The Annals of Statistics
2000, Vol. 28, No. 2, 337–407

SPECIAL INVITED PAPER

ADDITIVE LOGISTIC REGRESSION: A STATISTICAL VIEW OF BOOSTING

BY JEROME FRIEDMAN,[1] TREVOR HASTIE[2, 3] AND ROBERT TIBSHIRANI[2, 4]

Stanford University

Boosting is one of the most important recent developments in classification methodology. Boosting works by sequentially applying a classification algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers thus produced. For many classification algorithms, this simple strategy results in dramatic improvements in performance. We show that this seemingly mysterious phenomenon can be understood in terms of well-known statistical principles, namely additive modeling and maximum likelihood. For the two-class problem, boosting can be viewed as an approximation to additive modeling on the logistic scale using maximum Bernoulli likelihood as a criterion. We develop more direct approximations and show that they exhibit nearly identical results to boosting. Direct multiclass generalizations based on multinomial likelihood are derived that exhibit performance comparable to other recently proposed multiclass generalizations of boosting in most situations, and far superior in some. We suggest a minor modification to boosting that can reduce computation, often by factors of 10 to 50. Finally, we apply these insights to produce an alternative formulation of boosting decision trees. This approach, based on best-first truncated tree induction, often leads to better performance, and can provide interpretable descriptions of the aggregate decision rule. It is also much faster computationally, making it more suitable to large-scale data mining applications.

1. Introduction. The starting point for this paper is an interesting procedure called "boosting," which is a way of combining the performance of many "weak" classifiers to produce a powerful "committee." Boosting was proposed in the computational learning theory literature [Schapire (1990), Freund (1995), Freund and Schapire (1997)] and has since received much attention.

While boosting has evolved somewhat over the years, we describe the most commonly used version of the *AdaBoost* procedure [Freund and Schapire

Received August 1998; revised December 1999.

[1]Also at Stanford Linear Accelerator Center, Stanford, CA 94305. Supported in part by Dept. of Energy Contract DE-AC03-76 SF 00515 and NSF Grant DMS-97-64431.

[2]Also at Division of BioStatistics, Dept. of Health, Research and Policy, Stanford University, Stanford, CA 94305.

[3]Supported in part by NSF Grants DMS-95-04495, DMS-98-03645 and NIH Grant ROI-CA-72028-01.

[4]Supported in part by Natural Sciences and Engineering Research Council of Canada.

AMS 1991 *subject classifications*. 62G05, 62G07, 68T10, 68T05.

Key words and phrases. classification, tree, nonparametric estimation, stagewise fitting, machine learning.

(1996b)], which we call *Discrete AdaBoost*. This is essentially the same as AdaBoost.M1 for binary data in Freund and Schapire. Here is a concise description of AdaBoost in the two-class classification setting. We have training data $(x_1, y_1), \ldots, (x_N, y_N)$ with x_i a vector valued feature and $y_i = -1$ or 1. We define $F(x) = \sum_1^M c_m f_m(x)$ where each $f_m(x)$ is a classifier producing values *plus or minus* 1 and c_m are constants; the corresponding prediction is $\text{sign}(F(x))$. The AdaBoost procedure trains the classifiers $f_m(x)$ on weighted versions of the training sample, giving higher weight to cases that are currently misclassified. This is done for a sequence of weighted samples, and then the final classifier is defined to be a linear combination of the classifiers from each stage. A detailed description of Discrete AdaBoost is given in the boxed display titled Algorithm 1.

Much has been written about the success of AdaBoost in producing accurate classifiers. Many authors have explored the use of a tree-based classifier for $f_m(x)$ and have demonstrated that it consistently produces significantly lower error rates than a single decision tree. In fact, Breiman (1996) (referring to a NIPS workshop) called AdaBoost with trees the "best off-the-shelf classifier in the world" [see also Breiman (1998b)]. Interestingly, in many examples the test error seems to consistently decrease and then level off as more classifiers are added, rather than ultimately increase. For some reason, it seems that AdaBoost is resistant to overfitting.

Figure 1 shows the performance of Discrete AdaBoost on a synthetic classification task, using an adaptation of CART™ [Breiman, Friedman, Olshen and Stone (1984)] as the base classifier. This adaptation grows fixed-size trees in a "best-first" manner (see Section 8). Included in the figure is the *bagged* tree [Breiman (1996)] which averages trees grown on bootstrap resampled versions of the training data. Bagging is purely a variance-reduction technique, and since trees tend to have high variance, bagging often produces good results.

Discrete AdaBoost [Freund and Schapire (1996b)]

1. Start with weights $w_i = 1/N$, $i = 1, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the classifier $f_m(x) \in \{-1, 1\}$ using weights w_i on the training data.
   (b) Compute $\text{err}_m = E_w[1_{(y \neq f_m(x))}]$, $c_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (c) Set $w_i \leftarrow w_i \exp[c_m 1_{(y_i \neq f_m(x_i))}]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.
3. Output the classifier $\text{sign}[\sum_{m=1}^M c_m f_m(x)]$.

ALGORITHM 1. *E_w represents expectation over the training data with weights $w = (w_1, w_2, \ldots, w_N)$, and $1_{(S)}$ is the indicator of the set S. At each iteration, AdaBoost increases the weights of the observations misclassified by $f_m(x)$ by a factor that depends on the weighted training error.*

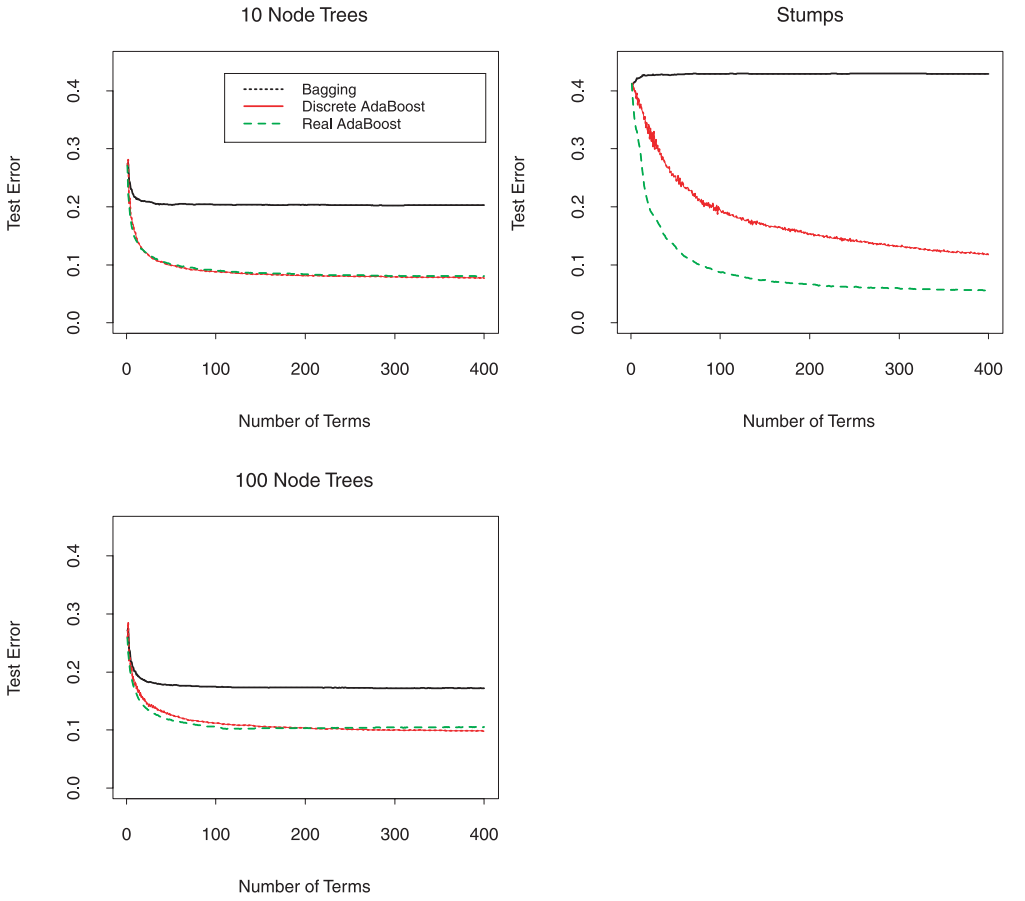

FIG. 1. *Test error for Bagging, Discrete AdaBoost and Real AdaBoost on a simulated two-class nested spheres problem* (*see Section* 6). *There are* 2000 *training data points in ten dimensions, and the Bayes error rate is zero. All trees are grown "best-first" without pruning. The leftmost iteration corresponds to a single tree.*

Early versions of AdaBoost used a resampling scheme to implement step 2 of Algorithm 1, by weighted sampling from the training data. This suggested a connection with bagging and that a major component of the success of boosting has to do with variance reduction.

However, boosting performs comparably well when:

1. A weighted tree-growing algorithm is used in step 2 rather than weighted resampling, where each training observation is assigned its weight w_i. This removes the randomization component essential in bagging.
2. "Stumps" are used for the weak learners. Stumps are single-split trees with only two terminal nodes. These typically have low variance but high bias. Bagging performs very poorly with stumps [Figure 1 (top right panel)].

These observations suggest that boosting is capable of both bias and variance reduction, and thus differs fundamentally from bagging.

The *base classifier* in Discrete AdaBoost produces a classification rule $f_m(x)$: $\mathscr{X} \mapsto \{-1, 1\}$, where $\mathscr{X}$ is the domain of the predictive features x. Freund and Schapire (1996b), Breiman (1998a) and Schapire and Singer (1998) have suggested various modifications to improve the boosting algorithms.

A generalization of Discrete AdaBoost appeared in Freund and Schapire (1996b), and was developed further in Schapire and Singer (1998), that uses real-valued "confidence-rated" predictions rather than the $\{-1, 1\}$ of Discrete AdaBoost. The weak learner for this generalized boosting produces a mapping $f_m(x)$: $\mathscr{X} \mapsto R$; the sign of $f_m(x)$ gives the classification, and $|f_m(x)|$ a measure of the "confidence" in the prediction. This real-valued contribution is combined with the previous contributions with a multiplier c_m as before, and a slightly different recipe for c_m is provided.

We present a generalized version of AdaBoost, which we call *Real AdaBoost* in Algorithm 2, in which the weak learner returns a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$. The contribution to the final classifier is half the logit-transform of this probability estimate. One form of Schapire and Singer's generalized AdaBoost coincides with Real AdaBoost, in the special case where the weak learner is a decision tree. Real AdaBoost tends to perform the best in our simulated examples in Figure 1, especially with stumps, although we see with 100 node trees Discrete AdaBoost overtakes Real AdaBoost after 200 iterations.

In this paper we analyze the AdaBoost procedures from a statistical perspective. The main result of our paper rederives AdaBoost as a method for fitting an additive model $\sum_m f_m(x)$ in a forward stagewise manner. This simple fact largely explains why it tends to outperform a single base learner. By fitting an additive model of different and potentially simple functions, it expands the class of functions that can be approximated.

Real AdaBoost

1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the classifier to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$, using weights w_i on the training data.
   (b) Set $f_m(x) \leftarrow \frac{1}{2} \log p_m(x)/(1 - p_m(x)) \in R$.
   (c) Set $w_i \leftarrow w_i \exp[-y_i f_m(x_i)]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.
3. Output the classifier $\text{sign}[\sum_{m=1}^{M} f_m(x)]$.

ALGORITHM 2. *The Real AdaBoost algorithm uses class probability estimates $p_m(x)$ to construct real-valued contributions $f_m(x)$.*

Given this fact, Discrete and Real AdaBoost appear unnecessarily complicated. A much simpler way to fit an additive model would be to minimize squared-error loss $E(y - \sum f_m(x))^2$ in a forward stagewise manner. At the mth stage we fix $f_1(x) \cdots f_{m-1}(x)$ and minimize squared error to obtain $f_m(x) = E(y - \sum_1^{m-1} f_j(x)|x)$. This is just "fitting of residuals" and is commonly used in linear regression and additive modeling [Hastie and Tibshirani (1990)].

However squared error loss is not a good choice for classification (see Figure 2 in Section 4.2) and hence "fitting of residuals" doesn't work very well in that case. We show that AdaBoost fits an additive model using a better loss function for classification. Specifically we show that AdaBoost fits an additive logistic regression model, using a criterion similar to, but not the same as, the binomial log-likelihood. [If $p_m(x)$ are the class probabilities, an additive logistic regression approximates $\log p_m(x)/(1 - p_m(x))$ by an additive function $\sum_m f_m(x)$.] We then go on to derive a new boosting procedure "LogitBoost" that directly optimizes the binomial log-likelihood.

The original boosting techniques [Schapire (1990), Freund (1995)] provably improved or "boosted" the performance of a single classifier by producing a "majority vote" of similar classifiers. These algorithms then evolved into more adaptive and practical versions such as AdaBoost, whose success was still explained in terms of boosting individual classifiers by a "weighted majority vote" or "weighted committee." We believe that this view, along with the appealing name "boosting" inherited by AdaBoost, may have led to some of the mystery about how and why the method works. As mentioned above, we instead view boosting as a technique for fitting an additive model.

Section 2 gives a short history of the boosting idea. In Section 3 we briefly review additive modeling. Section 4 shows how boosting can be viewed as an additive model estimator and proposes some new boosting methods for the two-class case. The multiclass problem is studied in Section 5. Simulated and real data experiments are discussed in Sections 6 and 7. Our tree-growing implementation, using truncated best-first trees, is described in Section 8. Weight trimming to speed up computation is discussed in Section 9, and we briefly describe generalizations of boosting in Section 10. We end with a discussion in Section 11.

2. A brief history of boosting. Schapire (1990) developed the first simple boosting procedure in the PAC-learning framework [Valiant (1984), Kearns and Vazirani (1994)]. Schapire showed that a *weak learner* could always improve its performance by training two additional classifiers on filtered versions of the input data stream. A weak learner is an algorithm for producing a two-class classifier with performance guaranteed (with high probability) to be significantly better than a coinflip. After learning an initial classifier h_1 on the first N training points:

1. h_2 is learned on a new sample of N points, half of which are misclassified by h_1.
2. h_3 is learned on N points for which h_1 and h_2 disagree.
3. The boosted classifier is $h_B = \mathit{Majority\ Vote}(h_1, h_2, h_3)$.

Schapire's "Strength of Weak Learnability" theorem proves that h_B has improved performance over h_1.

Freund (1995) proposed a "boost by majority" variation which combined many weak learners simultaneously and improved the performance of the simple boosting algorithm of Schapire. The theory supporting both of these algorithms requires the weak learner to produce a classifier with a fixed error rate. This led to the more adaptive and realistic AdaBoost [Freund and Schapire (1996b)] and its offspring, where this assumption was dropped.

Freund and Schapire (1996b) and Schapire and Singer (1998) provide some theory to support their algorithms, in the form of upper bounds on generalization error. This theory has evolved in the computational learning community, initially based on the concepts of PAC learning. Other theories attempting to explain boosting come from game theory [Freund and Schapire (1996a), Breiman (1997)] and VC theory [Schapire, Freund, Bartlett and Lee (1998)]. The bounds and the theory associated with the AdaBoost algorithms are interesting, but tend to be too loose to be of practical importance. In practice, boosting achieves results far more impressive than the bounds would imply.

3. Additive models. We show in the next section that AdaBoost fits an *additive* model $F(x) = \sum_{m=1}^{M} c_m f_m(x)$. We believe that viewing current boosting procedures as stagewise algorithms for fitting additive models goes a long way toward understanding their performance. Additive models have a long history in statistics, and so we first give some examples here.

3.1. *Additive regression models.* We initially focus on the regression problem, where the response y is quantitative, x and y have some joint distribution, and we are interested in modeling the mean $E(y|x) = F(x)$. The additive model has the form

$$F(x) = \sum_{j=1}^{p} f_j(x_j). \tag{1}$$

There is a separate function $f_j(x_j)$ for each of the p input variables x_j. More generally, each component f_j is a function of a small, prespecified subset of the input variables. The *backfitting algorithm* [Friedman and Stuetzle (1981), Buja, Hastie and Tibshirani (1989)] is a convenient modular "Gauss–Seidel" algorithm for fitting additive models. A backfitting update is

$$f_j(x_j) \leftarrow E\left[y - \sum_{k \neq j} f_k(x_k) \middle| x_j \right] \quad \text{for } j = 1, 2, \ldots, p, 1, \ldots . \tag{2}$$

Any method or algorithm for estimating a function of x_j can be used to obtain an estimate of the conditional expectation in (2). In particular, this can include nonparametric *smoothing* algorithms, such as local regression or smoothing splines. In the right-hand side, all the latest versions of the functions f_k are used in forming the partial residuals. The backfitting cycles are repeated until convergence. Under fairly general conditions, backfitting can

be shown to converge to the minimizer of $E(y - F(x))^2$ [Buja, Hastie and Tibshirani (1989)].

3.2. *Extended additive models.* More generally, one can consider additive models whose elements $\{f_m(x)\}_1^M$ are functions of potentially all of the input features x. Usually in this context the $f_m(x)$ are taken to be simple functions characterized by a set of parameters γ and a multiplier β_m,

$$f_m(x) = \beta_m b(x; \gamma_m). \tag{3}$$

The additive model then becomes

$$F_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m). \tag{4}$$

For example, in single hidden layer neural networks $b(x; \gamma) = \sigma(\gamma^t x)$ where $\sigma(\cdot)$ is a sigmoid function and γ parameterizes a linear combination of the input features. In signal processing, wavelets are a popular choice with γ parameterizing the location and scale shifts of a "mother" wavelet $b(x)$. In these applications $\{b(x; \gamma_m)\}_1^M$ are generally called "basis functions" since they span a function subspace.

If least-squares is used as a fitting criterion, one can solve for an optimal set of parameters through a generalized backfitting algorithm with updates,

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta, \gamma} E\left[y - \sum_{k \neq m} \beta_k b(x; \gamma_k) - \beta b(x; \gamma) \right]^2 \tag{5}$$

for $m = 1, 2, \ldots, M$ in cycles until convergence. Alternatively, one can use a "greedy" forward stepwise approach,

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta, \gamma} E\big[y - F_{m-1}(x) - \beta b(x; \gamma) \big]^2 \tag{6}$$

for $m = 1, 2, \ldots, M$, where $\{\beta_k, \gamma_k\}_1^{m-1}$ are fixed at their corresponding solution values at earlier iterations. This is the approach used by Mallat and Zhang (1993) in "matching pursuit," where the $b(x; \gamma)$ are selected from an over-complete dictionary of wavelet bases. In the language of boosting, $f(x) = \beta b(x; \gamma)$ would be called a "weak learner" and $F_M(x)$ (4) the "committee." If decision trees were used as the weak learner, the parameters γ would represent the splitting variables, split points, the constants in each terminal node and number of terminal nodes of each tree.

Note that the backfitting procedure (5) or its greedy cousin (6) only require an algorithm for fitting a *single* weak learner (3) to data. This base algorithm is simply applied repeatedly to modified versions of the original data

$$y_m \leftarrow y - \sum_{k \neq m} f_k(x).$$

In the forward stepwise procedure (6), the modified output y_m at the mth iteration depends only on its value y_{m-1} and the solution $f_{m-1}(x)$ at the previous iteration,

$$y_m = y_{m-1} - f_{m-1}(x). \tag{7}$$

At each step m, the previous output values y_{m-1} are modified (7) so that the previous model $f_{m-1}(x)$ has no explanatory power on the new outputs y_m. One can therefore view this as a procedure for boosting a weak learner $f(x) = \beta b(x; \gamma)$ to form a powerful committee $F_M(x)$ (4).

3.3. *Classification problems.* For the classification problem, we learn from Bayes theorem that all we need is $P(y = j|x)$, the posterior or conditional class probabilities. One could transfer all the above regression machinery across to the classification domain by simply noting that $E(1_{[y=j]}|x) = P(y = j|x)$, where $1_{[y=j]}$ is the 0/1 indicator variable representing class j. While this works fairly well in general, several problems have been noted [Hastie, Tibshirani and Buja (1994)] for constrained regression methods. The estimates are typically not confined to $[0, 1]$, and severe masking problems can occur when there are more than two classes. A notable exception is when trees are used as the regression method, and in fact this is the approach used by Breiman, Friedman, Olshen and Stone (1984).

Logistic regression is a popular approach used in statistics for overcoming these problems. For a two-class problem, an additive logistic model has the form

$$\log \frac{P(y = 1|x)}{P(y = -1|x)} = \sum_{m=1}^{M} f_m(x). \tag{8}$$

The monotone *logit* transformation on the left guarantees that for any values of $F(x) = \sum_{m=1}^{M} f_m(x) \in R$, the probability estimates lie in $[0, 1]$; inverting we get

$$p(x) = P(y = 1|x) = \frac{e^{F(x)}}{1 + e^{F(x)}}. \tag{9}$$

Here we have given a general additive form for $F(x)$; special cases exist that are well known in statistics. In particular, linear logistic regression [McCullagh and Nelder (1989), e.g.] and additive logistic regression [Hastie and Tibshirani (1990)] are popular. These models are usually fit by maximizing the binomial log-likelihood and enjoy all the associated asymptotic optimality features of maximum likelihood estimation.

A generalized version of backfitting (2), called "Local Scoring" in Hastie and Tibshirani (1990), can be used to fit the additive logistic model by maximum likelihood. Starting with guesses $f_1(x_1) \cdots f_p(x_p)$, $F(x) = \sum f_k(x_k)$ and $p(x)$ defined in (9), we form the working response:

$$z = F(x) + \frac{1_{[y=1]} - p(x)}{p(x)(1 - p(x))}. \tag{10}$$

We then apply backfitting to the response z with observation weights $p(x)(1-p(x))$ to obtain new $f_k(x_k)$. This process is repeated until convergence. The forward stagewise version (6) of this procedure bears a close similarity to the LogitBoost algorithm described later in the paper.

4. AdaBoost: an additive logistic regression model. In this section we show that the AdaBoost algorithms (Discrete and Real) can be interpreted as stagewise estimation procedures for fitting an additive logistic regression model. They optimize an exponential criterion which to second order is equivalent to the binomial log-likelihood criterion. We then propose a more standard likelihood-based boosting procedure.

4.1. *An exponential criterion.* Consider minimizing the criterion

$$J(F) = E(e^{-yF(x)}) \tag{11}$$

for estimation of $F(x)$. Here E represents expectation; depending on the context, this may be a population expectation (with respect to a probability distribution) or else a sample average. E_w indicates a weighted expectation. Lemma 1 shows that the function $F(x)$ that minimizes $J(F)$ is the symmetric logistic transform of $P(y=1|x)$.

