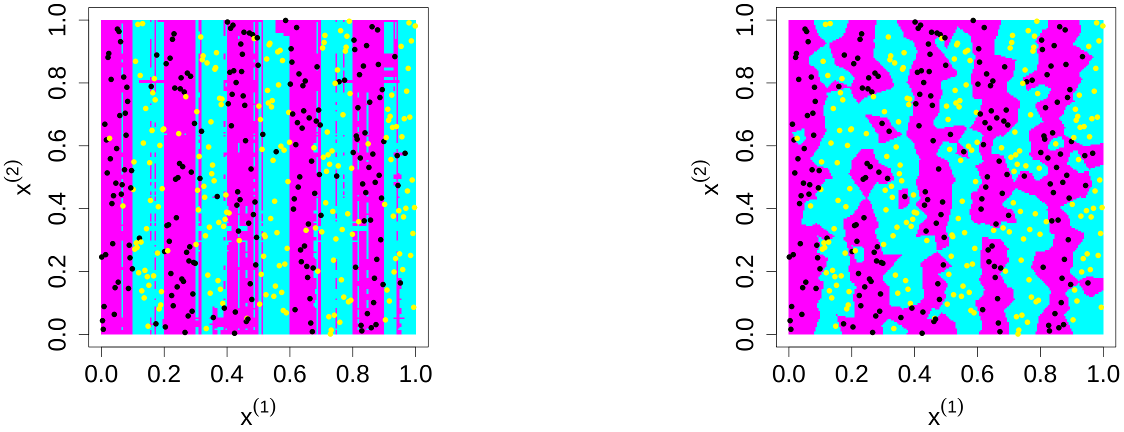

Figure 19: Comparison of AdaBoost (Left) and Nearest Neighbor (Right) with 10% Bayes Error

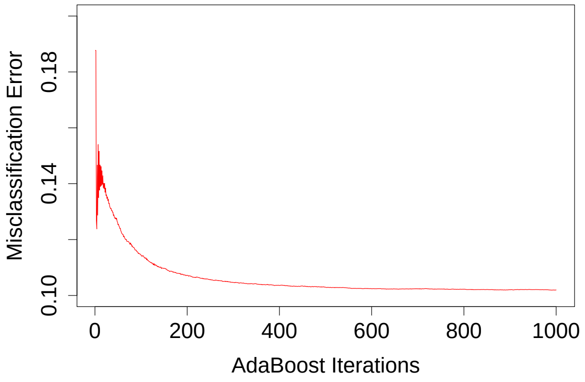

Figure 20: Performance of AdaBoost for a Simulation with a Bayes Error of 0.1

4.10 Is Boosting Consistent?

In Section 3.10 we illustrated that with a large sample size n, the misclassification error for AdaBoost can come quite close to the Bayes error rate, despite the fact that AdaBoost fits the training data perfectly. We illustrate this again in this section. As in Section 3.10, we use 2^8-node trees and a Bayes error rate of $q = 0.1$ but now take $n = 10,000$ and use the new simulation model.

Figure 20 shows the misclassification error averaged over 100 repetitions using hold out samples of size 1000. The mean misclassification error after 1000 iterations was 0.102 with a standard error of $0.009/\sqrt{100}$=0.0009. As we saw in Section 3.10, this is extremely close to the Bayes error rate and much less than the nearest neighbor bound of $2q(1-q) = 0.18$. We encourage readers to rerun the simulation with larger n to make the misclassification error even closer to the Bayes error.

5. Additional Experiments Which Contradict the Statistical View of Boosting

As mentioned at the beginning of Section 4, we encourage the reader to try simulation models other than those considered in this paper by using the R code provided on the web page http://www.davemease.com/contraryevidence. The simulation model can be specified by changing only three lines of this code in most cases. We have only considered two simulation models in this paper due to space constraints.

One criticism of the two simulation models considered in this paper is that both have a discontinuous (piecewise constant) conditional class probability function $p(x) \equiv \mathrm{P}(Y=1|x)$. An argument can be made that both AdaBoost and LogitBoost can not provide a good fit to these models because of the discontinuities. To investigate this, we examined additional experiments from the simulation model specified by

$$p(x) = 1/(1+e^{k(\Sigma_{j=1}^{J} x^{(j)} - J/2)})$$

where J is the number of effective dimensions as in Section 3 and k is a constant which determines the Bayes error rate. We note that this model has the same Bayes decision boundary as the model in Section 3 but now has a smooth conditional class probability function without any discontinuities. The results for this model are not included in the paper but are qualitatively extremely similar to those in Section 3. We encourage the reader to investigate this further.