LEMMA 1. *$E(e^{-yF(x)})$ is minimized at*

$$F(x) = \frac{1}{2}\log\frac{P(y=1|x)}{P(y=-1|x)}. \tag{12}$$

Hence

$$P(y=1|x) = \frac{e^{F(x)}}{e^{-F(x)}+e^{F(x)}}, \tag{13}$$

$$P(y=-1|x) = \frac{e^{-F(x)}}{e^{-F(x)}+e^{F(x)}}. \tag{14}$$

PROOF. While E entails expectation over the joint distribution of y and x, it is sufficient to minimize the criterion conditional on x:

$$E\big(e^{-yF(x)}|x\big) = P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)},$$

$$\frac{\partial E\big(e^{-yF(x)}|x\big)}{\partial F(x)} = -P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)}.$$

The result follows by setting the derivative to zero. □

This exponential criterion appeared in Schapire and Singer (1998), motivated as an upper bound on misclassification error. Breiman (1997) also used this criterion in his results on AdaBoost and prediction games. The usual

logistic transform does not have the factor $\frac{1}{2}$ as in (12); by multiplying the numerator and denominator in (13) by $e^{F(x)}$, we get the usual logistic model

$$p(x) = \frac{e^{2F(x)}}{1 + e^{2F(x)}}. \tag{15}$$

Hence the two models are equivalent up to a factor 2.

COROLLARY 1. *If E is replaced by averages over regions of x where $F(x)$ is constant (as in the terminal node of a decision tree), the same result applies to the sample proportions of $y = 1$ and $y = -1$.*

Results 1 and 2 show that both Discrete and Real AdaBoost, as well as the Generalized AdaBoost of Freund and Schapire (1996b), can be motivated as iterative algorithms for optimizing the (population based) exponential criterion. The results share the same format.

1. Given an imperfect $F(x)$, an update $F(x) + f(x)$ is proposed based on the population version of the criterion.
2. The update, which involves population conditional expectations, is imperfectly approximated for finite data sets by some restricted class of estimators, such as averages in terminal nodes of trees.

Hastie and Tibshirani (1990) use a similar derivation of the local scoring algorithm used in fitting generalized additive models. Many terms are typically required in practice, since at each stage the approximation to conditional expectation is rather crude. Because of Lemma 1, the resulting algorithms can be interpreted as a stagewise estimation procedure for fitting an additive logistic regression model. The derivations are sufficiently different to warrant separate treatment.

RESULT 1. *The Discrete AdaBoost algorithm (population version) builds an additive logistic regression model via Newton-like updates for minimizing $E(e^{-yF(x)})$.*

PROOF. Let $J(F) = E[e^{-yF(x)}]$. Suppose we have a current estimate $F(x)$ and seek an improved estimate $F(x) + cf(x)$. For fixed c (and x), we expand $J(F(x) + cf(x))$ to second order about $f(x) = 0$,

$$\begin{aligned} J(F + cf) &= E[e^{-y(F(x)+cf(x))}] \\ &\approx E[e^{-yF(x)}(1 - ycf(x) + c^2y^2f(x)^2/2)] \\ &= E[e^{-yF(x)}(1 - ycf(x) + c^2/2)], \end{aligned}$$

since $y^2 = 1$ and $f(x)^2 = 1$. Minimizing pointwise with respect to $f(x) \in \{-1, 1\}$, we write

$$f(x) = \arg\min_f E_w(1 - ycf(x) + c^2/2|x). \tag{16}$$

Here the notation $E_w(\cdot|x)$ refers to a *weighted conditional expectation*, where $w = w(x, y) = e^{-yF(x)}$, and

$$E_w[g(x, y)|x] \stackrel{\text{def}}{=} \frac{E[w(x, y)g(x, y)|x]}{E[w(x, y)|x]}.$$

For $c > 0$, minimizing (16) is equivalent to maximizing

$$E_w[yf(x)]. \tag{17}$$

The solution is

$$f(x) = \begin{cases} 1, & \text{if } E_w(y|x) = P_w(y = 1|x) - P_w(y = -1|x) > 0, \\ -1, & \text{otherwise.} \end{cases} \tag{18}$$

Note that

$$-E_w[yf(x)] = E_w[y - f(x)]^2/2 - 1 \tag{19}$$

[again using $f(x)^2 = y^2 = 1$]. Thus minimizing a quadratic approximation to the criterion leads to a weighted least-squares choice of $f(x) \in \{-1, 1\}$, and this constitutes the Newton-like step.

Given $f(x) \in \{-1, 1\}$, we can directly minimize $J(F + cf)$ to determine c:

$$\begin{aligned} c &= \arg\min_c E_w e^{-cyf(x)} \\ &= \frac{1}{2}\log\frac{1 - \text{err}}{\text{err}}, \end{aligned} \tag{20}$$

where $\text{err} = E_w[1_{[y \neq f(x)]}]$. Note that c can be negative if the weak learner does *worse* than 50%, in which case it automatically reverses the polarity. Combining these steps we get the update for $F(x)$,

$$F(x) \leftarrow F(x) + \frac{1}{2}\log\frac{1 - \text{err}}{\text{err}} f(x).$$

In the next iteration the new contribution $cf(x)$ to $F(x)$ augments the weights

$$w(x, y) \leftarrow w(x, y)e^{-cf(x)y}.$$

Since $-yf(x) = 2 \times 1_{[y \neq f(x)]} - 1$, we see that the update is equivalent to

$$w(x, y) \leftarrow w(x, y)\exp\left(\log\left(\frac{1 - \text{err}}{\text{err}}\right)1_{[y \neq f(x)]}\right).$$

Thus the function and weight updates are of an identical form to those used in Discrete AdaBoost.

This population version of AdaBoost translates naturally to a data version using trees. The weighted conditional expectation in (18) is approximated by the terminal-node weighted averages in a tree. In particular, the weighted least-squares criterion is used to grow the tree-based classifier $f(x)$, and given $f(x)$, the constant c is based on the weighted training error.

Note that after each Newton step, the weights change, and hence the tree configuration will change as well. This adds an adaptive twist to the data version of a Newton-like algorithm.

Parts of this derivation for AdaBoost can be found in Breiman (1997) and Schapire and Singer (1998), but without making the connection to additive logistic regression models.

COROLLARY 2. *After each update to the weights, the* weighted *misclassification error of the most recent weak learner is* 50%.

PROOF. This follows by noting that the c that minimizes $J(F+cf)$ satisfies

$$\frac{\partial J(F+cf)}{\partial c} = -E[e^{-y(F(x)+cf(x))}yf(x)] = 0. \tag{21}$$

The result follows since $yf(x)$ is 1 for a correct and -1 for an incorrect classification. □

Schapire and Singer (1998) give the interpretation that the weights are updated to make the new weighted problem maximally difficult for the next weak learner.

The Discrete AdaBoost algorithm expects the tree or other "weak learner" to deliver a classifier $f(x) \in \{-1, 1\}$. Result 1 requires minor modifications to accommodate $f(x) \in R$, as in the generalized AdaBoost algorithms [Freund and Schapire (1996b), Schapire and Singer (1998)]; the estimate for c_m differs. Fixing f, we see that the minimizer of (20) must satisfy

$$E_w[yf(x)e^{-cyf(x)}] = 0. \tag{22}$$

If f is not discrete, this equation has no closed-form solution for c, and requires an iterative solution such as Newton–Raphson.

We now derive the Real AdaBoost algorithm, which uses weighted probability estimates to update the additive logistic model, rather than the classifications themselves. Again we derive the population updates and then apply it to data by approximating conditional expectations by terminal-node averages in trees.

RESULT 2. *The Real AdaBoost algorithm fits an additive logistic regression model by stagewise and approximate optimization of* $J(F) = E[e^{-yF(x)}]$.

PROOF. Suppose we have a current estimate $F(x)$ and seek an improved estimate $F(x) + f(x)$ by minimizing $J(F(x) + f(x))$ at each x.

$$\begin{aligned} J(F(x)+f(x)) &= E(e^{-yF(x)}e^{-yf(x)}|x) \\ &= e^{-f(x)}E[e^{-yF(x)}1_{[y=1]}|x] + e^{f(x)}E[e^{-yF(x)}1_{[y=-1]}|x]. \end{aligned}$$

Dividing through by $E[e^{-yF(x)}|x]$ and setting the derivative w.r.t. $f(x)$ to zero we get

$$f(x) = \frac{1}{2}\log\frac{E_w[1_{[y=1]}|x]}{E_w[1_{[y=-1]}|x]} \tag{23}$$

$$= \frac{1}{2} \log \frac{P_w(y=1|x)}{P_w(y=-1|x)}, \tag{24}$$

where $w(x, y) = \exp(-yF(x))$. The weights get updated by

$$w(x, y) \leftarrow w(x, y)e^{-yf(x)}.$$

The algorithm as presented would stop after one iteration. In practice we use crude approximations to conditional expectation, such as decision trees or other constrained models, and hence many steps are required.

COROLLARY 3. *At the optimal $F(x)$, the weighted conditional mean of y is* 0.

PROOF. If $F(x)$ is optimal, we have

$$\frac{\partial J(F(x))}{F(x)} = -Ee^{-yF(x)}y = 0. \tag{25}$$

□

We can think of the weights as providing an alternative to residuals for the binary classification problem. At the optimal function F, there is no further information about F in the weighted conditional distribution of y. If there is, we use it to update F.

An iteration M in either the Discrete or Real AdaBoost algorithms, we have composed an additive function of the form

$$F(x) = \sum_{m=1}^{M} f_m(x), \tag{26}$$

where each of the components are found in a greedy forward stagewise fashion, fixing the earlier components. Our term "stagewise" refers to a similar approach in statistics:

1. Variables are included sequentially in a stepwise regression.
2. The coefficients of variables already included receive no further adjustment.

4.2. *Why $Ee^{-yF(x)}$?* So far the only justification for this exponential criterion is that it has a sensible population minimizer, and the algorithm described above performs well on real data. In addition:

1. Schapire and Singer (1998) motivate $e^{-yF(x)}$ as a differentiable upper bound to misclassification error $1_{[yF<0]}$ (see Figure 2).
2. The AdaBoost algorithm that it generates is extremely modular, requiring at each iteration the retraining of a classifier on a weighted training database.

Let $y^* = (y + 1)/2$, taking values 0, 1, and parametrize the binomial probabilities by

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}.$$

The binomial log-likelihood is

$$\begin{aligned} l(y^*, p(x)) &= y^* \log(p(x)) + (1 - y^*)\log(1 - p(x)) \\ &= -\log(1 + e^{-2yF(x)}). \end{aligned} \tag{27}$$

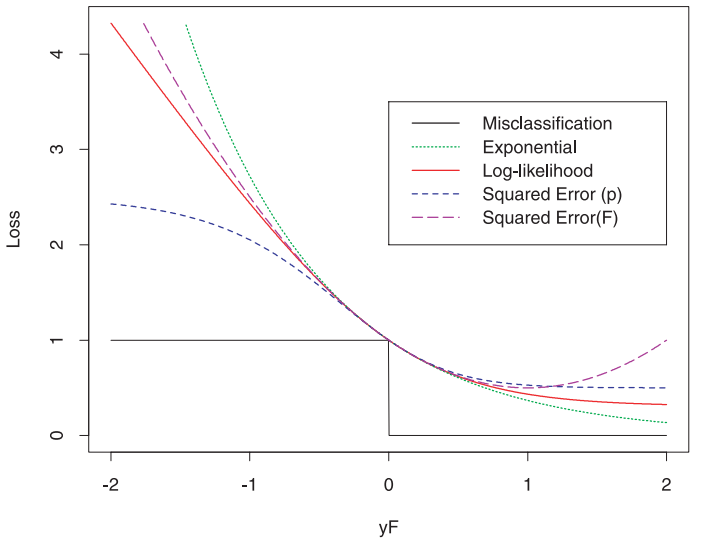

FIG. 2. *A variety of loss functions for estimating a function $F(x)$ for classification. The horizontal axis is yF, which is negative for errors and positive for correct classifications. All the loss functions are monotone in yF, and are centered and scaled to match e^{-yF} at $F = 0$. The curve labeled "Log-likelihood" is the binomial log-likelihood or cross-entropy $y^* \log p + (1 - y^*)\log(1 - p)$. The curve labeled "Squared Error(p)" is $(y^* - p)^2$. The curve labeled "Squared Error(F)" is $(y - F)^2$ and increases once yF exceeds* 1, *thereby increasingly penalizing classifications that are "too correct."*

Hence we see that:

3. The population minimizers of $-El(y^*, p(x))$ and $Ee^{-yF(x)}$ coincide. This is easily seen because the expected log-likelihood is maximized at the true probabilities $p(x) = P(y^* = 1|x)$, which define the logit $F(x)$. By Lemma 1 we see that this is exactly the minimizer of $Ee^{-yF(x)}$. In fact, the exponential criterion and the (negative) log-likelihood are equivalent to second order in a Taylor series around $F = 0$,

$$-l(y^*, p) \approx \exp(-yF) + \log(2) - 1. \tag{28}$$

Graphs of $\exp(-yF)$ and $\log(1 + e^{-2yF(x)})$ are shown in Figure 2, as a function of yF—positive values of yF imply correct classification. Note that $-\exp(-yF)$ itself is not a proper log-likelihood, as it does not equal the log of any probability mass function on plus or minus 1.

4. There is another way to view the criterion $J(F)$. It is easy to show that

$$e^{-yF(x)} = \frac{|y^* - p(x)|}{\sqrt{p(x)(1 - p(x))}}, \tag{29}$$

with $F(x) = \frac{1}{2}\log(p(x)/(1 - p(x)))$. The right-hand side is known as the χ statistic in the statistical literature. χ^2 is a quadratic approximation to the log-likelihood, and so χ can be considered a "gentler" alternative.

One feature of both the exponential and log-likelihood criteria is that they are monotone and smooth. Even if the training error is zero, the criteria will drive the estimates towards purer solutions (in terms of probability estimates).

Why not estimate the f_m by minimizing the squared error $E(y - F(x))^2$? If $F_{m-1}(x) = \sum_1^{m-1} f_j(x)$ is the current prediction, this leads to a forward stagewise procedure that does an unweighted fit to the response $y - F_{m-1}(x)$ at step m as in (6). Empirically we have found that this approach works quite well, but is dominated by those that use monotone loss criteria. We believe that the nonmonotonicity of squared error loss (Figure 2) is the reason. Correct classifications, but with $yF(x) > 1$, incur increasing loss for increasing values of $|F(x)|$. This makes squared-error loss an especially poor approximation to misclassification error rate. Classifications that are "too correct" are penalized as much as misclassification errors.

4.3. *Direct optimization of the binomial log-likelihood.* In this section we explore algorithms for fitting additive logistic regression models by stagewise optimization of the Bernoulli log-likelihood. Here we focus again on the two-class case and will use a 0/1 response y^* to represent the outcome. We represent the probability of $y^* = 1$ by $p(x)$, where

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}. \tag{30}$$

Algorithm 3 gives the details.

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \ldots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Compute the working response and weights
   $$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$
   $$w_i = p(x_i)(1 - p(x_i)).$$
   (b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i.
   (c) Update $F(x) \leftarrow F(x) + \frac{1}{2} f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.
3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

ALGORITHM 3. *An adaptive Newton algorithm for fitting an additive logistic regression model.*

RESULT 3. *The LogitBoost algorithm (two classes, population version) uses Newton steps for fitting an additive symmetric logistic model by maximum likelihood.*

DERIVATION. Consider the update $F(x) + f(x)$ and the expected log-likelihood

$$El(F + f) = E\big[2y^*(F(x) + f(x)) - \log[1 + e^{2(F(x)+f(x))}]. \tag{31}$$

Conditioning on x, we compute the first and second derivative at $f(x) = 0$,

$$\begin{aligned} s(x) &= \left.\frac{\partial El(F(x) + f(x))}{\partial f(x)}\right|_{f(x)=0} \\ &= 2E(y^* - p(x)|x), \end{aligned} \tag{32}$$

$$\begin{aligned} H(x) &= \left.\frac{\partial^2 El(F(x) + f(x))}{\partial f(x)^2}\right|_{f(x)=0} \\ &= -4E(p(x)(1 - p(x))|x), \end{aligned} \tag{33}$$

where $p(x)$ is defined in terms of $F(x)$. The Newton update is then

$$\begin{aligned} F(x) &\leftarrow F(x) - H(x)^{-1}s(x) \\ &= F(x) + \frac{1}{2}\frac{E(y^* - p(x)|x)}{E(p(x)(1 - p(x))|x)} \end{aligned} \tag{34}$$

$$= F(x) + \frac{1}{2}E_w\left(\frac{y^* - p(x)}{p(x)(1 - p(x))}\middle| x\right), \tag{35}$$

where $w(x) = p(x)(1 - p(x))$. Equivalently, the Newton update $f(x)$ solves the weighted least-squares approximation [about $F(x)$] to the log-likelihood

$$\min_{f(x)} E_{w(x)}\left(F(x) + \frac{1}{2}\frac{y^* - p(x)}{p(x)(1 - p(x))} - (F(x) + f(x))\right)^2. \tag{36}$$

The population algorithm described here translates immediately to an implementation on data when $E(\cdot|x)$ is replaced by a regression method, such as regression trees [Breiman, Friedman, Olshen and Stone (1984)]. While the role of the weights are somewhat artificial in the population case, they are not in any implementation; $w(x)$ is constant when conditioned on x, but the $w(x_i)$ in a terminal node of a tree, for example, depend on the current values $F(x_i)$, and will typically not be constant.

Sometimes the $w(x)$ get very small in regions of (x) perceived [by $F(x)$] to be *pure*—that is, when $p(x)$ is close to 0 or 1. This can cause numerical problems in the construction of z, and lead to the following crucial implementation protections:

1. If $y^* = 1$, then compute $z = ((y^* - p)/p(1 - p))$ as $1/p$. Since this number can get large if p is small, threshold this ratio at $z\ max$. The particular value chosen for $z\ max$ is not crucial; we have found empirically that

$z\ max \in [2, 4]$ works well. Likewise, if $y^* = 0$, compute $z = -1/(1-p)$ with a lower threshold of $-z\ max$.

2. Enforce a lower threshold on the weights: $w = \max(w, 2 \times machine\text{-}zero)$.

4.4. *Optimizing $Ee^{-yF(x)}$ by Newton stepping.* The population version of the Real AdaBoost procedure (Algorithm 2) optimizes $E\exp(-y(F(x)+f(x)))$ exactly with respect to f at each iteration. In Algorithm 4 we propose the "Gentle AdaBoost" procedure that instead takes adaptive Newton steps much like the LogitBoost algorithm just described.

RESULT 4. *The Gentle AdaBoost algorithm (population version) uses Newton steps for minimizing $Ee^{-yF(x)}$.*

DERIVATION.

$$\left.\frac{\partial J(F(x)+f(x))}{\partial f(x)}\right|_{f(x)=0} = -E(e^{-yF(x)}y|x),$$

$$\left.\frac{\partial^2 J(F(x)+f(x))}{\partial f(x)^2}\right|_{f(x)=0} = E(e^{-yF(x)}|x) \text{ since } y^2 = 1.$$

Hence the Newton update is

$$\begin{aligned} F(x) &\leftarrow F(x) + \frac{E(e^{-yF(x)}y|x)}{E(e^{-yF(x)}|x)} \\ &= F(x) + E_w(y|x), \end{aligned}$$

where $w(x, y) = e^{-yF(x)}$.

The main difference between this and the Real AdaBoost algorithm is how it uses its estimates of the weighted class probabilities to update the functions. Here the update is $f_m(x) = P_w(y = 1|x) - P_w(y = -1|x)$, rather than half the

---

Gentle AdaBoost

1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$, $F(x) = 0$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the regression function $f_m(x)$ by weighted least-squares of y_i to x_i with weights w_i.
   (b) Update $F(x) \leftarrow F(x) + f_m(x)$.
   (c) Update $w_i \leftarrow w_i \exp(-y_i f_m(x_i))$ and renormalize.
3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

ALGORITHM 4. *A modified version of the Real AdaBoost algorithm, using Newton stepping rather than exact optimization at each step.*

log-ratio as in (24): $f_m(x) = \frac{1}{2}\log(P_w(y = 1|x))/(P_w(y = -1|x))$. Log-ratios can be numerically unstable, leading to very large updates in pure regions, while the update here lies in the range $[-1, 1]$. Empirical evidence suggests (see Section 7) that this more conservative algorithm has similar performance to both the Real AdaBoost and LogitBoost algorithms, and often outperforms them both, especially when stability is an issue.

There is a strong similarity between the updates for the Gentle AdaBoost algorithm and those for the LogitBoost algorithm. Let $P = P(y = 1|x)$, and $p(x) = e^{F(x)}/(e^{F(x)} + e^{-F(x)})$. Then

$$\begin{aligned}\frac{E(e^{-yF(x)}y|x)}{E(e^{-yF(x)}|x)} &= \frac{e^{-F(x)}P - e^{F(x)}(1-P)}{e^{-F(x)}P + e^{F(x)}(1-P)} \\ &= \frac{P - p(x)}{(1-p(x))P + p(x)(1-P)}.\end{aligned} \tag{37}$$

The analogous expression for LogitBoost from (34) is

$$\frac{1}{2}\frac{P - p(x)}{p(x)(1-p(x))}. \tag{38}$$

At $p(x) \approx \frac{1}{2}$ these are nearly the same, but they differ as the $p(x)$ become extreme. For example, if $P \approx 1$ and $p(x) \approx 0$, (38) blows up, while (37) is about 1 (and always falls in $[-1, 1]$).

5. Multiclass procedures. Here we explore extensions of boosting to classification with multiple classes. We start off by proposing a natural generalization of the two-class symmetric logistic transformation, and then consider specific algorithms. In this context Schapire and Singer (1998) define J responses y_j for a J class problem, each taking values in $\{-1, 1\}$. Similarly the *indicator response vector* with elements y_j^* is more standard in the statistics literature. Assume the classes are mutually exclusive.

DEFINITION 1. For a J class problem let $p_j(x) = P(y_j = 1|x)$. We define the symmetric multiple logistic transformation

$$F_j(x) = \log p_j(x) - \frac{1}{J}\sum_{k=1}^{J} \log p_k(x). \tag{39}$$

Equivalently,

$$p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}, \qquad \sum_{k=1}^{J} F_k(x) = 0. \tag{40}$$

The centering condition in (40) is for numerical stability only; it simply pins the F_j down, else we could add an arbitrary constant to each F_j and the probabilities remain the same. The equivalence of these two definitions is easily established, as well as the equivalence with the two-class case.

Schapire and Singer (1998) provide several generalizations of AdaBoost for the multiclass case, and also refer to other proposals [Freund and Schapire (1997), Schapire (1997)]; we describe their *AdaBoost.MH* algorithm (see Algorithm 5), since it seemed to dominate the others in their empirical studies. We then connect it to the models presented here. We will refer to the augmented variable in Algorithm 5 as the "class" variable C. We make a few observations:

1. The population version of this algorithm minimizes $\sum_{j=1}^{J} Ee^{-y_j F_j(x)}$, which is equivalent to running separate population boosting algorithms on each of the J problems of size N obtained by partitioning the $N \times J$ samples in the obvious fashion. This is seen trivially by first conditioning on $C = j$, and then $x|C = j$, when computing conditional expectations.
2. The same is almost true for a tree-based algorithm. We see this because:

   (a) If the first split is on C, either a J-nary split if permitted, or else $J - 1$ binary splits, then the subtrees are identical to separate trees grown to each of the J groups. This will always be the case for the first tree.
   (b) If a tree does not split on C anywhere on the path to a terminal node, then that node returns a function $f_m(x, j) = g_m(x)$ that contributes nothing to the classification decision. However, as long as a tree includes a split on C at least once on every path to a terminal node, it will make a contribution to the classifier for all input feature values.

   The advantage or disadvantage of building one large tree using class label as an additional input feature is not clear. No motivation is provided. We therefore implement AdaBoost.MH using the more traditional direct approach of building J separate trees to minimize $\sum_{j=1}^{J} E \exp(-y_j F_j(x))$

   We have thus shown

RESULT 5. *The AdaBoost.MH algorithm for a J-class problem fits J uncoupled additive logistic models, $G_j(x) = \frac{1}{2} \log p_j(x)/(1 - p_j(x))$, each class against the rest.*

AdaBoost.MH [Schapire and Singer (1998)]

1. Expand the original N observations into $N \times J$ pairs $((x_i, 1), y_{i1})$, $((x_i, 2), y_{i2}), \ldots, ((x_i, J), y_{iJ})$, $i = 1, \ldots, N$. Here y_{ij} is the $\{-1, 1\}$ response for class j and observation i.
2. Apply Real AdaBoost to the augmented dataset, producing a function $F: \mathscr{X} \times (1, \ldots, J) \mapsto R$; $F(x, j) = \sum_m f_m(x, j)$.
3. Output the classifier $\arg\max_j F(x, j)$.

ALGORITHM 5. *The AdaBoost.MH algorithm converts the J class problem into that of estimating a two class classifier on a training set J times as large, with an additional "feature" defined by the set of class labels.*

In principal this parametrization is fine, since $G_j(x)$ is monotone in $p_j(x)$. However, we are estimating the $G_j(x)$ in an uncoupled fashion, and there is no guarantee that the implied probabilities sum to 1. We give some examples where this makes a difference, and AdaBoost.MH performs more poorly than an alternative coupled likelihood procedure.

Schapire and Singer's AdaBoost.MH was also intended to cover situations where observations can belong to more than one class. The "MH" represents "Multi-Label Hamming", Hamming loss being used to measure the errors in the space of 2^J possible class labels. In this context fitting a separate classifier for each label is a reasonable strategy. However, Schapire and Singer also propose using AdaBoost.MH when the class labels are mutually exclusive, which is the focus in this paper.

Algorithm 6 is a natural generalization of Algorithm 3 for fitting the J-class logistic regression model (40).

RESULT 6. *The LogitBoost algorithm* (*J classes*, *population version*) *uses quasi-Newton steps for fitting an additive symmetric logistic model by maximum-likelihood*.

LogitBoost (*J classes*)

1. Start with weights $w_{ij} = 1/N,\ i = 1, \ldots, N,\ j = 1, \ldots, J,\ F_j(x) = 0$ and $p_j(x) = 1/J\ \forall\ j$.

2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Repeat for $j = 1, \ldots, J$:

      (i) Compute working responses and weights in the jth class,

$$z_{ij} = \frac{y^*_{ij} - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))},$$

$$w_{ij} = p_j(x_i)(1 - p_j(x_i)).$$

      (ii) Fit the function $f_{mj}(x)$ by a weighted least-squares regression of z_{ij} to x_i with weights w_{ij}.

   (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x))$, and $F_j(x) \leftarrow F_j(x) + f_{mj}(x)$.

   (c) Update $p_j(x)$ via (40).

3. Output the classifier $\arg\max_j F_j(x)$.

ALGORITHM 6. *An adaptive Newton algorithm for fitting an additive multiple logistic regression model*.

DERIVATION.

1. We first give the score and Hessian for the population Newton algorithm corresponding to a standard multilogit parametrization

$$G_j(x) = \log \frac{P(y_j^* = 1|x)}{P(y_J^* = 1|x)}$$

with $G_J(x) = 0$ (and the choice of J for the *base class* is arbitrary). The expected conditional log-likelihood is:

$$E(l(G+g)|x) = \sum_{j=1}^{J-1} E(y_j^*|x)(G_j(x) + g_j(x)) - \log\left(1 + \sum_{k=1}^{J-1} e^{G_k(x)+g_k(x)}\right),$$

$$s_j(x) = E(y_j^* - p_j(x)|x),\ j = 1, \ldots, J-1,$$

$$H_{j,k}(x) = -p_j(x)(\delta_{jk} - p_k(x)),\ j, k = 1, \ldots, J-1.$$

2. Our quasi-Newton update amounts to using a diagonal approximation to the Hessian, producing updates:

$$g_j(x) \leftarrow \frac{E(y_j^* - p_j(x)|x)}{p_j(x)(1 - p_j(x))},\ j = 1, \ldots, J-1.$$

3. To convert to the symmetric parametrization, we would note that $g_J = 0$, and set $f_j(x) = g_j(x) - (1/J)\sum_{k=1}^{J} g_k(x)$. However, this procedure could be applied using any class as the base, not just the Jth. By averaging over all choices for the base class, we get the update

$$f_j(x) = \left(\frac{J-1}{J}\right)\left(\frac{E(y_j^* - p_j(x)|x)}{p_j(x)(1-p_j(x))} - \frac{1}{J}\sum_{k=1}^{J} \frac{E(y_k^* - p_k(x)|x)}{p_k(x)(1-p_k(x))}\right).$$

For more rigid parametric models and full Newton stepping, this symmetrization would be redundant. With quasi-Newton steps and adaptive (tree based) models, the symmetrization removes the dependence on the choice of the base class.

6. Simulation studies. In this section the four flavors of boosting outlined above are applied to several artificially constructed problems. Comparisons based on real data are presented in Section 7.

An advantage of comparisons made in a simulation setting is that all aspects of each example are known, including the Bayes error rate and the complexity of the decision boundary. In addition, the population expected error rates achieved by each of the respective methods can be estimated to arbitrary accuracy by averaging over a large number of different training and test data

sets drawn from the population. The four boosting methods compared here are:

DAB: Discrete AdaBoost—Algorithm 1.

RAB: Real AdaBoost—Algorithm 2.

LB: LogitBoost—Algorithms 3 and 6.

GAB: Gentle AdaBoost—Algorithm 4.

DAB, RAB and GAB handle multiple classes using the AdaBoost.MH approach.

In an attempt to differentiate performance, all of the simulated examples involve fairly complex decision boundaries. The ten input features for all examples are randomly drawn from a ten-dimensional standard normal distribution $x \sim N^{10}(0, I)$. For the first three examples the decision boundaries separating successive classes are nested concentric ten-dimensional spheres constructed by thresholding the squared-radius from the origin

$$r^2 = \sum_{j=1}^{10} x_j^2. \tag{41}$$

Each class C_k $(1 \leq k \leq K)$ is defined as the subset of observations

$$C_k = \{x_i | t_{k-1} \leq r_i^2 < t_k\} \tag{42}$$

with $t_0 = 0$ and $t_K = \infty$. The $\{t_k\}_1^{K-1}$ for each example were chosen so as to put approximately equal numbers of observations in each class. The training sample size is $N = K \cdot 1000$ so that approximately 1000 training observations are in each class. An independently drawn test set of 10,000 observations was used to estimate error rates for each training set. Averaged results over ten such independently drawn training–test set combinations were used for the final error rate estimates. The corresponding statistical uncertainties (standard errors) of these final estimates (averages) are approximately a line width on each plot.

Figure 3 (top left) compares the four algorithms in the two-class ($K = 2$) case using a two-terminal node decision tree ("stump") as the base classifier. Shown is error rate as a function of number of boosting iterations. The upper (black) line represents DAB and the other three nearly coincident lines are the other three methods (dotted red = RAB, short-dashed green = LB, and long-dashed blue = GAB). Note that the somewhat erratic behavior of DAB, especially for less than 200 iterations, is not due to statistical uncertainty. For less than 400 iterations LB has a minuscule edge, after that it is a dead heat with RAB and GAB. DAB shows substantially inferior performance here with roughly twice the error rate at all iterations.

Figure 3 (lower left) shows the corresponding results for three classes ($K = 3$) again with two-terminal node trees. Here the problem is more difficult as represented by increased error rates for all four methods, but their relationship is roughly the same: the upper (black) line represents DAB and the other

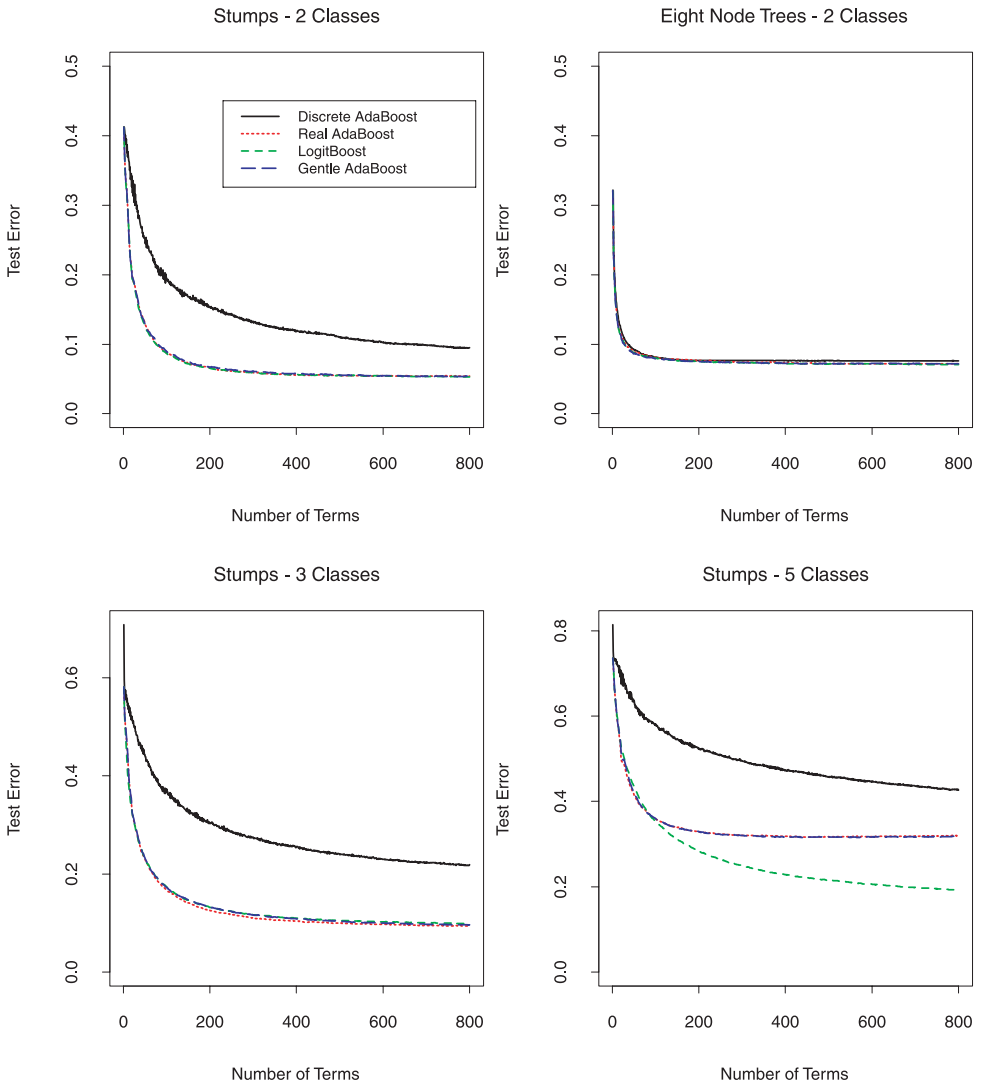

FIG. 3. *Test error curves for the simulation experiment with an additive decision boundary, as described in* (42). *In all panels except the top right, the solid curve* (*representing Discrete AdaBoost*) *lies alone above the other three curves.*

three nearly coincident lines are the other three methods. The situation is somewhat different for larger number of classes. Figure 3 (lower right) shows results for $K = 5$ which are typical for $K \geq 4$. As before, DAB incurs much higher error rates than all the others, and RAB and GAB have nearly identical performance. However, the performance of LB relative to RAB and GAB has changed. Up to about 40 iterations it has the same error rate. From 40 to about 100 iterations LB's error rates are slightly higher than the other two. After 100 iterations the error rate for LB continues to improve whereas that

for RAB and GAB level off, decreasing much more slowly. By 800 iterations the error rate for LB is 0.19 whereas that for RAB and GAB is 0.32. Speculation as to the reason for LB's performance gain in these situations is presented below.

In the above examples a stump was used as the base classifier. One might expect the use of larger trees would do better for these rather complex problems. Figure 3 (top right) shows results for the two-class problem, here boosting trees with eight terminal nodes. These results can be compared to those for stumps in Figure 3 (top left). Initially, error rates for boosting eight-node trees decrease much more rapidly than for stumps, with each successive iteration, for all methods. However, the error rates quickly level off and improvement is very slow after about 100 iterations. The overall performance of DAB is much improved with the bigger trees, coming close to that of the other three methods. As before RAB, GAB and LB exhibit nearly identical performance. Note that at each iteration the eight-node tree model consists of four times the number of additive terms as does the corresponding stump model. This is why the error rates decrease so much more rapidly in the early iterations. In terms of model complexity (and training time), a 100-iteration model using eight-terminal node trees is equivalent to a 400-iteration stump model.

Comparing the top two panels in Figure 3, one sees that for RAB, GAB and LB the error rate using the bigger trees (0.072) is in fact 33% higher than that for stumps (0.054) at 800 iterations, even though the former is four times more complex. This seemingly mysterious behavior is easily understood by examining the nature of the decision boundary separating the classes. The Bayes decision boundary between two classes is the set

$$\left\{x\colon \log\frac{P(y=1|x)}{P(y=-1|x)}=0\right\} \tag{43}$$

or simply $\{x\colon B(x)=0\}$. To approximate this set it is sufficient to estimate the logit $B(x)$, or any monotone transformation of $B(x)$, as closely as possible. As discussed above, boosting produces an additive logistic model whose component functions are represented by the base classifier. With stumps as the base classifier, each component function has the form

$$f_m(x)=c_m^L 1_{[x_j\le t_m]}+c_m^R 1_{[x_j>t_m]} \tag{44}$$

$$=f_m(x_j) \tag{45}$$

if the mth stump chose to split on coordinate j. Here t_m is the split-point, and c_m^L and c_m^R are the weighted means of the response in the left and right terminal nodes. Thus the model produced by boosting stumps is additive in the *original* features,

$$F(x)=\sum_{j=1}^{p} g_j(x_j), \tag{46}$$

where $g_j(x_j)$ adds together all those stumps involving x_j (and is 0 if none exist).

Examination of (41) and (42) reveals that an optimal decision boundary for the above examples is also additive in the original features, with $f_j(x_j) = x_j^2 + constant$. Thus, in the context of decision trees, stumps are ideally matched to these problems; larger trees are not needed. However boosting larger trees need not be counterproductive in this case if all of the splits in each individual tree are made on the same predictor variable. This would also produce an additive model in the *original* features (46). However, due to the forward greedy stagewise strategy used by boosting, this is not likely to happen if the decision boundary function involves more than one predictor; each individual tree will try to do its best to involve all of the important predictors. Owing to the nature of decision trees, this will produce models with *interaction effects*; most terms in the model will involve products in more than one variable. Such nonadditive models are not as well suited for approximating truly additive decision boundaries such as (41) and (42). This is reflected in increased error rate as observed in Figure 3.

The above discussion also suggests that if the decision boundary separating pairs of classes were inherently *nonadditive* in the predictors, then boosting stumps would be less advantageous than using larger trees. A tree with m terminal nodes can produce basis functions with a maximum interaction order of $\min(m-1, p)$ where p is the number of predictor features. These higher order basis functions provide the possibility to more accurately estimate those decision boundaries $B(x)$ with high-order interactions. The purpose of the next example is to verify this intuition. There are two classes ($K = 2$) and 5000 training observations with the $\{x_i\}_1^{5000}$ drawn from a ten-dimensional normal distribution as in the previous examples. Class labels were randomly assigned to each observation with log-odds

$$\log\left(\frac{\Pr[y=1|x]}{\Pr[y=-1|x]}\right) = 10\sum_{j=1}^{6} x_j\left(1+\sum_{l=1}^{6}(-1)^l x_l\right). \tag{47}$$

Approximately equal numbers of observations are assigned to each of the two classes, and the Bayes error rate is 0.046. The decision boundary for this problem is a complicated function of the first six predictor variables involving all of them in second-order interactions of equal strength. As in the above examples, test sets of 10,000 observations was used to estimate error rates for each training set, and final estimates were averages over ten replications.

Figure 4 (top left) shows test-error rate as a function of iteration number for each of the four boosting methods using stumps. As in the previous examples, RAB and GAB track each other very closely. DAB begins very slowly, being dominated by all of the others until around 180 iterations, where it passes below RAB and GAB. LB mostly dominates, having the lowest error rate until about 650 iterations. At that point DAB catches up and by 800 iterations it may have a very slight edge. However, none of these boosting methods perform well with stumps on this problem, the best error rate being 0.35.

Figure 4 (top right) shows the corresponding plot when four terminal node trees are boosted. Here there is a dramatic improvement with all of the four

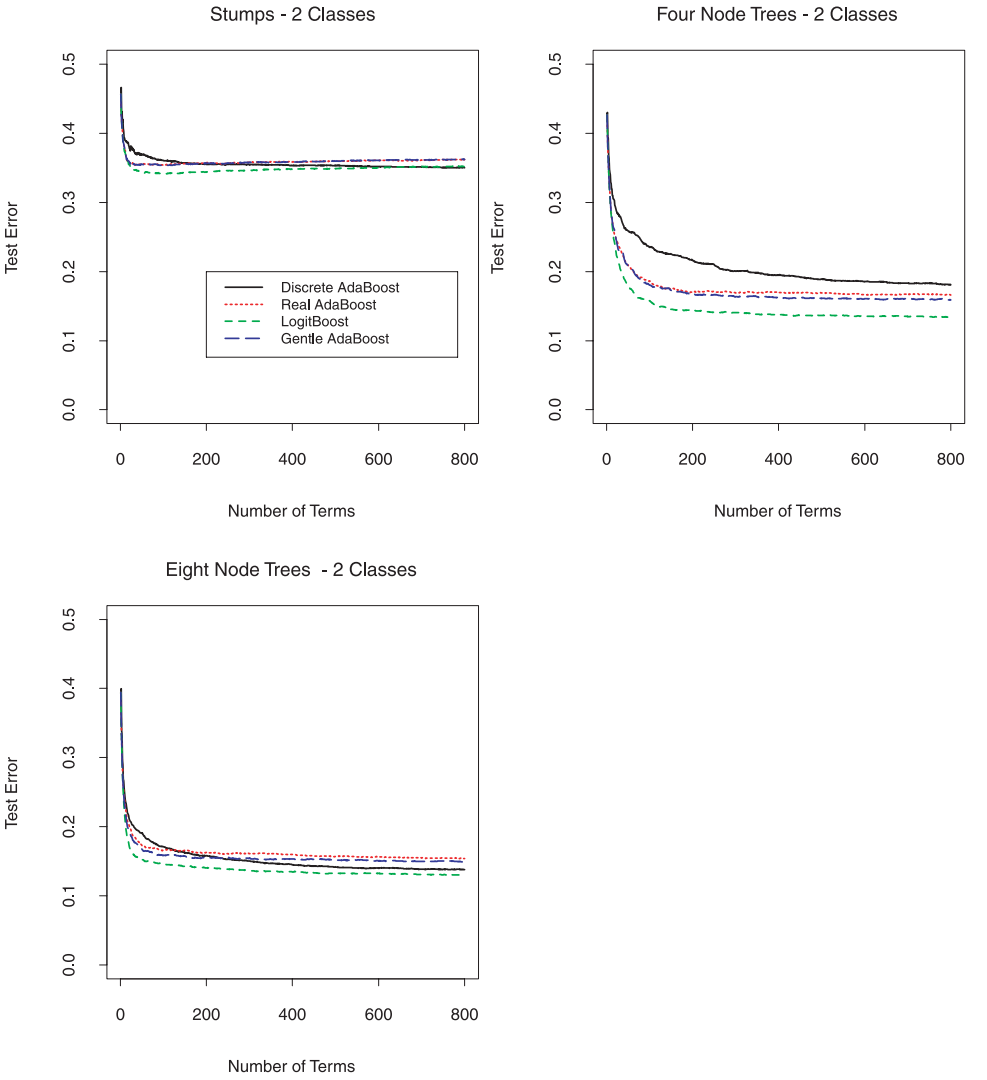

FIG. 4. *Test error curves for the simulation experiment with a nonadditive decision boundary, as described in* (47).

methods. For the first time there is some small differentiation between RAB and GAB. At nearly all iterations the performance ranking is LB best, followed by GAB, RAB and DAB in order. At 800 iterations LB achieves an error rate of 0.134. Figure 4 (lower left) shows results when eight terminal node trees are boosted. Here, error rates are generally further reduced with LB improving the least (0.130), but still dominating. The performance ranking among the other three methods changes with increasing iterations; DAB overtakes RAB

at around 150 iterations and GAB at about 230 becoming fairly close to LB by 800 iterations with an error rate of 0.138.