6. Concluding Remarks and Practical Suggestions

By way of the simulations in Sections 3 and 4 we have seen that there are many problems with the statistical view of boosting and practical suggestions arising from that view. We do not endeavor to explain in this paper why these inconsistencies exist, nor do we offer a more complete view of boosting. Simply put, the goal of this paper has been to call into question this view of boosting that has come to dominate in the statistics community. The hope is that by doing so we have opened the door for future research toward a more thorough understanding of this powerful classification technique.

The statistical view of boosting focuses only on one aspect of the algorithm - the optimization. A more comprehensive view of boosting should also consider the stagewise nature of the algorithm as well as the empirical variance reduction that can be observed on hold out samples as with the experiments in this paper. Much insight on such ideas can be gained from reading work by the late Leo Breiman (e.g., Breiman, 2000, 2001) who subsequently abandoned interest in boosting and went on to work on his own classification technique known as Random Forests. The Random Forests algorithm achieves variance reduction directly through averaging as opposed to AdaBoost for which the variance reduction seems to happen accidently.

While we do not offer much in the way of an explanation for the behavior of AdaBoost in this paper, we will conclude with some practical advice in light of the evidence presented. First of all, AdaBoost remains one of, if not *the*, most successful boosting algorithms. One should not assume that newer, regularized and modified versions of boosting are necessarily better. We encourage readers to try standard AdaBoost along with these newer algorithms. If AdaBoost is not available as an option in your preferred software package, it is only a few lines of code to write yourself. Secondly, if classification is your goal, the best way to judge the effectiveness of boosting is by monitoring the misclassification error on hold out (or cross-validation) samples. We have seen that other loss functions are not necessarily indicative of the performance of boosting's classification rule. Finally, much of the evidence we have presented is indeed counter-intuitive. For this reason, a practitioner needs to keep an open mind when experimenting with AdaBoost. For example, if stumps are causing overfitting, be willing to try larger trees. Intuition may suggest the larger trees will overfit even more, but we have seen that is not necessarily true.

Acknowledgments

D. Mease's research was supported by an NSF-DMS post-doctoral fellowship. The authors are grateful to Andreas Buja and Abba Krieger for their help and guidance.

References

L. Breiman. Discussion of additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:374–377, 2000.

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

L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

P. Buhlmann and B. Yu. Boosting with the L_2 loss: Regression and classification. *Journal of the American Statistical Association*, 98:324–339, 2003.

A. Buja. Discussion of additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:387–391, 2000.

A. Buja, W. Stuetzle, and Y. Shen. Loss functions for binary class probability estimation and classification: Structure and applications. 2006.

M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *Computational Learing Theory*, pages 158–169, 2000.

A. Cutler and G. Zhao. Pert: Perfect random tree ensembles. *Computing Science and Statistics*, 33: 490–497, 2001.

M. Dettling and P. Buhlmann. Boosting for tumor classification with gene expression data. *Bioinformatics*, 19:1061–1069, 2003.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

Y. Freund and R. E. Schapire. Discussion of additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:391–393, 2000.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:337–374, 2000a.

J. Friedman, T. Hastie, and R. Tibshirani. Rejoiner for additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:400–407, 2000b.

A. J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 692–699, 1998.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

W. Jiang. Does boosting overfit: Views from an exact solution. *Technical Report 00-03, Department of Statistics, Northwestern University*, 2000.

W. Jiang. Is regularization unnecessary for boosting? In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2001.

W. Jiang. On weak base hypotheses and their implications for boosting regression and classification. *Annals of Statistics*, 30:51–73, 2002.

A. Krieger, C. Long, and A. J. Wyner. Boosting noisy data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 274–281, 2001.

G. Lugosi and N. Vayatis. On the bayes-risk consistency of regularized boosting methods. *Annals of Statistics*, 32:30–55, 2004.

D. Mease, A. Wyner, and A. Buja. Boosted classification trees and class probability/quantile estimation. *Journal of Machine Learning Research*, 8:409–439, 2007.

G. Ridgeway. Discussion of additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:393–400, 2000.

G. Ridgeway. Generalized boosted models: A guide to the gbm package. 2005.

R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26:1651–1686, 1998.

T. Zhang and B. Yu. Boosting with early stopping: Convergence and consistency. *Annals of Statistics*, 33:1538–1579, 2005.