Although limited in scope, these simulation studies suggest several trends. They explain why boosting stumps can sometimes be superior to using larger trees, and suggest situations where this is likely to be the case; that is when decision boundaries $B(x)$ can be closely approximated by functions that are additive in the original predictor features. When higher order interactions are required, stumps exhibit poor performance. These examples illustrate the close similarity between RAB and GAB. In all cases the difference in performance between DAB and the others decreases when larger trees and more iterations are used, sometimes overtaking the others. More generally, relative performance of these four methods depends on the problem at hand in terms of the nature of the decision boundaries, the complexity of the base classifier and the number of boosting iterations.

The superior performance of LB in Figure 3 (lower right) appears to be a consequence of the multiclass logistic model (Algorithm 6). All of the other methods use the asymmetric AdaBoost.MH strategy (Algorithm 5) of building separate two-class models for each individual class against the pooled complement classes. Even if the decision boundaries separating all class pairs are relatively simple, pooling classes can produce complex decision boundaries that are difficult to approximate [Friedman (1996)]. By considering all of the classes simultaneously, the symmetric multiclass model is better able to take advantage of simple pairwise boundaries when they exist [Hastie and Tibshirani (1998)]. As noted above, the pairwise boundaries induced by (41) and (42) are simple when viewed in the context of additive modeling, whereas the pooled boundaries are more complex; they cannot be well approximated by functions that are additive in the original predictor variables.

The decision boundaries associated with these examples were deliberately chosen to be geometrically complex in an attempt to elicit performance differences among the methods being tested. Such complicated boundaries are not likely to often occur in practice. Many practical problems involve comparatively simple boundaries [Holte (1993)]; in such cases performance differences will still be situation dependent, but correspondingly less pronounced.

7. Some experiments with real world data. In this section we show the results of running the four fitting methods: LogitBoost, Discrete AdaBoost, Real AdaBoost and Gentle AdaBoost on a collection of datasets from the UC-Irvine machine learning archive, plus a popular simulated dataset. The base learner is a tree in each case, with either two or eight terminal nodes. For comparison, a single decision tree was also fit (using the tree function in Splus), with the tree size determined by 5-fold cross-validation.

The datasets are summarized in Table 1. The test error rates are shown in Table 2 for the smaller datasets, and in Table 3 for the larger ones. The `vowel`, `sonar`, `satimage` and `letter` datasets come with a prespecified test set. The `waveform` data is simulated, as described in Breiman, Friedman, Olshen and

TABLE 1
Datasets used in the experiments

Data set	# Train	# Test	# Inputs	# Classes
Vowel	528	462	10	11
Breast cancer	699	5-fold CV	9	2
Ionosphere	351	5-fold CV	34	2
Glass	214	5-fold CV	10	7
Sonar	210	5-fold CV	60	2
Waveform	300	5000	21	3
Satimage	4435	2000	36	6
Letter	16000	4000	16	26

Stone (1984). For the others, 5-fold cross-validation was used to estimate the test error.

It is difficult to discern trends on the small datasets (Table 2) because all but quite large observed differences in performance could be attributed to sampling fluctuations. On the `vowel`, `breast cancer`, `ionosphere`, `sonar` and `waveform` data, purely additive stump models seem to perform comparably to the larger (eight-node) trees. The `glass` data seems to benefit a little from larger trees. There is no clear differentiation in performance among the boosting methods.

On the larger data sets (Table 3) clearer trends are discernible. For the `satimage` data the eight-node tree models are only slightly, but significantly, more accurate than the purely additive models. For the `letter` data there is no contest. Boosting stumps is clearly inadequate. There is no clear differentiation among the boosting methods for eight-node trees. For the stumps, LogitBoost, Real AdaBoost and Gentle AdaBoost have comparable perfor-
break mance, distinctly superior to Discrete AdaBoost. This is consistent with the results of the simulation study (Section 6).

Except perhaps for Discrete AdaBoost, the real data examples fail to demonstrate performance differences between the various boosting methods. This is in contrast to the simulated data sets of Section 6. There LogitBoost generally dominated, although often by a small margin. The inability of the real data examples to discriminate may reflect statistical difficulties in estimating subtle differences with small samples. Alternatively, it may be that their underlying decision boundaries are all relatively simple [Holte (1993)] so that all reasonable methods exhibit similar performance.

8. Additive logistic trees. In most applications of boosting the base classifier is considered to be a primitive, repeatedly called by the boosting procedure as iterations proceed. The operations performed by the base classifier are the same as they would be in any other context given the same data and weights. The fact that the final model is going to be a linear combination of a large number of such classifiers is not taken into account. In particular, when using decision trees, the same tree growing and pruning algorithms are

TABLE 2
Test error rates on small real examples

Method	Iterations	2 Terminal Nodes			8 Terminal Nodes		
		50	**100**	**200**	**50**	**100**	**200**
`Vowel`	CART error = 0.642						
LogitBoost		0.532	0.524	0.511	0.517	0.517	0.517
Real AdaBoost		0.565	0.561	0.548	0.496	0.496	0.496
Gentle AdaBoost		0.556	0.571	0.584	0.515	0.496	0.496
Discrete AdaBoost		0.563	0.535	0.563	0.511	0.500	0.500
`Breast`	CART error = 0.045						
LogitBoost		0.028	0.031	0.029	0.034	0.038	0.038
Real AdaBoost		0.038	0.038	0.040	0.032	0.034	0.034
Gentle AdaBoost		0.037	0.037	0.041	0.032	0.031	0.031
Discrete AdaBoost		0.042	0.040	0.040	0.032	0.035	0.037
`Ion`	CART error = 0.076						
LogitBoost		0.074	0.077	0.071	0.068	0.063	0.063
Real AdaBoost		0.068	0.066	0.068	0.054	0.054	0.054
Gentle AdaBoost		0.085	0.074	0.077	0.066	0.063	0.063
Discrete AdaBoost		0.088	0.080	0.080	0.068	0.063	0.063
`Glass`	CART error = 0.400						
LogitBoost		0.266	0.257	0.266	0.243	0.238	0.238
Real AdaBoost		0.276	0.247	0.257	0.234	0.234	0.234
Gentle AdaBoost		0.276	0.261	0.252	0.219	0.233	0.238
Discrete AdaBoost		0.285	0.285	0.271	0.238	0.234	0.243
`Sonar`	CART error = 0.596						
LogitBoost		0.231	0.231	0.202	0.163	0.154	0.154
Real AdaBoost		0.154	0.163	0.202	0.173	0.173	0.173
Gentle AdaBoost		0.183	0.183	0.173	0.154	0.154	0.154
Discrete AdaBoost		0.154	0.144	0.183	0.163	0.144	0.144
`Waveform`	CART error = 0.364						
LogitBoost		0.196	0.195	0.206	0.192	0.191	0.191
Real AdaBoost		0.193	0.197	0.195	0.185	0.182	0.182
Gentle AdaBoost		0.190	0.188	0.193	0.185	0.185	0.186
Discrete AdaBoost		0.188	0.185	0.191	0.186	0.183	0.183

TABLE 3
Test error rates on larger data examples

Method	Terminal Nodes	Iterations 20	50	100	200	Fraction
Satimage	CART error = 0.148					
LogitBoost	2	0.140	0.120	0.112	0.102	
Real AdaBoost	2	0.148	0.126	0.117	0.119	
Gentle AdaBoost	2	0.148	0.129	0.119	0.119	
Discrete AdaBoost	2	0.174	0.156	0.140	0.128	
LogitBoost	8	0.096	0.095	0.092	0.088	
Real AdaBoost	8	0.105	0.102	0.092	0.091	
Gentle AdaBoost	8	0.106	0.103	0.095	0.089	
Discrete AdaBoost	8	0.122	0.107	0.100	0.099	
Letter	CART error = 0.124					
LogitBoost	2	0.250	0.182	0.159	0.145	0.06
Real AdaBoost	2	0.244	0.181	0.160	0.150	0.12
Gentle AdaBoost	2	0.246	0.187	0.157	0.145	0.14
Discrete AdaBoost	2	0.310	0.226	0.196	0.185	0.18
LogitBoost	8	0.075	0.047	0.036	0.033	0.03
Real AdaBoost	8	0.068	0.041	0.033	0.032	0.03
Gentle AdaBoost	8	0.068	0.040	0.030	0.028	0.03
Discrete AdaBoost	8	0.080	0.045	0.035	0.029	0.03

generally employed. Sometimes alterations are made (such as no pruning) for programming convenience and speed.

When boosting is viewed in the light of additive modeling, however, this greedy approach can be seen to be far from optimal in many situations. As discussed in Section 6 the goal of the final classifier is to produce an accurate approximation to the decision boundary function $B(x)$. In the context of boosting, this goal applies to the final additive model, not to the individual terms (base classifiers) at the time they were constructed. For example, it was seen in Section 6 that if $B(x)$ was close to being additive in the original predictive features, then boosting stumps was optimal since it produced an approximation with the same structure. Building larger trees increased the error rate of the final model because the resulting approximation involved high-order interactions among the features. The larger trees optimized error rates of the individual base classifiers, given the weights at that step, and even produced lower unweighted error rates in the early stages. However, after a sufficient number of boosts, the stump-based model achieved superior performance.

More generally, one can consider an expansion of the decision boundary function in a functional ANOVA decomposition [Friedman (1991)]

$$B(x) = \sum_j f_j(x_j) + \sum_{j,k} f_{jk}(x_j, x_k) + \sum_{j,k,l} f_{jkl}(x_j, x_k, x_l) + \cdots. \tag{48}$$

The first sum represents the closest function to $B(x)$ that is additive in the original features, the first two represent the closest approximation involving at most two-feature interactions, the first three represent three-feature interactions, and so on. If $B(x)$ can be accurately approximated by such an expansion, truncated at low interaction order, then allowing the base classifier to produce higher order interactions can reduce the accuracy of the final boosted model. In the context of decision trees, higher order interactions are produced by deeper trees.

In situations where the true underlying decision boundary function admits a low order ANOVA decomposition, one can take advantage of this structure to improve accuracy by restricting the depth of the base decision trees to be not much larger than the actual interaction order of $B(x)$. Since this is not likely to be known in advance for any particular problem, this maximum depth becomes a "meta-parameter" of the procedure to be estimated by some model selection technique, such as cross-validation.

One can restrict the depth of an induced decision tree by using its standard pruning procedure, starting from the largest possible tree, but requiring it to delete enough splits to achieve the desired maximum depth. This can be computationally wasteful when this depth is small. The time required to build the tree is proportional to the depth of the largest possible tree before pruning. Therefore, dramatic computational savings can be achieved by simply stopping the growing process at the maximum depth, or alternatively at a maximum number of terminal nodes. The standard heuristic arguments in favor of growing large trees and then pruning do not apply in the context of boosting. Shortcomings in any individual tree can be compensated by trees grown later in the boosting sequence.

If a truncation strategy based on number of terminal nodes is to be employed, it is necessary to define an order in which splitting takes place. We adopt a "best-first" strategy. An optimal split is computed for each currently terminal node. The node whose split would achieve the greatest reduction in the tree building criterion is then actually split. This increases the number of terminal nodes by one. This continues until a maximum number M of terminal notes is induced. Standard computational tricks can be employed so that inducing trees in this order requires no more computation than other orderings commonly used in decision tree induction.

The truncation limit M is applied to all trees in the boosting sequence. It is thus a meta-parameter of the entire boosting procedure. An optimal value can be estimated through standard model selection techniques such as minimizing cross-validated error rate of the final boosted model. We refer to this combination of truncated best-first trees, with boosting, as "additive logistic trees" (ALT). Best-first trees were used in all of the simulated and real examples. One can compare results on the latter (Tables 2 and 3) to corresponding results reported by Dietterich [(1998), Table 1] on common data sets. Error rates achieved by ALT with very small truncation values are seen to compare quite favorably with other committee approaches using much larger trees at each boosting step. Even when error rates are the same, the computational savings

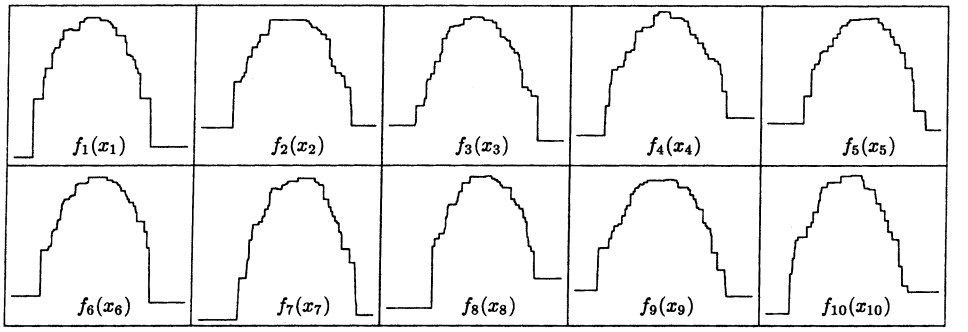

FIG. 5. *Coordinate functions for the additive logistic tree obtained by boosting (Logitboost) with stumps, for the two-class nested sphere example from Section* 6.

associated with ALT can be quite important in data mining contexts where large data sets cause computation time to become an issue.

Another advantage of low order approximations is model visualization. In particular, for models additive in the input features (46), the contribution of each feature x_j can be viewed as a graph of $g_j(x_j)$ plotted against x_j. Figure 5 shows such plots for the ten features of the two-class nested spheres example of Figure 3. The functions are shown for the first class concentrated near the origin; the corresponding functions for the other class are the negatives of these functions.

The plots in Figure 5 clearly show that the contribution to the log-odds of each individual feature is approximately quadratic, which matches the generating model (41) and (42).

When there are more than two classes, plots similar to Figure 5 can be made for each class and analogously interpreted. Higher order interaction models are more difficult to visualize. If there are at most two-feature interactions, the two-variable contributions can be visualized using contour or perspective mesh plots. Beyond two-feature interactions, visualization techniques are even less effective. Even when noninteraction (stump) models do not achieve the highest accuracy, they can be very useful as descriptive statistics owing to the interpretability of the resulting model.

9. Weight trimming. In this section we propose a simple idea and show that it can dramatically reduce computation for boosted models without sacrificing accuracy. Despite its apparent simplicity, this approach does not appear to be in common use [although similar ideas have been proposed before: Schapire (1990), Freund (1995)]. At each boosting iteration there is a distribution of weights over the training sample. As iterations proceed, this distribution tends to become highly skewed towards smaller weight values. A larger fraction of the training sample becomes correctly classified with increasing confidence, thereby receiving smaller weights. Observations with very low relative weight have little impact on training of the base classifier; only those that carry the dominant proportion of the weight mass are influen-

tial. The fraction of such high weight observations can become very small in later iterations. This suggests that at any iteration one can simply delete from the training sample the large fraction of observations with very low weight without having much effect on the resulting induced classifier. However, computation is reduced since it tends to be proportional to the size of the training sample, regardless of weights.

At each boosting iteration, training observations with weight w_i less than a threshold $w_i < t(\beta)$ are not used to train the classifier. We take the value of $t(\beta)$ to be the βth quantile of the weight distribution over the training data at the corresponding iteration. That is, only those observations that carry the fraction $1 - \beta$ of the total weight mass are used for training. Typically $\beta \in [0.01, 0.1]$ so that the data used for training carries from 90 to 99% of the total weight mass. Note that the weights for *all* training observations are recomputed at each iteration. Observations deleted at a particular iteration may therefore reenter at later iterations if their weights subsequently increase relative to other observations.

Figure 6 (left panel) shows test-error rate as a function of iteration number for the `letter` recognition problem described in Section 7, here using Gentle AdaBoost and eight-node trees as the base classifier. Two error rate curves are shown. The black solid one represents using the full training sample at each iteration ($\beta = 0$), whereas the blue dashed curve represents the corresponding error rate for $\beta = 0.1$. The two curves track each other very closely, especially at the later iterations. Figure 6 (right panel) shows the corresponding fraction of observations used to train the base classifier as a function of iteration number. Here the two curves are not similar. With $\beta = 0.1$ the number of observations used for training drops very rapidly reaching roughly 5% of the total at 20 iterations. By 50 iterations it is down to about 3% where it stays throughout the rest of the boosting procedure. Thus, computation is reduced

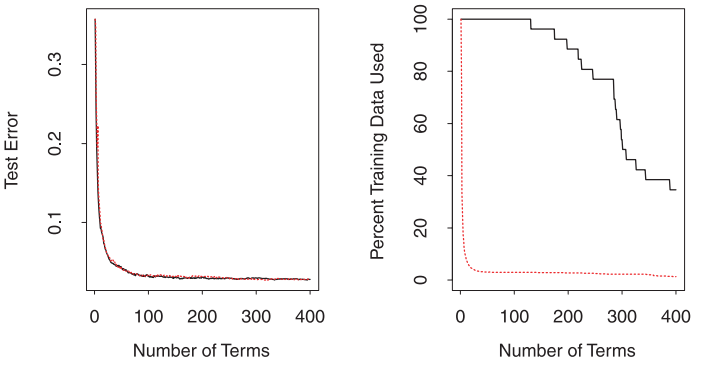

FIG. 6. *The left panel shows the test error for the letter recognition problem as a function of iteration number. The black solid curve uses all the training data, the red dashed curve uses a subset based on weight thresholding. The right panel shows the percent of training data used for both approaches. The upper curve steps down, because training can stop for an entire class if it is fit sufficiently well* (*see text*).

by over a factor of 30 with no apparent loss in classification accuracy. The reason why sample size in this case decreases for $\beta = 0$ after 150 iterations is that if all of the observations in a particular class are classified correctly with very high confidence $[F_k > 15 + \log(N)]$ training for that class stops, and continues only for the remaining classes. At 400 iterations, 12 classes remained of the original 26 classes.

The last column labeled *fraction* in Table 3 for the letter-recognition problem shows the average fraction of observations used in training the base classifiers over the 200 iterations, for all boosting methods and tree sizes. For eight-node trees, all methods behave as shown in Figure 6. With stumps, LogitBoost uses considerably less data than the others and is thereby correspondingly faster.

This is a genuine property of LogitBoost that sometimes gives it an advantage with weight trimming. Unlike the other methods, the LogitBoost weights $w_i = p_i(1 - p_i)$ do not in any way involve the class outputs y_i; they simply measure nearness to the currently estimated decision boundary $F_M(x) = 0$. Discarding small weights thus retains only those training observations that are estimated to be close to the boundary. For the other three procedures the weight is monotone in $-y_i F_M(x_i)$. This gives highest weight to currently misclassified training observations, especially those far from the boundary. If after trimming, the fraction of observations remaining is less than the error rate, the subsample passed to the base learner will be highly unbalanced containing very few correctly classified observations. This imbalance seems to inhibit learning. No such imbalance occurs with LogitBoost since near the decision boundary, correctly and misclassified observations appear in roughly equal numbers.

As this example illustrates, very large reductions in computation for boosting can be achieved by this simple trick. A variety of other examples (not shown) exhibit similar behavior with all boosting methods. Note that other committee approaches to classification such as bagging [Breiman (1996)] and randomized trees [Dietterich (1998)] while admitting parallel implementations, cannot take advantage of this approach to reduce computation.

10. Further generalizations of boosting. We have shown above that AdaBoost fits an additive model, optimizing a criterion similar to binomial log-likelihood, via an adaptive Newton method. This suggests ways in which the boosting paradigm may be generalized. First, the Newton step can be replaced by a gradient step, slowing down the fitting procedure. This can reduce susceptibility to overfitting and lead to improved performance. Second, any smooth loss function can be used: for regression, squared error is natural, leading to the "fitting of residuals" boosting algorithm mentioned in the introduction. However, other loss functions might have benefits, for example, tapered squared error based on Huber's robust influence function [Huber (1964)]. The resulting procedure is a fast, convenient method for resistant fitting of additive models. Details of these generalizations may be found in Friedman (1999).

11. Concluding remarks. In order to understand a learning procedure statistically it is necessary to identify two important aspects: its structural model and its error model. The former is most important since it determines the function space of the approximator, thereby characterizing the class of functions or hypotheses that can be accurately approximated with it. The error model specifies the distribution of random departures of sampled data from the structural model. It thereby defines the criterion to be optimized in the estimation of the structural model.

We have shown that the structural model for boosting is additive on the logistic scale with the base learner providing the additive components. This understanding alone explains many of the properties of boosting. It is no surprise that a large number of such (jointly optimized) components defines a much richer class of learners than one of them alone. It reveals that in the context of boosting all base learners are not equivalent, and there is no universally best choice over all situations. As illustrated in Section 6, the base learners need to be chosen so that the resulting additive expansion matches the particular decision boundary encountered. Even in the limited context of boosting decision trees the interaction order, as characterized by the number of terminal nodes, needs to be chosen with care. Purely additive models induced by decision stumps are sometimes, but not always, the best. However, we conjecture that boundaries involving very high-order interactions will rarely be encountered in practice. This motivates our additive logistic trees (ALT) procedure described in Section 8.

The error model for two-class boosting is the obvious one for binary variables, namely the Bernoulli disribution. We show that the AdaBoost procedures maximize a criterion that is closely related to expected log-Bernoulli likelihood, having the identical solution in the distributional (L_2) limit of infinite data. We derived a more direct procedure for maximizing this log-likelihood (LogitBoost) and show that it exhibits properties nearly identical to those of Real AdaBoost.

In the multiclass case, the AdaBoost procedures maximize a separate Bernoulli likelihood for each class versus the others. This is a natural choice and is especially appropriate when observations can belong to more than one class [Schapire and Singer (1998)]. In the more usual setting of a unique class label for each observation, the symmetric multinomial distribution is a more appropriate error model. We develop a multiclass LogitBoost procedure that maximizes the corresponding log-likelihood by quasi-Newton stepping. We show through simulated examples that there exist settings where this approach leads to superior performance, although none of these situations seems to have been encountered in the set of real data examples used for illustration; the performance of both approaches had quite similar performance over these examples.

The concepts developed in this paper suggest that there is very little, if any, connection between (deterministic) weighted boosting and other (randomized) ensemble methods such as bagging [Breiman (1996)] and randomized trees [Dietterich (1998)]. In the language of least-squares regression, the latter are

purely "variance" reducing procedures intended to mitigate instability, especially that associated with decision trees. Boosting on the other hand seems fundamentally different. It appears to be mainly a "bias" reducing procedure, intended to increase the flexibility of stable (highly biased) weak learners by incorporating them in a jointly fitted additive expansion.

The distinction becomes less clear [Breiman (1998a)] when boosting is implemented by finite weighted random sampling instead of weighted optimization. The advantages or disadvantages of introducing randomization into boosting by drawing finite samples is not clear. If there turns out to be an advantage with randomization in some situations, then the degree of randomization, as reflected by the sample size, is an open question. It is not obvious that the common choice of using the size of the original training sample is optimal in all (or any) situations.

One fascinating issue not covered in this paper is the fact that boosting, whatever flavor, seems resistant to overfitting. Some possible explanations are:

1. As the LogitBoost iterations proceed, the overall impact of changes introduced by $f_m(x)$ reduces. Only observations with appreciable weight determine the new functions—those near the decision boundary. By definition these observations have $F(x)$ near zero and can be affected by changes, while those in pure regions have large values of $|F(x)|$ and are less likely to be modified.
2. The stagewise nature of the boosting algorithms does not allow the full collection of parameters to be jointly fit, and thus has far lower variance than the full parameterization might suggest. In the computational learning theory literature this is explained in terms of VC dimension of the ensemble compared to that of each weak learner.

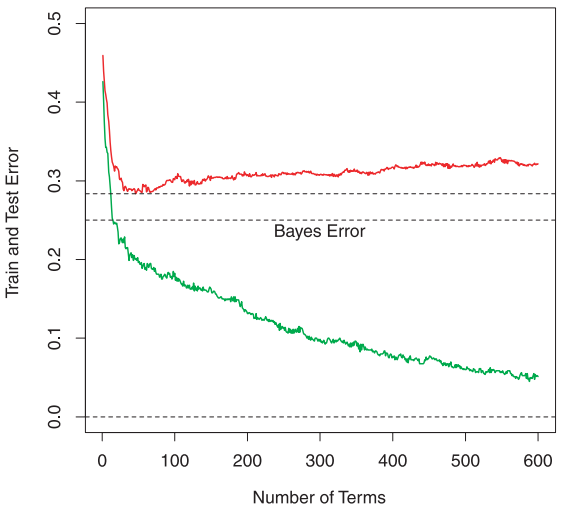

FIG. 7. *Real AdaBoost (stumps) on a noisy concentric-sphere problem, with* 400 *observations per class and Bayes error* 25%. *The test error (upper curve) increases after about fifty iterations.*

3. Classifiers are hurt less by overfitting than other function estimators [e.g., the famous risk bound of the 1-nearest-neighbor classifier, Cover and Hart (1967)].

Figure 7 shows a case where boosting does overfit. The data are generated from two ten-dimensional spherical Gaussians with the same mean, and variances chosen so that the Bayes error is 25% (400 samples per class). We used Real AdaBoost and stumps (the results were similar for all the boosting algorithms). After about 50 iterations the test error (slowly) increases.

Schapire, Freund, Bartlett and Lee (1998) suggest that the properties of AdaBoost, including its resistance to overfitting, can be understood in terms of classification margins. However, Breiman (1997) presents evidence counter to this explanation. Whatever the explanation, the empirical evidence is strong; the introduction of boosting by Schapire, Freund and colleagues has brought an exciting and important set of new ideas to the table.

Acknowledgments. We thank Andreas Buja for alerting us to the recent work on text classification at AT&T laboratories, Bogdan Popescu for illuminating discussions on PAC learning theory and Leo Breiman and Robert Schapire for useful comments on an earlier version of this paper. We also thank two anonymous referees and an Associate Editor for detailed and useful comments on an earlier draft of the paper.

REFERENCES

BREIMAN, L. (1996). Bagging predictors. *Machine Learning* **24** 123–140.

BREIMAN, L. (1997). Prediction games and arcing algorithms. Technical Report 504, Dept. Statistics, Univ. California, Berkeley.

BREIMAN, L. (1998a). Arcing classifiers (with discussion). *Ann. Statist.* **26** 801–849.

BREIMAN, L. (1998b). Combining predictors. Technical report, Dept. Statistics, Univ. California, Berkeley.

BREIMAN, L., FRIEDMAN, J., OLSHEN, R. and STONE, C. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, CA.

BUJA, A., HASTIE, T. and TIBSHIRANI, R. (1989). Linear smoothers and additive models (with discussion). *Ann. Statist.* **17** 453–555.

COVER, T. and HART, P. (1967). Nearest neighbor pattern classification. *Proc. IEEE Trans. Inform. Theory* **13** 21–27.

DIETTERICH, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*? 1–22.

FREUND, Y. (1995). Boosting a weak learning algorithm by majority. *Inform. and Comput.* **121** 256–285.

FREUND, Y. and SCHAPIRE, R. (1996a). Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory* 325–332.

FREUND, Y. and SCHAPIRE, R. E. (1996b). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference* 148–156. Morgan Kaufman, San Francisco.

FREUND, Y. and SCHAPIRE, R. E. (1997). A decision-theoretic generalization of online learning and an application to boosting. *J. Comput. System Sciences* **55**.

FRIEDMAN, J. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Statist.* **19** 1–141.

FRIEDMAN, J. (1996). Another approach to polychotomous classification. Technical report, Stanford Univ.

FRIEDMAN, J. (1999). Greedy function approximation: the gradient boosting machine. Technical report, Stanford Univ.

FRIEDMAN, J. and STUETZLE, W. (1981). Projection pursuit regression. *J. Amer. Statist. Assoc.* **76** 817–823.

HASTIE, T. and TIBSHIRANI, R. (1990). *Generalized Additive Models*. Chapman and Hall, London.

HASTIE, T. and TIBSHIRANI, R. (1998). Classification by pairwise coupling. *Ann. Statist.* **26** 451–471.

HASTIE, T., TIBSHIRANI, R. and BUJA, A. (1994). Flexible discriminant analysis by optimal scoring. *J. Amer. Statist. Assoc.* **89** 1255–1270.

HOLTE, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning* **11** 63–90.

HUBER, P. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.* **53** 73–101.

KEARNS, M. and VAZIRANI, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

MALLAT, S. and ZHANG, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Processing* **41** 3397–3415.

MCCULLAGH, P. and NELDER, J. (1989). *Generalized Linear Models*. Chapman and Hall, London.

SCHAPIRE, R. (1997). Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning* 313–321. Morgan Kaufman, San Francisco.

SCHAPIRE, R. E. (1990). The strength of weak learnability. *Machine Learning* **5** 197–227.

SCHAPIRE, R. E. and SINGER, Y. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*.

SCHAPIRE, R., FREUND, Y., BARTLETT, P. and LEE, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Statist.* **26** 1651–1686.

VALIANT, L. G. (1984). A theory of the learnable. *Comm. ACM* **27** 1134–1142.

DEPARTMENT OF STATISTICS
SEQUOIA HALL
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305
E-MAIL: {jnf, hastie, tibs}@stat.stanford.edu

DISCUSSION

LEO BREIMAN

University of California, Berkeley

The authors have presented a fascinating view of boosting and I congratulate them for the steps they have taken to clear up the mystery of why the AdaBoost algorithm works so well. Yet I have misgivings about their explanation (which I expressed earlier in my comments to them on receiving the first draft of their paper).

A crucial property of AdaBoost is that it almost never overfits the data no matter how many iterations it is run. The authors counterexample is the first convincing one I have seen but stands in contrast to the hundreds of data sets on which AdaBoost has not produced overfitting. Any complete explanation of AdaBoost has to explain this empirical fact. Ordinarily, in logistic regression,

if one adds more and more variables, increasing the likelihood at each step, then at some point overfitting sets in and the test-set error blossoms. The fact that this does not happen in AdaBoost or in LogitBoost is extraordinary. And, unless I am missing something, there is no explanation in the paper.

I approach the problem differently. In Breiman (1999a), I assume an ensemble of classifiers $\{h(x, \theta), \theta \in \Theta\}$. Let the sequence $\{\theta_n\}$ be iid selections from Θ according to some probability Q and let the classifiers $h(\mathbf{x}, \theta_1), h(\mathbf{x}, \theta_2), \ldots, h(\mathbf{x}, \theta_N)$ each cast a unit vote for the predicted class at $\mathbf{x}$. Then, under weak conditions, as N goes to infinity, the selected class converges a.s. to $\arg\max_j Q(h(\mathbf{x}, \theta) = j)$. This holds, for example, if the classifiers are trees, in which case I refer to the ensemble as a "random forest." The theorem is a simple consequence of the law of large numbers.

For example, in bagging, each classifier is grown on a bootstrap sample from the original training set. If the training set has K examples, let θ be a K-vector of nonnegative integers adding to K. Under Q, let θ have the distribution of sampling K times with replacement from K objects. Thus, using the results cited, bagging converges a.s. as the number of iterations becomes large.

AdaBoost is probably a random forest. Consider the following version of AdaBoost: given a probability P on the training set, use a fixed algorithm to construct a classifier $h(\mathbf{x}, P)$ using the weighted training set. The next probability P' on the training set depends only on P, the indices of the examples misclassified and the weighted misclassification error. Then there is a deterministic function Φ defined on the space of all probabilities on the training set such that $P' = \Phi(P)$. Assume that Φ is ergodic with invariant measure Q and that the voting is unweighted. Then the class selected converges a.s. to $\arg\max_j Q(h(\mathbf{x}, P) = j)$.

The implication is that the random forest constructed by letting the sequence $\{P_n\}$ be iid from Q is equivalent to AdaBoost. AdaBoost uses weighted voting, but a small alteration in the argument gives the same result. In Breiman (1999a), empirical evidence is given to show that AdaBoost is a random forest. This would settle the question of why AdaBoost does not overfit, but not the question of why AdaBoost gives such low test-set error.

There are many different random forests, and some are more accurate than others. For instance, Dieterrich (1998) constructs a random forest by choosing at random among the ten best splits at any node and shows empirically that this gives generally lower test-set error than bagging. In (1999a) we show that the test-set misclassification error is bounded above by an expression that depends on two parameters: the first is the expected correlation between pairs of classifiers, and the second is the expected accuracy of the individual classifiers.

One very effective way to reduce correlation is by random choice of features at each node. That is, suppose there are ten predictor variables. At each node, choose a small subgroup of them at random to split on. This idea first appeared in a paper by Amit and Geman (1997). Using this approach and 100 iterations gives the following test-set errors as compared to the best corresponding values for LogitBoost.

Test-Set Errors (%)

Dataset	LogitBoost	Random Forest
Breast	3.8	2.9
Ionosphere	6.3	7.1
Glass	23.8	20.6
Sonar	15.4	15.9
Waveform	19.1	17.2
Sat-images	11.2	8.6

The comparison is on runs I have already done. Since I have not done a run on the vowel dataset that is similar to that of the paper, it does not appear. Similarly, I have not run the letter dataset referred to. The major point of this comparison is to show that correlation and strength are the essential ingredients of an accurate ensemble, and that if boosting succeeds, it is because it gives low correlation between classifiers of reasonable strength. The fact that a similar algorithm maximizes the likelihood in a logistic setting is not the primary reason for its success.

So what is known about the factors leading to AdaBoost's success? One clear factor is that at each step it decouples itself from the previous step. This would intuitively seem to lead to low correlation between pairs of classifiers in AdaBoost considered as a random forest. However, this has not yet been analytically or empirically established.

A recently discovered empirical property of AdaBoost is mysterious but may lead to better understanding. At each iteration, the examples currently misclassified by AdaBoost are weighted more heavily for the next iteration, to encourage their correct classification. The commonly accepted interpretation, therefore, is that AdaBoost tries to classify correctly the hard-to-classify examples, and ignores the easy-to-classify examples. This interpretation is incorrect, and I have to admit some culpability.

Suppose AdaBoost is run for a while. Then for each case in the training set, look at the weighted proportion of times that it has been misclassified (the weights are just those that AdaBoost assigns to each tree in the ensemble). For all data sets I've looked at so far, these proportions are nearly the same over the entire training set. That is, AdaBoost is an *equalizer*. When it puts more emphasis on points that have been recently misclassified, it is trying to get them correctly classified next time around [see Wheway (1999)].

Let the points in the training set be $\{(y_n, \mathbf{x}_n)\}$. Let $\text{pr}_k(n)$ be the unweighted proportion of times that the nth case in the training set is misclassified by the first k classifiers. To check on my hypothesis that AdaBoost worked so well because it is an equalizer, using the methods in Breiman (1999b), I devised an algorithm called arc-equal which sequentially minimizes $\text{Av}_n(\text{pr}_k(n) - t)^2$, where t is an adjustable parameter.

Then I did runs on a number of datasets, proceeding as follows: first AdaBoost was run 100 times. The t was taken equal to the average weighted misclassification rate over the training set produced by AdaBoost and arc-

equal run 100 times. When this was repeated 50 times, each time leaving out 10% of the data as a test set, the test-set error rate of arc-equal was always nearly the same as that of AdaBoost.

The experiments performed and the empirical observation that our datasets, AdaBoost tends to equalize the misclassification error over the training set is suggestive. But it does not yet explain why equalization leads to low test-set error. This mystery should be most interesting to some of our more theoretically motivated statisticians.

REFERENCES

AMIT, Y. and GEMAN, D. (1997). Shape quantization and recognition with randomized trees. *Neural Comput.* **9** 1545–1588.

BREIMAN, L. (1999a). Random forests. Technical report, available at www.stat.berkeley.edu.

BREIMAN, L. (1999b). Prediction games and arcing algorithms. *Neural Comput.* **11** 1493–1517.

DIETTERICH, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Mach. Learning* **40** 139–158.

WHEWAY, V. (1999). Variance reduction trends on 'boosted' classifiers. Available from virg@cse.unsw.edu.au

DEPARTMENT OF STATISTICS
UNIVERSITY OF CALIFORNIA
367 EVANS HALL, #3860
BERKELEY, CALIFORNIA 94720-3860
E-MAIL: leo@stat.berkeley.edu

DISCUSSION

PETER BÜHLMANN AND BIN YU

ETH Zürich and Bell Laboratories, Murray Hill, Lucent Technologies and University of California, Berkeley

We congratulate Friedman, Hastie and Tibshirani (FHT) for their significant work connecting boosting and (kinds of) additive models. FHT provide a much-needed bridge between an important and effective machine learning procedure and traditional statistical modeling ideas. With this bridge, ideas can now flow easily in both directions so that a deeper and more thorough understanding of boosting will eventually emerge.

In this discussion, we would like to share our thoughts and reflections on boosting and related statistical ideas, inspired by FHT.

1. What makes boosting work in classification? Let us consider the two-class problem in this section. In an elegant and convincing way, FHT bring Freund and Schapire's Discrete AdaBoosting to our home domain by rederiving its population version as a familiar Newton-like step in the optimization of a not-so-familiar exponential loss function serving as a surrogate for the zero–one loss function. This surrogate is sensible since it is uniquely minimized by

(half of) the log odds ratio of the conditional probability of $Y = 1$ given $X = x$. Under this light, it is not hard to find instances of statistical procedures sharing traces of similarities with boosting. For example, in parametric estimation, we indeed have been "boosting" a consistent estimator to an efficient estimator by taking a Newton step in the optimization of the likelihood function and indeed have been using estimating equations as sensible surrogates because the likelihood equations are intractable. Despite these similarities, there are fundamental differences between boosting and these statistical procedures. As recounted by FHT, boosting as a conjecture was proposed as a theoretical learning question in the machine learning community and is now being viewed as a nonparametric stepwise fitting technique in an additive style. Its superb empirical performance in high-dimensional classification problems has very few, if any, rivals. Even though the concept of a "true" model does get used in the evaluation stage of boosting, the starting point of boosting is not a "true model" as is commonly done in statistics. Its starting point is a "weak learner" and the question posed was how to make it better or "boost" it. In hindsight, this is a natural and realistic approach in modern data analysis where the size of datasets could not be imagined in the time of our forebears such as Fisher or Neyman. Because of the complexity and scale of these problems, it is impossible to come up with an effective likelihood approach in one step. Often, if not always, a sensible procedure or a "weak learner" is available, either ad hoc or based on a simple approximation. Moreover, an evaluation criterion or a loss function is always available. The boosting question now becomes how to improve the weak learner in terms of this loss function. With a starting point and an objective function, a greedy approach for improvement is very natural. Numerical considerations of the greedy approach explain why boosting in terms of the *evaluating* zero–one loss function might not be a good idea. Any derivative-based greedy algorithm such as Newton's method is not appropriate for this discontinuous loss function whose minimizer is not unique but a whole class. A surrogate, the exponential loss function, is used in AdaBoosting as the *implementing* objective function although the evaluation function remains the zero–one loss. From a numerical point of view, the exponential function is an excellent function to apply Newton's method, because of its convexity and gradually changing derivatives. Furthermore, this is a relevant loss function to optimize since its minimizer is sensible for the classification problem. A second surrogate, proposed by FHT, is the binomial likelihood that as a function is very close to the exponential loss and has the same minimizer. FHT devise the Gentle Boosting algorithm based on the binomial surrogate and show a similar performance to those based on the exponential loss. A third surrogate, the squared loss function, is also discussed in FHT. The squared function is the best possible for Newton's method: convex and having constant second derivatives. FHT report good performance but think that it is dominated by schemes based on either the exponential or the binomial likelihood loss functions and hint that the reason might be that the squared loss function loses its monotonicity when $yF(x) > 1$. It is well known that the unique minimizer of the squared loss is the conditional expectation of Y given $X = x$

which is naturally bounded between -1 and 1 since $|Y| = 1$. If this constraint is taken into account in the greedy search for the optimizer of the squared loss, then one should not wander out to the region where $yF(x) > 1$ but stay in the region where the squared loss is monotone. It is curious to know whether or not taking this constraint into account (obviously numerically more complicated) would bring the performance of squared loss boosting on a par with those based on the exponential or binomial likelihood loss.

Although the three surrogates mentioned so far have sensible minimizers, they are only qualitative approximations to the zero–one loss function. They are actually quite bad quantitative approximations (cf. Figure 2 of FHT). But the population minimizers of the expected zero–one loss function include those of exponential, binomial likelihood and the squared loss. The first question then simply becomes:

1. Which surrogate (or implementing loss function for boosting) to use so that the boosting *estimate* best approximates its corresponding population minimizer? This (difficult) question might involve numerical as well as statistical–stochastic efficiency issues.

Now let us step back from these successful (implementing) surrogates to look at the original zero–one loss function. From the quantitative approximation point of view, an obvious surrogate is a smoothed version of the zero–one loss function $1 - \Phi(yF(x)/\sigma)$ where Φ is the cdf of $\mathcal{N}(0, 1)$ and σ is a tuning parameter to control the approximation of such a smoothed version to the original zero–one loss function. This makes the numerical greedy approach possible. If one follows the exact steps (when c is fixed) to this objective function as in the derivation of the population version of Discrete AdaBoost (Result 1 in FHT), one gets a boosting algorithm with a *different reweighting* scheme having weights.

$$\varphi(yF(x)/\sigma) = \varphi(F(x)/\sigma),$$

because $y^2 = 1$ and $\varphi(\cdot)$, the standard normal density, is symmetric.

These weights ignore the values of y or avoid reality check and concentrate more and more on the cases where the previous classifier F is unsure about itself. Since this is the opposite of the boosting reweighting philosophy, one might expect that it would not work well. The smoothed zero–one loss function is not convex (or concave) and its derivatives change rapidly, especially for a small tuning parameter σ or a closer approximation to the zero–one loss function. For such functions, the quadratic approximation is not accurate and Newton's method easily overshoots to miss the true optimizer. Hence this recipe should not work from a numerical point of view.

2. The question is, however, for this smoothed zero–one loss, will a more suitable numerical optimization method such as trust region [cf. Gill, Murray and Wright (1981)] lead to a sensible boosting procedure having a different weighting philosophy, presumably with higher computational cost?

As argued above, boosting algorithms benefit greatly from a smart choice of a surrogate implementing objective function from both statistical and numerical points of view. FHT (Section 4.4) also point out that Gentle Boosting has an edge because of the numerical stability of the derivative calculation of the binomial likelihood function. However, boosting's most powerful advocate is its stunning success on real data sets and its "mysterious" resistance to overfitting in most cases in classification. The summary in Table 2 of FHT on real dataset results suggests that the performance of a particular variant of boosting depends on the interaction among the choice of the weak learner, the underlying problem and the effectiveness of the Newton method as a numerical approximation to the surrogate loss function. Figuring out how these factors interact should bring us a deeper understanding of boosting and might shed light on its resistance to overfitting.

We explore this interaction in the rest of this discussion. In the context of boosting L_2 regression, we compare boosting with another ensemble scheme, bagging [Breiman (1996)], for which we have gained some understanding recently [Bühlmann and Yu (2000)]. Different weak learners are looked at, the overfitting issue is touched upon, and a bag-boosting scheme is proposed. In the last section, we return to classification to emphasize the importance of the choice of the weak learner and to make the point that the resistance to overfitting of boosting is probably due to the zero–one evaluation loss function.

2. Boosting and bagging with L_2 loss for regression. Boosting as explained and analyzed by FHT is a general method: a greedy forward stagewise technique to fit a model of additive style by minimizing a loss function. This view opens the door for boosting in other contexts than classification (although L_2 loss is also an appropriate surrogate as described in the previous section). For L_2 regression, the boosting algorithm works as follows:

(a) Set $F_0 \equiv 0$ (or another more sensible starting value).

(b) For $m = 1, 2, \ldots, M$, fit the function estimator $f_m(\cdot)$ which is parameterized as $f_m(x) = \beta b(x, \gamma)$ (as in FHT):

$$\beta_m, \gamma_m = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} \big(Y_i - F_{m-1}(X_i) - \beta b(X_i; \gamma)\big)^2.$$

Set $F_m(\cdot) = F_{m-1}(\cdot) + f_m(\cdot)$.

(c) Output the function estimator

$$F_M(\cdot) = \sum_{m=1}^{M} f_m(\cdot).$$

This algorithm is indicated by FHT [formula (6)] and was also given by Friedman (1999). It does not involve any "reweighting". The weak learner $b(\cdot; \gamma)$ is fitted in the mth step to the current residuals $Y_i - F_{m-1}(X_i)$. There is no need for a surrogate loss function in this case since the evaluating L_2 loss is the best possible for Newton's method and the quadratic approximation is

exact. We now discuss the issue of choosing the weak learner. This discussion is continued in Section 3 for the case of classification.

Linear weak learners. It is well known that boosting's cousin bagging does not change any linear procedures. It can be shown easily that it is the same with boosting in L_2 regression. When the weak learner is linear $b(x, \gamma) = x^T\gamma$ and $f_m(x) = x^T\gamma_m$, the following two statements hold:

(a) For L_2 boosting in regression as described earlier,

$$f_m(\cdot) \equiv 0 \text{ for all } m = 2, 3, \ldots, M,$$

$$F_M(\cdot) = \text{least-squares linear regression predictor for all } M \geq 1.$$

(b) For LogitBoost as presented by FHT, with

$$p_M(x) = \exp(F_M(x))/\big(\exp(F_M(x)) + \exp(-F_M(x))\big),$$

$p_M(x)$ converges to the corresponding MLE in logistic linear regression as $M \to \infty$, provided that standard conditions for convergence of the MLE hold.

Stumps as the weak learner. We now investigate the effect of boosting in a concrete simple linear model,

$$Y_i = 2 + 3X_i + \varepsilon_i, \qquad i = 1, \ldots, n, \tag{1}$$

with ε_i's i.i.d. $\sim \mathcal{N}(0, 1)$, X_i's i.i.d. $\sim$ Uniform($[0, 1]$) and all X_i's independent from all ε_i's. The target is the true function $F_{\text{true}}(x) = 2 + 3x$, $x \in [0, 1]$.

We use stumps (with one-dimensional covariate space) as weak learners and run the L_2 boosting algorithm for regression from above. For a typical sample, the boosting stumps estimate is displayed in Figure 1; it is still a rather crude and erratic approximation for the target. Figure 1 uses six boosting iterations which is optimal on average with respect to MSE as indicated in Figure 2. Based on 50 independent simulations of model (1), we have estimated bias, variance and hence also mean-squared error as a function of the number of boosting iterations. The result is displayed in Figure 2.

3. Interestingly, there is a clear indication for overfitting, starting after six iterations already. We see here a simple example in L_2 boosting for regression where overfitting occurs easily in contrast to classification; similar phenomena are reported in Friedman (1999).

From our own work [Bühlmann and Yu (2000)] we know that stumps evaluated at x have high variances for x in a whole region of the covariate space. From an asymptotic point of view, this region is "centered around" the true optimal split point for a stump and has "substantial" size $O(n^{-1/3})$. That is, stumps do have high variances even in low dimensions as in this simple case (with only three parameters) as long as one is looking at the "right scale" $O(n^{-1/3})$; such a high variance presumably propagates when combining stumps in boosting. This observation is the starting point for another boosting machine to be described next.

The bag-boosting machine with bagged weak learners. Bagging is known to smooth the weak learner such as stumps and thus achieves a variance reduction. The precise (asymptotic) smoothing function is given in Bühlmann and Yu (2000), which also characterizes its variance reduction effect in simple yet canonical cases. The high variance of a stump is reduced up to a factor of size 2 to 3, while leaving the bias approximately unchanged. This implies that bagging is also effective for a low-dimensional predictor such as stumps. Hence, we combine bagging with boosting and we call it bag-boosting. For the weak learner in boosting, just replace the stump (or a more general tree) by the bagged stump (or bagged tree). This is a very natural idea and has been thought about by a couple of researchers although we have not seen anything in writing. The resulted fit from bag-boosting is shown in Figures 1 and 2. Note that performance for bagging alone is given by bag-boosting with one boost. Instead of a classical bagging step, we actually use "sub-bagging" or bagging on subsamples of size $[n/2]$ (resampling without replacement) which is computationally cheaper while still being as accurate as bagging [Bühlmann and Yu (2000)]. The visual improvement of bag-boosting in Figure 1 can be

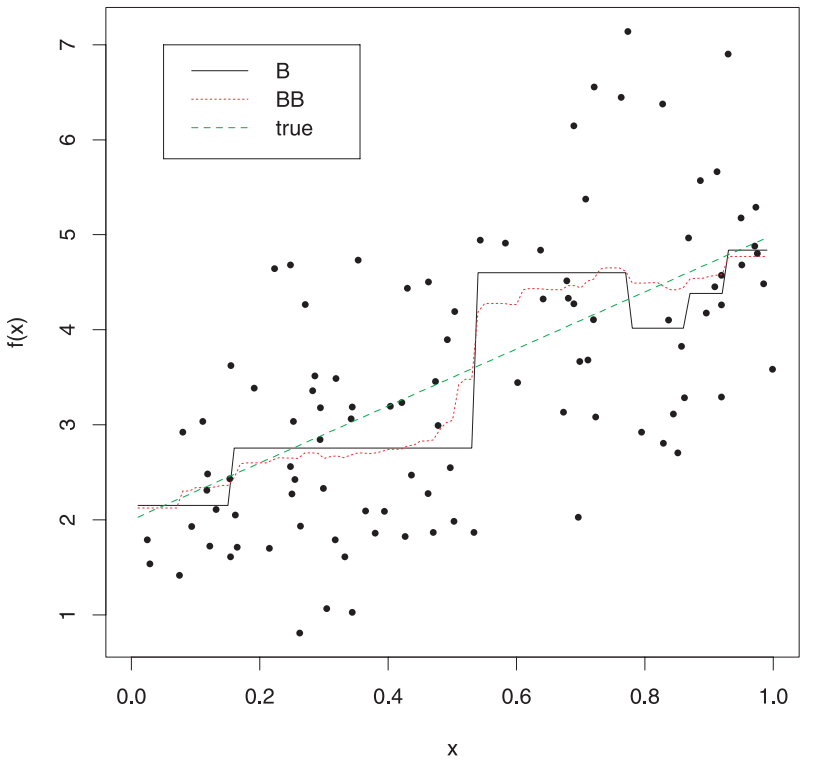

FIG. 1. *Boosting stumps* (B, *black*) *and bag-boosting stumps* (BB, *red*) *estimate based on a typical sample* (*dots*) *from* (1). *The target function is indicated in green. Boosting is done with* 6 *and bag-boosting with* 4 *iterations.*

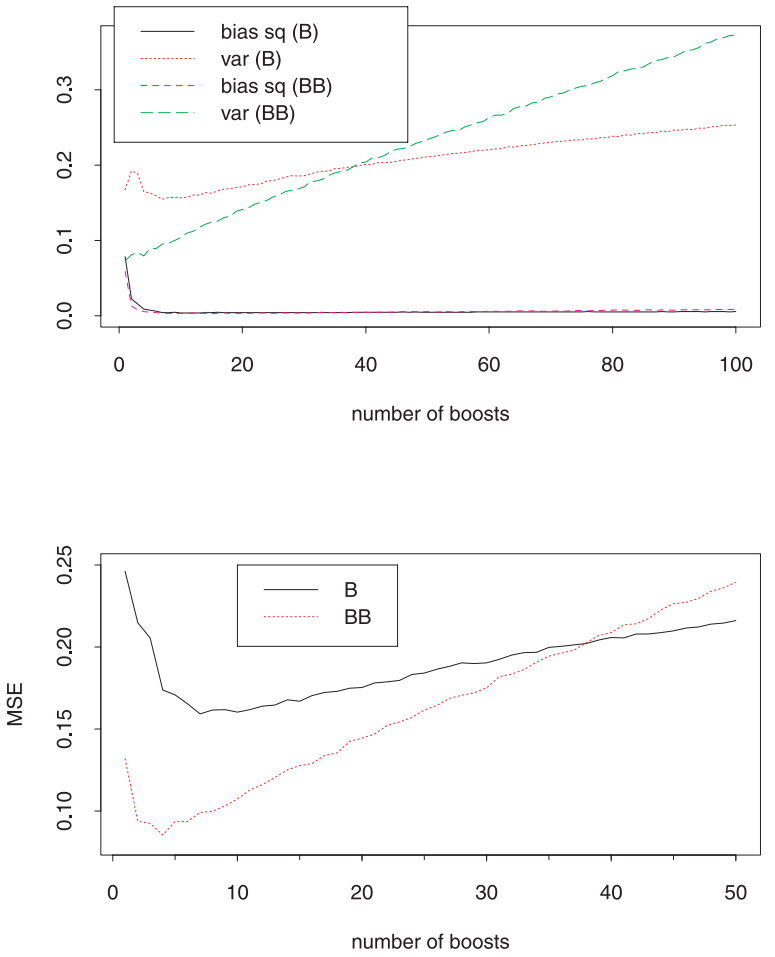

FIG. 2. *Boosting stumps and bag-boosting stumps for model* (1). *Top: squared bias and variance of boosting* (B, *black and red*) *and bag-boosting* (BB, *purple and green*). *Bottom*: *mean-squared error for boosting* (B, *black*) *and bag-boosting* (BB, *red*).

explained by the fact that a bagged stump is a smooth rather than a step function [Bühlmann and Yu (2000)]. Its variance and MSE improvements are impressively described by Figure 2, provided that we know roughly where to stop with boosting iterations. Figure 2 illustrates that the more efficient base learner (namely the bagged stump) has a faster increase in variance with the number of boosting iterations: It would be interesting to know whether this is a more general fact.

4. Thus bag-boosting has a potential to improve upon boosting with stumps or larger decision trees. Its drawback is the higher, although still feasible, computational cost.

3. Back to classification. Many issues remain. For example, the question about convergence is not touched by FHT. In particular, for a fixed sample size, there are two issues: the (non)convergence of the boosting algorithm and whether it is better to stop iterations before having approximately converged, if the algorithm actually would converge. The latter issue is known in fitting complex parametric models such as neural networks where a possible regularization is given by stopping before convergence. There is another convergence issue about the next level when the sample size tends to infinity.

5. A more important question is how much FHT's population story tells about the data driven algorithms. The population quantities are always approximated: the accuracy depends on the choice of weak learners (see also our remarks about linear weak learners, stumps and bagged stumps in Section 2) and the data-generating distributions. Finally, the hope is that the errors do not propagate with the boosting iterations!

Particularly, even with the connection made by FHT, it is hard to know whether one should use a simple learner like stumps or a more complex one such as trees with more terminal nodes.

In the boosting literature, which is almost exclusively on classification, decision trees are the most popular weak learners. We do agree that trees are generally attractive, but it is worth pointing out that other (nonlinear) weak learners may exhibit good performance as well. Even when using trees as weak learners, the choice of the number of terminal nodes (e.g., with best-first induction) in the learners does matter. Figure 1 in FHT can be misleading for a superficial reader. It wrongly suggests that the number of terminal nodes in the learner is irrelevant when there is sufficient computing power for performing many boosting steps; and that with stumps, Discrete AdaBoost is outperformed by Real AdaBoost consistently. The careful reader will exploit a more subtle sensitivity with respect to choosing the learner in Section 6 of FHT [e.g., Figure 4 (top left and top right)] and results on real data sets (Table 2).

FHT distinguish between additive and nonadditive decision boundaries. They argue convincingly that in the first case, choosing stumps as weak learners is better than using a larger tree which is expected to overestimate nonexisting interaction terms. If we knew that the decision boundary could be well approximated by an additive function, we would feel more comfortable by fitting an additive model (with backfitting) as in Hastie and Tibshirani (1990), rather than boosting stumps. Or in other words, if the conjecture by FHT [Section 11] "that boundaries involving very high-order interactions will rarely be encountered in practice" (whatever "high" means here) is really relevant, there would not be any good reason to prefer boosting over the established and popular GAM backfitting estimate.

Boosting's resistance to overfitting has been one of its major virtues in comparison with other nonparametric techniques. Overfitting in boosting algorithms in the sense of fitting a large model can happen in two ways. The first is having a large model fitted as the weak learner and the other having to run

TABLE 1
Misclassification rates (in percentages) for the breast cancer data original tree (unpruned; the program from R): 5.643

Subbagging	4.812
Boosting stumps	4.029 (with optimal 97 boosts)
Bag-boosting stumps	3.652 (with optimal 47 boosts)
Boosting large tree	2.929 (with optimal 117 boosts)
Bag-boosting large tree	2.768 (with optimal 11 boosts)

the algorithm for many iterations. The most overfitted model comes from a large or complex weak learner in combination with a long iteration in boosting. FHT touch on this overfitting issue in the concluding section by listing three plausible explanations. As emphasized in Section 1 of this Discussion, the use of the (implementing) surrogate exponential loss function is crucial from the numerical point of view and is thus partially responsible for the success of boosting. However, the evaluation in a classification problem has always been the zero–one loss.

6. This divorce of the implementation and the evaluation loss function does not exist in L_2 boosting where we saw overfitting easily in Section 2. We concur with the last point of FHT in Section 11 that the zero–one loss is very robust against overfitting and we further argue that an evaluation based on the exponential implementing loss does show strong overfitting. This conjecture is supported by the breast cancer dataset below. Moreover, we believe that "parts of the real world" are nonadditive. Under this belief and knowing boosting's resistance to overfitting, it can be very valuable to use best-first inducted trees with *more than two* terminal nodes. This is confirmed also by our analysis of the breast cancer data in Table 1.

Analysis of breast cancer data. We partially redo the analysis of FHT for the breast cancer data and thereby add insights on bag-boosting and overfitting. Table 1 and Figure 3 give comparative results to the ones in FHT. We use "sub-bagging" for computational efficiency and it has a performance similar to bagging [Bühlmann and Yu (2000)]. The misclassification rates (called test error in FHT) are estimated by averaging over 100 random partitions of the data set into a 90% training and a 10% testing, while FHT use a 5-fold CV. So our results differ slightly from FHT (also because presumably we are using different tree algorithms), but we believe ours have better accuracies. Boosting and its variants are done with Real AdaBoost, and the trees are run with the program from R. Bag-boosting large trees gives the best result, followed by boosting large trees, bag-boosting stumps, boosting stumps and then bagging. Bag-boosting only needs few iterations and brings nontrivial improvement over boosting alone, at the expense of a more computationally costly weak learner. Note that Figure 3 (bottom panel) shows strong overfitting with respect to $\mathbf{E}[\exp(-YF(X))]$!

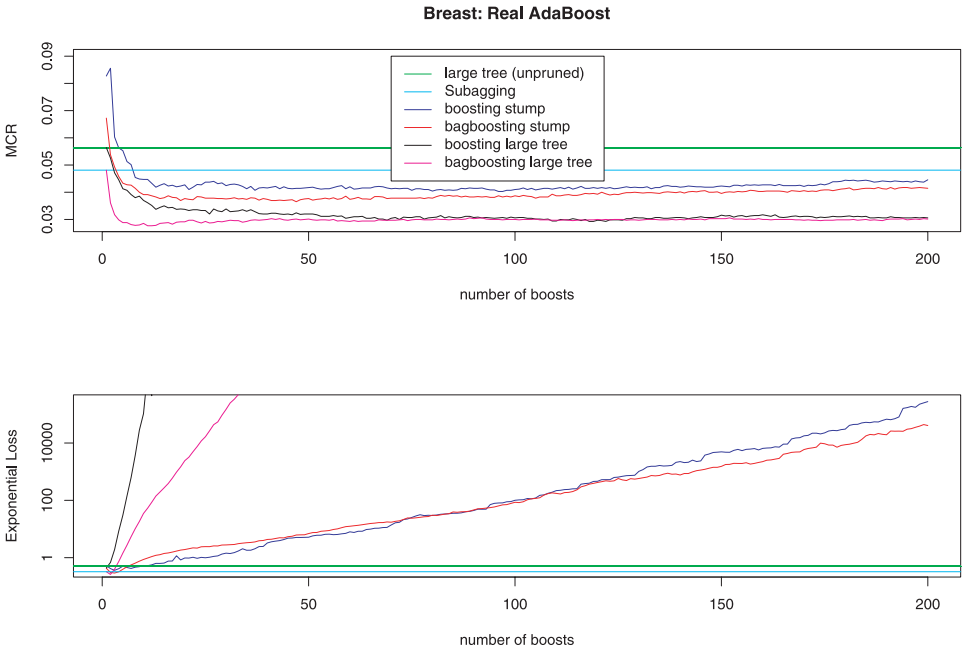

FIG. 3. *Different weak learners in Real AdaBoost for breast cancer data* (*Section* 7 *in FHT*). *Top*: *misclassification rates. Bottom*: *expected exponential losses* $\mathbf{E}[\exp(-YF(X))]$ *on the logarithmic scale.*

7. For this particular dataset and with respect to the misclassification rate, there is hardly any overfitting. It suggests that there is no loss in using large trees, if there is never any substantial overfitting. Boosting stumps on the other hand restricts to additive models. With respect to the misclassification rate, should we always boost with large trees as weak learners?

Acknowledgments. Bin Yu thanks Mark Hansen for many stimulating discussions and helpful comments on the draft and Margaret Wright for helpful discussions on optimization. Special thanks are due to Josef Vogt for the results on the breast cancer data.

REFERENCES

BREIMAN, L. (1996). Bagging predictors. *Machine Learning* **24** 123–140.
BÜHLMANN, P. and YU, B. (2000). Explaining bagging. Preprint.
FRIEDMAN, J. (1999). Greedy function approximation: the gradient boosting machine. Technical report, Stanford Univ.
GILL, E. P., MURRAY, W. and WRIGHT, M. H. (1981). *Practical Optimization.* Academic Press, New York.
HASTIE, T. and TIBSHIRANI, R. (1990). *Generalized Additive Models*. Chapman and Hall, London.

SEMINAR FÜR STATISTIK
ETH-ZENTRUM, LEO D72
CH-8092 ZURICH
SWITZERLAND
E-MAIL buhlmann@stat.math.ethz.ch

DEPARTMENT OF STATISTICS
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720-3860

DISCUSSION

ANDREAS BUJA

AT&T Labs

1. The cultural background of boosting: machine learning versus statistics. I have worked under the same roof with the inventors of boosting, Freund and Schapire, for several years, but I might as well have been on another planet. The physical vicinity was of no help in coming to grips with boosting. I listened to their talks, but I heard the words and missed the point. Later I read Leo Breiman's debates with Freund and Schapire, and it really got my attention when I heard his praise of boosting. His penetrating analysis was an effort to understand the performance of boosting, in particular its mysterious immunity to overfitting. What he did not provide was a translation of boosting into terms that are familiar to statisticians. He did not bridge the culture gap that separates the stodgy main stream of statistics from its dynamic younger sibling in computer science, machine learning. This is one of the achievements of the paper by Friedman, Hastie and Tibshirani (FHT): It gives statisticians a language with which to think about boosting.

If Freund and Schapire feel misinterpreted or if they think of FHT's statistical modeling interpretation as misguided, as I expect they will, it is beside the point. 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, and this is what the FHT paper certainly does for statisticians and the idea of boosting.

An interpretation of new ideas in terms of old ones is not a mere reduction of the ideas to an "old hat." On the contrary, such a reduction can clarify what is novel. The emphasis of the FHT interpretations may be too long on the reduction aspect and too short on working out the novelty in boosting; I will try to explain what I mean below. The novelty that we see in light of FHT's interpretations, though, may not agree with what Freund and Schapire think is novel. Thus, while the FHT paper may provide a gate for statisticians to a new area, the same gate may not be open in reverse for the folks in machine learning (although I'd love to be proved wrong).

2. Boosting arbitrary generalized linear and additive models. According to FHT, boosting can be seen as a two-class classification method that fits additive models on a logistic scale in a forward stagewise manner with regard to a loss function that is second-order equivalent to the negative log likelihood for the conditional Bernoulli model. This interpretation leads FHT naturally to propose their "LogitBoost" method by substituting the exact negative log likelihood for the original loss function.

The test of every interpretation is in its capacity to spawn new ideas and generalizations. Differing interpretations are worthwhile because they lead to differing generalizations. The FHT interpretation resulted immediately in a

very competitive new version of boosting, as well as an improvement in boosting multiclass classification.

It seems to me, though, that the FHT interpretation calls, even screams, for a much more global extension of boosting: Now that we know how to boost additive logistic regression, we also know how to boost any generalized additive model. If Algorithm 3 of FHT for LogitBoost is the familiar Newton–Raphson algorithm modified for the acquisition of nonlinear terms in a stagewise logistic regression, then the same can be used for the acquisition of nonlinear terms in any stagewise generalized regression.

In particular, we now know how to boost ordinary least-squares regression: by stagewise LS fitting of nonlinear terms to the previous residuals. This may seem trite, but it isn't if one has ever tried to come up with a version of boosting for regression with continuous response. Freund and Schapire have tried, and their tentative proposal was to reduce regression to some form of classification, in a classical attempt at reducing a new problem to an old one. This seems like a circuitous approach, however, once the FHT interpretation of boosting is at hand. With the extension of the FHT interpretation to generalized additive regression, it seems most natural to interpret stagewise LS regression as the regression version of boosting.

3. The role of greedy stagewise fitting in the protection against overfit. FHT's interpretation of boosting is lucid, but their presentation of stagewise fitting needs to be put in perspective. I have no quibbles with the facts, but some with the positioning of the facts.

FHT do state that the fitting à la boosting is "forward stagewise," but the impression we (and above all the readers in machine learning) may get is that this term describes a common mode of model fitting in statistics. It doesn't. We have the term, but it is not very well known, and it does not enjoy high standing among most statisticians who know about it. It is a computational device, or more accurately, a capitulation, in those situations where one is unable to fit a large number of model terms jointly. We even teach our students that fitting one term at a time and not adjusting the others is *not* the way to fit a model; we teach them that changing one term in a model affects all others in a joint model fit, assuming that the joint model fit is the norm. Interestingly, one popular method for joint fitting, the backfitting algorithm, was devised by Friedman and Stuetzle (1981) to more closely achieve joint optimality in the otherwise greedy fitting procedure of projection pursuit regression (PPR). Had they omitted the backfitting part in their algorithm and allowed the acquisition of a greater number of ridge function terms instead, PPR would have been a regression version of boosting. It appears that even the unconventional minds of Friedman and Stuetzle were involuntarily constrained by statisticians' prejudice in favor of parsimonious models obtained by joint model fits. Doing away with this prejudice is certainly one of the novelties in the boosting endeavor. Ockham's razor is being replaced by Freund and Schapire's sledgehammer.

Omitting joint fitting is crucial to the success of boosting: Its purported advantage, relative protection against overfit, disappears instantaneously if termwise (stagewise) fitting is replaced with joint fitting of all terms. The 500 or so stumps and twigs that I have seen fitted in a single boosting fit would come crashing down if they were fitted jointly. Therefore, the essence of the protection against overfit is in the suboptimality of the fitting method.

This also explains why boosting could not have been invented by statisticians: Our preoccupations are such that the statistical benefits of a suboptimal computational device would not have appeared on our radar. There are exceptions to the rule, though: Statisticians have, for example, developed a notion of one-step estimates in robustness, where it was shown that one Newton step in the computation of an M-estimate provides essentially the robustness benefits of full iteration. Similarly, in order to understand the performance of boosting, we will need a theory that explains why this particular type of suboptimality often avoids overfit.

A curious observation in this context concerns the cavalier attitude in the boosting community, including FHT, toward the stopping problem. They seem to consider it sufficient to provide plots or tables that show misclassification rates for 100, 200, 300, 400 terms. So we stop around 300 or 400, and this is part of the definition of the method. The assumption, empirically substantiated, is that it really doesn't matter much as long as we stop after the acquisition of a sufficiently large number of terms. The situation is similar to that in artificial neural networks (NNs), where the stopping time of online optimization acts as a smoothing parameter. In NNs, however, one tries to detect the onset of overfit with cross-validation. In boosting, by contrast, the testing error remains flat for a long time, overfit does not occur before the human runs out of patience and a precise stopping rule seems therefore unnecessary.

A satisfactory theory of protection against overfit would have to explain the relative flatness of the test error in boosting. The flatness is only relative, however, and even boosting has to submit to overfit at some point. This is seen from the fact that usually the class of fitted terms $b(x, \gamma)$ is rich in the sense that for a given finite sample size N the N-dimensional design vectors $\mathbf{b}(\gamma) = (b(x_n, \gamma))_{n=1\cdots N}$ span all of R^N as γ runs across its domain.

This is obviously not a proof, but a slightly tighter argument can be given. First, we note that, unlike in joint fitting, any iterative termwise fitting procedure depends not only on the space spanned by the terms, but the actual basis they represent and the order in which they are fitted. (I thank Werner Stuetzle who spelled this out most clearly to me.)

In the simple case where the domain of the γ's is finite and consists of K elements $\gamma_1, \ldots, \gamma_K$ only, boosting is a greedy form of backfitting the *linear* model $\sum_{k=1\cdots K} \beta_k b(x, \gamma_k)$. Instead of cycling through the terms in an orderly fashion $k = 1, 2, \ldots, K, 1, 2, \ldots$ as in conventional backfitting, the next k is picked greedily so as to lower the loss function maximally. We can conceive of at least three types of backfitting according to the order in which the terms are fitted: (1) cycle over the terms in a fixed order (traditional backfitting), (2) pick the next term randomly, (3) pick the next term greedily. It would

seem plausible to examine combinations of type 2 and type 3, whereby the picking probability of a term is a function of its loss reduction; as is, type 2 uses a uniform distribution on the terms and type 3 puts probability 1 on the term with maximal reduction.

It is clear from the known convergence of type 1 backfitting to the joint least-squares solution that overfit must occur if the joint solution overfits. If similar convergence properties for type 2 and type 3 backfitting hold, overfit will occur there also. In light of these considerations, it may be just a matter of time till boosting follows the lead of NNs and cross-validation is introduced for the stopping problem in boosting as well.

An issue that a theory of protection against overfit would have to tackle is that there are two ways for a learner or base term $b(x, \gamma)$ to be "weak": weakness can arise from bias *or* from variance. In either case the weak learner usually contains signal, but the signal is weak for different reasons. Weakness due to bias means underfit, weakness due to variance means overfit. Boosting is designed to reduce bias but not variance. The latter is addressed by Breiman's (1996) "bagging" proposal and by Bayesian model averaging. As things stand, one either approaches good fits from poor fits with boosting, or from overly flexible fits with model averaging. It would be desirable if both bias and variance could be tackled in one unified approach.

4. The role of the functional form of the weak learner. Additive models with small trees as base terms are an innovation we owe the machine learning literature. Trees are not essential to boosting, but the idea of building additive models from multiple trees is a welcome side benefit.

Functional form hasn't been a topic of great debate in the boosting literature, probably because of the original justifications in terms of the "weak learner" metaphor. These justifications do not carry us far, however. The question is not *whether* but *how much* a weak learner can be boosted, or in statistical terminology: how well an additive model can approximate an underlying response surface. A major determinant for answering this second question is the functional form of the weak learner $b(x, \gamma)$. The fact that the boosting folks haven't hit on this the hard way is probably due to the Holte (1993) effect, cited a couple of times by FHT: "Very simple classification rules perform well on most commonly used datasets." That is, nature rarely chooses complex decision boundaries that couldn't be described by thresholding additive functions or at most two-variable or three-variable interactions.

Inspired by ANOVA practices, FHT propose to measure and control complexity of a functional form in terms of interaction order, that is, the maximum number of variables admitted in the weak learner at any given time. For trees, the interaction order is identical with the maximum admitted tree depth. But the idea of interaction order can be extended to other functional forms as well. The most obvious are Friedman's MARS fits, in which interaction order can be controlled as easily as in trees. Another possibility are projection pursuit regression and classification (PPR and PPC, respectively), although a major modification is necessary: the linear combinations that enter the ridge

functions would have to be constrained to, say, two or three predictors at a time.

In order to more fully understand boosting, it will be necessary to apply it to functional forms other than trees. This would not be difficult in the case of MARS and PPR/C. In fact, for a boosting version of either one would simply replace the joint fitting with unadjusted greedy fitting. Ironically, stagewise modifications of MARS and PPR/C require that the technically most creative ideas of their inventors be thrown out: Friedman used sophisticated stable Cholesky updates for the fast computation of joint fits in MARS, and Friedman and Stuetzle devised backfitting for the same purpose in PPR/C. Now we are led to expect that giving up these technical marvels will be beneficial! Having experienced first hand the problems of overfit in the original MARS, I would be thrilled if the anticipated benefits of boosting showed up.

5. Conclusion. The FHT interpretation of boosting has implications for machine learning as well as for statistics. For machine learning, the generalization of FHT's ideas lead us straight to a regression version of boosting. For statistics, FHT's ideas raise the expectation that the benefits of boosting might carry over to other stagewise modifications of nonparametric fitting procedures. The latter is just a guess so far, and in the absence of a convincing theory of boosting's protection against overfit, we wouldn't even know why it held if it did. Clearly, there is a lot of work ahead of us. FHT have given us a means for generating new ideas that can guide our future research.

REFERENCES

BREIMANN, L. (1996). Bagging predictors. *Machine Learning* **24** 123–140.

FRIEDMAN, J. and STUETZLE, W. (1981). Projection pursuit regression. *J. Amer. Statist. Assoc.* **76** 817–823.

HOLTE, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning* **11** 63–90.

AT&T LABS
SHANNON LABORATORY
180 PARK AVENUE
FLORHAM PARK, NEW JERSEY 07932-0971
E-MAIL: andreas@research.att.com

DISCUSSION

YOAV FREUND AND ROBERT E. SCHAPIRE

AT&T Labs

The main and important contribution of this paper is in establishing a connection between boosting, a newcomer to the statistics scene, and additive models.

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. As pointed out by the authors, the current paper does not address this issue. Leo Breiman (1998) tried to explain this behaviour in terms of bias and variance. In our paper with Bartlett and Lee (1997), we gave an explanation in terms of the "margins" of the training examples and the VC-dimension of the base class. Breiman, as well as the current paper, point out that our bounds are very rough and are thus not useful in practice. While this is clearly true at this time, it is also true that the analysis given by Breiman and by this paper yield no provable bounds whatsoever. It is completely unclear whether this analysis can be used to predict the performance of classification rules outside of the training sample.

At the root of this argument about boosting is a much more fundamental argument about the type of prior assumptions that one should make when embarking on the task of inducing a classification rule from data. The assumption that seems to underlie the use of maximum likelihood in the current paper is that data are generated by a distribution from a *prespecified class*. In this case, this is the class of distributions in which the relationship between the features and the labels is described by a log-linear function. In comparison, the assumption that we make in our analysis is that the data are generated from some *arbitrary* distribution in an i.i.d. fashion. Clearly, our assumption is the weaker one and this leads to a theory that is more generally applicable.

From a related but more practical point of view, one main issue when applying boosting or boosting-like techniques in practice is how to choose the base class. The approach taken in this paper is that this choice is made based on our prior beliefs regarding the type of log-linear dependencies that might exist between the features and the label. On the other hand, in the boosting approach, we make an assumption about what kind of rules might have slight but significant correlations with the label. This is the essence of the "weak learning" assumption upon which the theory of boosting is founded.

In the current paper, boosting is analyzed mostly in the context of decision stumps and decision trees. The argument seems to be that while in most real-world cases decision stumps are powerful enough, in some less common cases the type of dependencies that exist in the data require a more powerful base class, such as two- or three-level decision trees. A rather different approach to the combination of decision trees and boosting was recently proposed by Freund and Mason (1999). They represent decision trees as sums of very simple functions and use boosting to simultaneously learn both the decision rules and the way to average them.

Another important issue discussed in this paper is the performance of boosting methods on data which are generated by classes that have a significant overlap, in other words, classification problems in which even the Bayes optimal prediction rule has a significant error. It has been observed by several authors, including those of the current paper, that AdaBoost is not an optimal method in this case. The problem seems to be that AdaBoost overemphasizes the atypical examples which eventually results in inferior rules. In the current

paper, the authors suggest "GentleBoost" as a better method than AdaBoost for this case. The reason that this might be a better method is that it gives less emphasis to misclassified examples. The increase in the weight of the example is quadratic in the negative margin, rather than exponential.

However, one can argue that this alteration of AdaBoost, while being a step in the right direction, is not large enough. In fact, one can argue that once an example has a very large negative margin it is best to assume that it is an outlier that should be completely removed from the training set. A new boosting algorithm based on this radical approach was recently proposed by Freund (1999).

REFERENCES

BREIMAN, L. (1998). Arcing classifiers. *Ann. Statist.* **26** 801–849.

FREUND, Y. (1999). An adaptive version of the boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*.

FREUND, Y. AND MASON, L. (1999). The alternating decision tree learning algorithm. In *Machine Learning: Proceedings of the Sixteenth International Conference* 124–133.

SCHAPIRE, R. E., FREUND, Y., BARTLETT, P. and LEE, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Ann. Statist.* **26** 1651–1686.

AT&T LABS–RESEARCH
180 PARK AVENUE
FLORHAM PARK, NEW JERSEY 07932-0971
E-MAIL: yoav@research.att.com

DISCUSSION

GREG RIDGEWAY

University of Washington

The authors have done a great service to the field of statistics by branching out to the problems of interests to the computational learning theorist. The connection between boosting and maximum likelihood not only helps to explain why boosting is an effective classification method but it also opens the door to the application of boosting to many other statistical prediction problems.

It is difficult not to understate the attention that boosting has received in the machine learning and data mining communities. Soon after Freund and Schapire's algorithm made its debut, a flurry of empirical work showcased its uncanny ability to improve prediction accuracy [Quinlan (1996), Bauer and Kohavi (1999), Drucker and Cortes (1996)]. Charles Elkan's application of the boosted naïve Bayes won first place out of 45 entries in the 1997 data mining competition [Elkan (1997)], demonstrating the strength of boosting against research and industrial prediction tools. Elkan also noted in this work that his model had the form of nonlinear logistic regression but did not completely

connect boosting with optimization and likelihood. Boosting is even making its way into popular science as noted in the *New York Times* [Lee (1999)].

Low variance, bias reducing steps. Likelihood based inference is in the business of obtaining good parameter estimates, which is not necessarily the same as producing a good predictor. In the classification game we win if we produce a classifier with the lowest misclassification rate. A good estimator of the underlying class distribution conditional on x is of secondary importance. Consider the following example.

EXAMPLE 1. Generate $N = 1000$ observations as

$$x \sim N(0,1), \qquad F(x) = -\frac{1}{2}x^2 - 1,$$

$$p(x) = \frac{\exp(2F(x))}{1+\exp(2F(x))}, \qquad y|x \sim \text{Bern}(p(x))$$

The Bayes decision rule is to classify a new observation as a 1 if $|x| > \sqrt{2}$ and 0 otherwise. The misclassification error for this rule is about 20.2%. Figure 1 shows the $p(x)$ that generated the data as a smooth curve. After 100

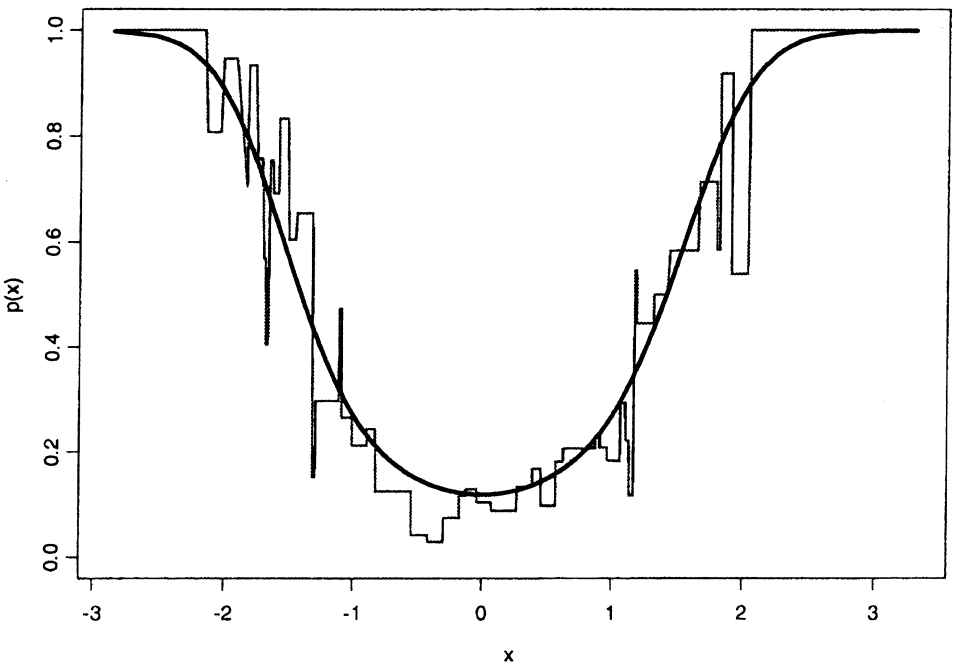

FIG. 1. *The smooth curve is* $P(Y=1|x)$ *and the jagged curve is the estimate that the Real AdaBoost algorithm produces after* 100 *iterations using a stump classifier on the Example* 1 *data.*

boosting iterations using a stump classifier (a piecewise constant with only one knot fit using CART) we see that boosting produces an estimate of $p(x)$ that is overfit in terms of providing an accurate picture of the underlying data generating mechanism. However, the misclassification rate on a test dataset with 10,000 observations is only 21.5%, not far off from the Bayes error rate. More iterations have little effect. This counts as a win for the analyst interested solely in predictive performance but a loss for those seeking to uncover the data generating mechanism. Even though the authors' Lemma 1 shows that the population minimizer of $\mathrm{E}e^{-yF(x)}$ is the true conditional class probability, with a finite sample it is easy for AdaBoost to overfit from a function estimate or calibration point of view. The debate between accurate prediction and the discovery of the contents of the black box is certainly not new but it is also has not gone away.

Some of the amazement surrounding boosting was that it could take a simple classifier like a stump and turn it into a near-Bayes optimal classifier. The choice of using a stump at each iteration in the example is critical. After two boosting iterations the classifier will pretty much nail the correct decision boundary since the two knots will likely be in the neighborhood of $-\sqrt{2}$ and $+\sqrt{2}$. Due to the constraints on the base classifier, subsequent iterations can do little to modify the decision boundary. Therefore, local jaggedness develops as successive stumps try to fit individual observations but it cannot be substantial enough to have an effect on the misclassification rate. This leads to the question of the importance of selecting the base classifier. In this example, fitting slightly more complex trees at each iteration has disastrous effects on the performance. Figure 2 tracks the change in misclassification rate with each boosting iteration when the base classifier is a tree with four terminal nodes. A four-terminal node tree is likely to nearly capture the decision boundary on the first iteration. Subsequent iterations have enough flexibility to fit individual observations and generalization performance declines. From Figure 2 we can see that the performance degrades to slightly worse than predicting 0, the majority class, for all observations.

The results of Example 1 yield the following insights into boosting.

1. AdaBoost (in its present form) is not necessarily useful in function estimation. Calibration of the boosted probability estimates is questionable.
2. The idea that AdaBoost is resistant to overfitting needs some qualification. The choice of base classifier can make or break the success of utilizing boosting.
3. Boosting seems to work well when we use a sequence of simple, low variance classifiers. Each stage makes a small, low variance step, sequentially chipping away at the bias.

Other looks at functional optimization. Once boosting is cast as functional optimization we can approach the problem from other avenues. In the AdaBoost setup we wish to find an $F(x)$ that minimizes $J(F) = \mathrm{E}e^{-yF(x)}$. An additive improvement to $F(x)$ would find an α and an $f(x)$ such that

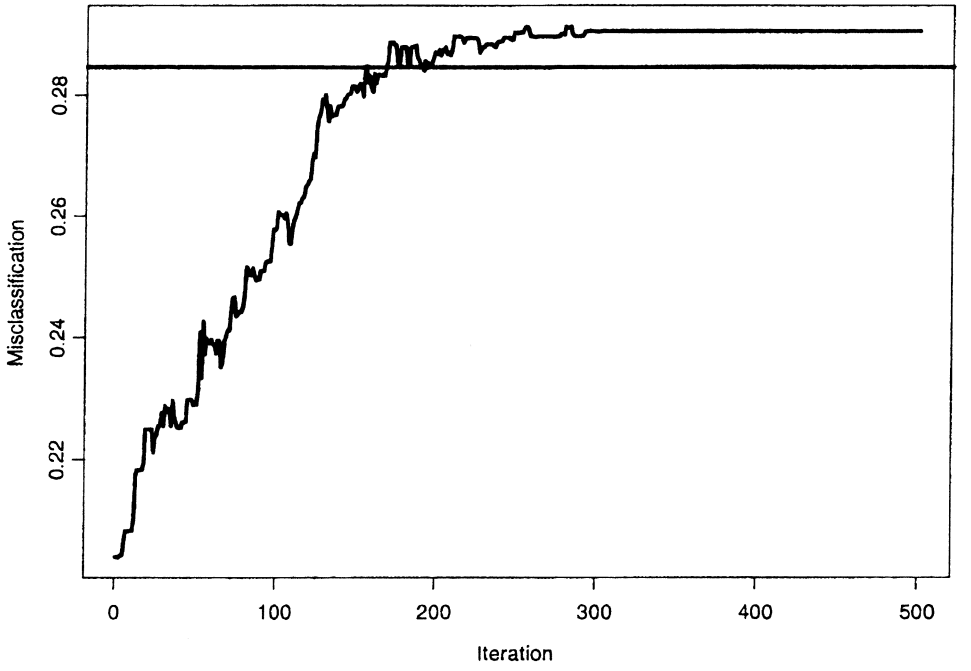

FIG. 2. *Misclassification rate when fitting trees with four terminal nodes to the Example* 1 *test data on each Real AdaBoost iteration. The horizontal line marks the error rate if we predicted all observations to be zeros.*

$J(F+\alpha f)$ decreases. The authors describe how AdaBoost selects α and $f(x)$. The Gentle AdaBoost algorithm that the authors propose performs a pointwise Newton step to suggest $f(x)$ with α fixed at 1. We can also view this as a gradient ascent in function space with a squared penalty on the magnitude of $f(x)$. Consider the following analysis.

The directional derivative, or Gâteaux variation, of $J(F)$ in the direction of some function $f(x)$ computes the local change in $J(F)$ if we make the modification $F(x) \leftarrow F(x)+\alpha f(x)$, moving $F(x)$ in the direction of $f(x)$. The directional derivative of $J(F)$ is

$$\begin{aligned}\delta J(F,f) &= \frac{d}{d\alpha}\mathrm{E}\,\exp(-y(F(x)+\alpha f(x)))\big|_{\alpha=0} \\ &= \mathrm{E} - yf(x)\exp(-yF(x)).\end{aligned} \tag{1}$$

A gradient ascent strategy would have us select from among some class of functions the $f(x)$ that minimizes $\delta J(F,f)$, or equivalently offers the greatest local decrease in $J(F)$. Without further restrictions this method is rather unstable since we can make $\delta J(F,f)$ arbitrarily small by setting

$$f(x) = \begin{cases} -\infty, & \text{if } \mathrm{E}\,y|x < 0, \\ \infty, & \text{if } \mathrm{E}\,y|x > 0. \end{cases} \tag{2}$$

On the other hand, if we penalize $f(x)$ in (1) for being too big then we might select $f(x)$ as

$$
\begin{aligned}
f(x) &= \arg\min_f \mathrm{E} - yf(x)e^{-yF(x)} + \tfrac{1}{2}f(x)^2 \\
&= \arg\min_f \mathrm{E} - yf(x)e^{-yF(x)} + \tfrac{1}{2}f(x)^2 + \tfrac{1}{2}(ye^{-yF(x)})^2 \\
&= \arg\min_f \mathrm{E}(f(x) - ye^{-yF(x)})^2.
\end{aligned} \tag{3}
$$

Note that (3) is equivalent to choosing the $f(x)$ that is similar to a least-squares fit to the gradient that the authors compute in their derivation of the Gentle AdaBoost algorithm. Obviously, in practice we only have a sample with which to estimate the expectation in (3). Therefore, it becomes critical that we use a low variance least-squares regression method to minimize the empirical estimate of (3). I will return to this topic in a moment. After estimating $f(x)$ we can use a line search to obtain the α that minimizes $\mathrm{E}\ \exp(-y(F(x)+\alpha f(x)))$.

Applying this same technique within the LogitBoost scenario, that is, maximizing the directional derivative of the Bernoulli log-likelihood with a squared penalty on the direction $f(x)$, the gradient step is

$$
f(x) = \arg\min_f \mathrm{E}\left(f(x) - \left(y^* - \frac{e^{F(x)}}{e^{-F(x)} + e^{F(x)}}\right)\right)^2. \tag{4}
$$

After estimating the optimal step direction the choice of the optimal step size, α, takes on an interesting form when we have a sample.

$$
\alpha = \arg\min_\alpha \sum_{i=1}^{N} 2y_i^*(F(x_i) + \alpha f(x_i)) - \log\bigl(1 + \exp\bigl(2(F(x_i) + \alpha f(x_i))\bigr)\bigr). \tag{5}
$$

Equation (5) has the form of a likelihood for the linear logistic regression model with $f(x_i)$ as a covariate and $F(x_i)$ as an offset term. The optimal α then is just the maximum likelihood estimate of the coefficient of $f(x_i)$ for which efficient software exists. Similar results hold for the class of exponential family regression models and the Cox model for survival analysis [Ridgeway (1999)].

Both the authors' results and this section arrive at the situation where we need to propose a direction in which to move our current guess for the regression function. When handed only finite training datasets, one is faced with estimating a direction that seems likely to offer a decrease in the loss function. Example 1 showed that high variance estimators of descent directions (e.g., four-terminal node trees) can cause problems. In the past statisticians have controlled the variance of predictors in other manners. As the authors point out, the LogitBoost algorithm has a familiar feel to statisticians. Traditional statistical practice uses linear regression instead of trees at each stage to estimate the conditional expectation. This yields the iteratively reweighted

least-squares algorithm for fitting linear logistic regression models. We can also use smoothers to estimate the conditional expectation as in the local scoring algorithm. These types of methods controlled variance at the expense of bias reduction by limiting the functional form, restricting variable interactions, or constraining the jaggedness of the curve.

When we introduce regression trees to estimate these gradient steps instability becomes an issue. However, various forms of averaging may help to keep variance in check. If we use bagged trees to estimate the gradient we can often reduce the variance introduced on the iteration. This is the basis of Breiman's adaptive bagging [Breiman (1999)]. Denison (2000) proposes averaging over all possible stumps at each iteration with each weighted by their posterior distribution. The amount of needed computation is nearly equivalent to CART with the benefit of more stable gradient steps. It seems that the future of nonparametric regression might consist of choosing an ideal loss function and a sequence of low variance estimates of the gradient direction in which each step decreases bias.

Bayesian prediction and function estimation. When I first learned of the problem of boosting I attempted to understand it from a Bayesian perspective. The redistribution of weights on observations, the model mixing and the improved performance seemed to have a Bayesian flavor to it. In fact, Freund and Schapire (1997) even offered a naïve Bayes justification for the AdaBoost algorithm. With the current results in hand it now seems clear that the Bayesian interpretation of boosting is that it is simply maximum likelihood! Boosting suggests moving the current function in some optimal direction and assembles a path through function space toward a maximum likelihood estimate. The Bayesian wants to move the current function in all directions at each iteration, eventually considering all paths through the function space with each weighted by some posterior probability. As is often the case, computation can impede a fully Bayesian solution. However, the lessons learned from boosting could be emminently useful for Bayesian prediction and function estimation problems.

In Bayesian prediction problems we want to find the predictive distribution of a y given its covariates, x, and a dataset containing previous observations,

$$\Pr(y|x, D) = \int \Pr(y|x, F)\Pr(F|D)\, dF. \tag{6}$$

The predictive distribution integrates over all members, F, in some class of functions. The crudest approximation simply assumes that $\Pr(F|D)$ has most of its mass in a small neighborhood of the maximum likelihood estimate of F so that $\Pr(y|x, D) \approx \Pr(y|x, \hat{F})$. To make a point prediction one would compute some functional of $\Pr(y|x, \hat{F})$ (e.g., mean, class with the largest probability). Recent efforts have proposed methods for sampling from $\Pr(F|D)$ in order to compute a Monte Carlo estimate of (6). Some of these ideas include MCMC on the space of trees [Denison, Mallick and Smith (1996), Chipman, George and McCulloch (1998)], on the space of additive models with a partic-

ular smoothness [Hastie and Tibshirani (1998)] and reversible jump MCMC on the space of piecewise constants [Heikkinen (1998)] or splines [DiMatteo, Genovese and Kass (1999)]. I think boosting may have a lot to offer the latter of these methods.

Heikkinen (1998) assumes that the regression function is a piecewise constant where the number of pieces and the constants associated with those pieces are random quantities. The simulation from $\Pr(F|D)$ proceeds by either randomly combining two neighboring pieces, splitting a random piece into two, or randomly perturbing the constant associated with one of the pieces. Among the curses of MCMC methods is the problem of wasting iterations exploring regions of the parameter space that are not particularly of interest or not exploring the entire interesting region. Each boosting iteration computes some kind of gradient in the function space indicating with respect to the current state which neighboring functions are most interesting in some sense. From the insight gained from boosting one could potentially produce more efficient proposal mechanisms for exploring $\Pr(F|D)$; that is in the works.

Closing comments. This work melds some exciting work in both computational learning theory and statistics. The authors have elegantly connected the AdaBoost algorithm with a novel interpretation of fitting additive logistic regression. Researchers from a diverse set of fields are in search of better methods for modeling datasets that are massive, noisy, complex and high dimensional. Boosting, as a modular procedure for constructing such models, offers an exciting paradigm for future prediction methods.

REFERENCES

BAUER, E. and KOHAVI, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning* **36** 105–139.

BREIMAN, L. (1999). Using adaptive bagging to debias regressions. Technical Report 547, Dept. Statistics, Univ. California, Berkeley.

CHIPMAN, H., GEORGE, E. and MCCULLOCH, R. (1998). Bayesian CART model search (with discussion). *J. Amer. Statist. Assoc.* **93** 935–960.

DENISON, D. (2000). Boosting with Bayesian stumps. Technical report, Dept. Mathematics, Imperial College.

DENISON, D., MALLICK, B. and SMITH, A. (1996). Bayesian CART. Technical report, Dept. Mathematics, Imperial College.

DIMATTEO, I., GENOVESE, C. and KASS, R. (1999). Bayesian curve fitting with free-knot splines. Technical report, Carnegie Mellon Univ.

DRUCKER, H. and CORTES, C. (1996). Boosting decision trees. In *Proceedings of Neural Information Processing 8* 479–485. MIT Press.

ELKAN, C. (1997). Boosting and naïve Bayes learning. Technical Report CS97-557, Univ. California, San Diego.

FREUND, Y. and SCHAPIRE, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* **55** 119–139.

HASTIE, T. and TIBSHIRANI, R. (1998). Bayesian backfitting. Technical report, Stanford Univ.

HEIKKINEN, J. (1998). Curve and surface estimation using dynamic step functions. In *Practical Nonparametric and Semiparametric Bayesian Statistics* (D. Dey, P. Müller and D. Sinha, eds.) 255–272. Springer, New York.

LEE, J. (1999). A computer program that plays a hunch. *New York Times* August 17.

QUINLAN, J. (1996). Bagging, boosting, and C4.5. In *Proceedings Thirteenth American Association for Artificial Intelligence National Conference on Artificial Intelligence* 725–730. AAAI Press, Menlo Park, CA.

RIDGEWAY, G. (1999). The state of boosting. In *Proceedings of the Thirty-first Symposium on the Interface* 172–181.

RIDGEWAY, G. (1999). The state of boosting. In *Computing Science and Statistics* **31** (K. Berk, M. Pourahmadi, eds.) Interface Foundation of North America, 172–181. Fairfax, VA.

DEPARTMENT OF STATISTICS
UNIVERSITY OF WASHINGTON
SEATTLE, WASHINGTON 98195-4322
E-MAIL: greg@stat.washington.edu

REJOINDER

JEROME FRIEDMAN, TREVOR HASTIE AND ROBERT TIBSHIRANI

Stanford University

We thank the discussants for their generous and thoughtful comments, and the editors for arranging this discussion. Since the paper and discussions are long, we can only reply to a few of the many important points raised by the discussants.

1. Overfitting. All of the discussions raise the issue of overfitting in the context of boosting. Breiman, and to a lesser extent, Freund and Schapire suggest that our explanation is inadequate since it suggests that enough boosting steps can lead to overfitting. Breiman cites empirical evidence suggesting that "AdaBoost almost never overfits the data no matter how many iterations it is run." It would be a nice world if that were the case, but unfortunately it isn't quite. Besides the example in our paper, Ridgeway provides a simple example that exhibits dramatic overfitting in Figure 2 of his discussion. Quoting Ratsch, Onoda and Muller (2000), "Recent studies with highly noisy patterns [Quinlan (1996), Grove and Schuurmans (1998), Ratsch (1998)] showed that it is clearly a myth that Boosting methods do not overfit." Still, it does appear that boosting is more resistant to overfitting that one might expect based on general modeling experience. Buja suggests that the suboptimal greedy "stagewise" nature of the fitting procedure may provide some resistance. This could be the case, and merits further investigation. However, we agree with Bühlmann and Yu that it is largely the nature of zero–one loss that provides general resistance to overfitting as judged in classification.

We illustrate this with a simple type of "nearest-neighbor" procedure. Consider a fixed set of data points $\{\mathbf{x}_i\}_1^N$. A training data set consists of $\{y_i, \mathbf{x}_i\}_1^N$, where each $y_i \in \{-1, 1\}$ is a random variable with $p_i = \Pr(y_i = 1)$. Given a training set, our estimate for each p_i is $\hat{p}_i = (1 + \alpha y_i)/2$, $0 \le \alpha \le 1$. Here α is a parameter that controls the degree of fit to the training data. Larger values provide a closer fit; the value $\alpha = 1$ interpolates the data, whereas $\alpha = 0$

TABLE 1

Criterion	$L(y, F)$	$\hat{R}$	R	a^*
exponential	$\exp(-yF)$	$(\frac{1-a}{1+a})^{1/2}$	$(1-\bar{e})(\frac{1-a}{1+a})^{1/2}$ $+\bar{e}(\frac{1+a}{1-a})^{1/2}$	$1-2\bar{e}$
likelihood	$\log(1+\exp(-2yF))$	$\log\frac{2}{1+a}$	$(1-\bar{e})\log\frac{2}{1+a}$ $+\bar{e}\log\frac{2}{1-a}$	$1-2\bar{e}$
squared error	$(y-F)^2$	$(1-a)^2$	$(1-a)^2+4a\bar{e}$	$1-2\bar{e}$
absolute error	$\lvert y-F\rvert$	$1-a$	$1-(1-2\bar{e})a$	1
zero–one loss	$1_{[y\neq\text{sign}(F)]}$	0	$\bar{e}$	>0

estimates $\hat{p}_i = 1/2$ irrespective of the data. We consider five loss functions $L(y, F)$—exponential, likelihood, squared error, absolute error, and zero–one loss—all defined in Table 1.

For each loss function, F is defined to be the population minimizer

$$\arg\min_F EL(y, F) = \arg\min_F pL(1, F) + (1-p)L(-1, F).$$

For the exponential and likelihood, this is $F = \frac{1}{2}\log(p/(1-p))$, for squared and absolute error $F = 2p-1$, and for zero–one loss $F = \text{sign}(2p-1)$.

Table 1 shows the average loss on the training data

$$\hat{R} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{F}_i)$$

and the corresponding average population risk

$$R = \frac{1}{N}\sum_{i=1}^{N} EL(y_i^*, \hat{F}_i).$$

The later expectation is with respect to the training y_i (which produces $\hat{F}_i$), and the test observation y_i^*, both drawn independently with probability p_i. The $\hat{F}_i$ are obtained by plugging $\hat{p}_i$ into the expressions for F. For the exponential and likelihood criteria,

$$\hat{F}_i = \frac{1}{2}\log\frac{\hat{p}_i}{1-\hat{p}_i} = \frac{1}{2}\log\frac{1+ay_i}{1-ay_i},$$

whereas for the squared and absolute error criteria,

$$\hat{F}_i = ay_i.$$

The risk values are reported in terms of the parameter a and

$$\bar{e} = \frac{1}{N}\sum_{i=1}^{N} E1_{[y_i^*\neq\text{sign}(F_i^*)]} = \frac{2}{N}\sum_{i=1}^{N} p_i(1-p_i),$$

which is just the average risk based on zero–one loss (and is twice the Bayes risk). This error rate does not depend on the value of a since it is independent of the magnitude of $\hat{F}_i$, depending only on its sign. Thus, overfitting as measured by zero–one loss does not exist for this procedure. Interpolating the data by setting $a = 1$ does not degrade zero–one loss. Also shown in Table 1 is the parameter value a^* that minimizes the population risk for each respective criterion.

As seen from Table 1, both the average training set loss and population risk for the four criteria, other than zero–one loss, do depend on the magnitude of $\hat{F}_i$, and thus depend strongly on how close the training data are fit, as regulated by the value of a. The average training set loss $\hat{R}$ is monotone decreasing with increasing values of a. Starting at $a = 0$, the population risk R for all four criteria initially decreases. For the first three a minimum is reached at the value $a^* = 1 - 2\bar{e}$. Increasing the value of a beyond that point monotonically increases population risk, symptomatic of classic overfitting. The degree of overfitting depends on the criterion. Criteria listed higher in Table 1 report more overfitting. For absolute error there is no overfitting; interpolating the training data here minimizes population risk for this criterion.

Figure 1 of this Rejoinder shows both training error (lowest green curve), population risk for $\bar{e} = 0.1$ (middle red curve), and $\bar{e} = 0.2$ (highest blue curve), as a function of $1/(1 - a)$ for the loss criteria shown in Table 1. All curves are normalized to have unit risk at $a = 0$. Note the difference in vertical scale between the lower and upper plots. The first three criteria exhibit classic overfitting. The degree of this overfit increases with increasing intrinsic noise as reflected by the value of $\bar{e}$. For exponential and likelihood loss the expected test error approaches infinity as the value of a approaches one (data interpolation), the latter much more slowly. Both squared and absolute loss approach finite values of $4\bar{e}$ and $2\bar{e}$, respectively. Except for the discontinuity at $a = 0$ (where the risk is that of a coin flip, $R = \frac{1}{2}$, normalized to be 1), the population zero–one risk curves drop immediately to $\bar{e}$. When the abscissa is plotted on a linear rather than logarithmic scale, the plot for exponential loss (upper left panel) closely resembles that displayed in Figure 3 (lower panel) of the Bühlmann and Yu discussion.

It is important to note that the extent to which overfitting is observed here depends only on the loss criterion used to measure degree-of-fit. For a given training data set, equal values of the parameter a produce identical models for the probabilities $\hat{p}_i$. The judgment as to whether, or how much, a particular model overfits the training data is determined solely by the criterion used to define "fit," and not by the training procedure used to obtain the model. Exponential loss reports dramatic overfitting, likelihood substantial overfitting, squared error loss moderate overfitting and absolute and zero–one loss no overfitting at all, on exactly the same sequence of models.

Of course, this example is highly idealized so as to permit exact calculations of risk. In other situations absolute and zero–one loss may exhibit some degree of overfitting. The assumption that the x-values are fixed prevents zero–one loss from reporting overfitting for this procedure. In actual practice,

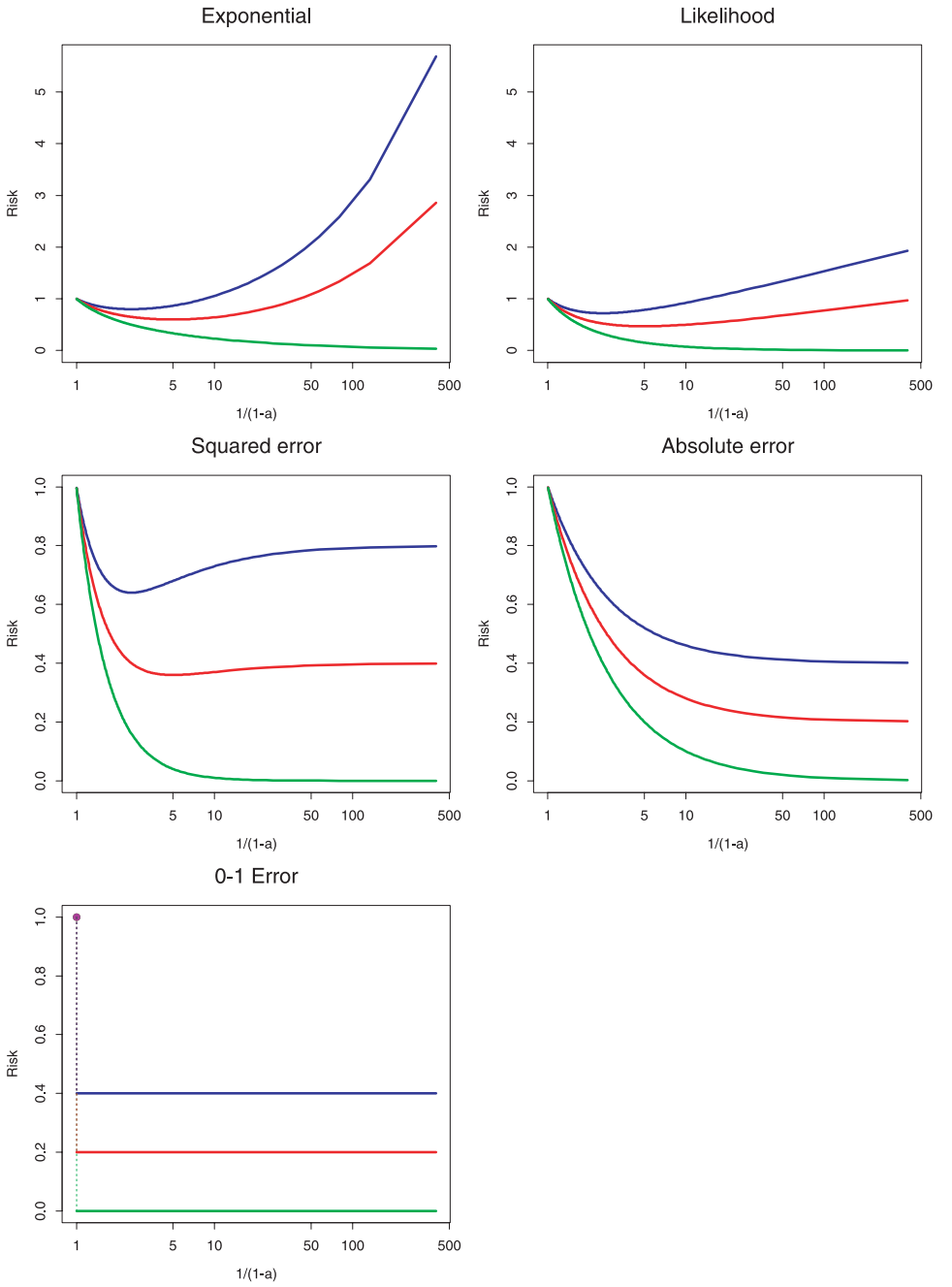

FIG. 1. *Training and population risk curves for the five loss functions in Table* 1, *as a function of* $1/(1-a)$. *The green curves are training risk. The red curves are population risk for* $\bar{e} = 0.1$, *and the blue curves are for* $\bar{e} = 0.2$. *All curves are normalized to have unit risk at* $a = 0$.

the x-values themselves are random variables and fitting the data too closely can result in some incorrect estimates of the sign of F, thereby degrading population zero–one risk. However, the analysis above suggests that overfitting as judged by absolute and especially zero–one loss should be less severe than when judged by exponential, likelihood or squared error loss. Also, there is no reason to expect that the degree-of-fit to the training data that results in minimum population risk will be the same for the former two measures as with the latter three. This is illustrated in Figure 3 of Bühlmann and Yu's discussion.

This analysis has no bearing on the relative merits of any of these criteria as implementing loss functions for boosting, or any other procedure. As pointed out by Bühlmann and Yu, it is often the case that the best criterion to drive a search strategy is not the criterion used to judge the value of the resulting solution. However, the phenomenon of "overfitting" is largely dictated by the latter.

In the field of statistics most of our experience has been derived from likelihood and squared error loss. As Breiman points out "adding more and more variables to a logistic regression, increasing the training likelihood at each step, usually results in overfitting at some point." This is surely the case when overfitting is judged by the likelihood criterion. It can sometimes be the case when judged by zero–one loss. However, the simple example presented here, as well as considerable empirical evidence elsewhere, suggest that zero–one loss, being less sensitive to variations in the estimates of the underlying probabilities (they depend only on the sign of $2\hat{p}_i - 1$), exhibit considerably less overfitting than other loss criteria that depend more heavily on the detailed probability estimates. This will be the case whether those estimates are obtained by AdaBoost, LogitBoost or any other procedure.

2. Regression. Most of the discussions consider the extension of boosting to regression. Buja and Bühlmann and Yu, as well as Ridgeway (1999), observe that our view of boosting leads to natural extensions to regression based on squared error loss. This is discussed in Section 10 of the paper. It is explored in more detail in Friedman (1999a), where gradient based boosting is extended to arbitrary differentiable loss criteria. As noted in Friedman (1999a), shrinkage is an important ingredient to the success of boosting in the regression context. At each step m of the iterative procedure, the current approximation $F_{m-1}(\mathbf{x})$ is updated by

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot f_m(\mathbf{x}). \tag{7}$$

Here $f_m(\mathbf{x})$ is the update computed in the usual manner by the boosting algorithm at the mth step, and $\nu \in (0, 1]$ is a parameter whose value controls the amount of shrinkage. Empirical evidence was presented that such shrinkage dramatically improves the accuracy of the target function estimate when measured by likelihood, squared error or absolute error. Less dramatic but significant gains were also observed for zero–one loss, where the quality of the function estimate (as judged by the other criteria) is less important. In

all cases, accuracy improved monotonically with decreasing value of ν (more shrinkage), provided that the number of iterations M was correspondingly increased so as to reach minimum test error.

Breiman and Bühlmann and Yu suggest that incorporating a stochastic component into boosting algorithms can substantially improve performance. Bühlmann and Yu propose "bag-boosting" which, as Ridgeway points out, is the essence of Breiman's (1999) "adaptive bagging" procedure. The base learner is replaced by the corresponding bagged learner at each boosting step. This can be viewed as an alternative to shrinkage for reducing the learning rate. The individual bagging substeps can be regarded as randomized boosting steps without updating the approximation. Friedman (1999b) investigated combining randomization with shrinkage in an integrated manner. The approximation is slowly but continuously updated at each randomized boost using a slow learning rate, rather than alternating between a large number of steps without updating (bagging), followed by a single step with a full update (boost). Combining randomization with shrinkage in this manner provides a modest but significant additional improvement over the already large gain produced by shrinkage alone. However, there is a substantial computational advantage by employing what Bühlmann and Yu call "subbagging." Friedman and Hall (1999) discuss the equivalence of bagging based on boostrapping, and half-sampling without replacement and show that different subsample sizes can often lead to better performance.

3. Interpretations of boosting. For their invention of boosting, Freund and Schapire deserve a great deal of praise. It has led to both practical learning algorithms and much intellectual study and debate. As Buja so poetically states, "There is no single 'true' interpretation of anything; interpretation is a vehicle in the service of human comprehension. The proof of an interpretation is in enabling others to fruitfully think about an idea." We hope that our work has helped serve that goal. Freund and Schapire and Breiman present two additional interpretations, different from ours and each other. Ridgeway considers yet another view along Bayesian lines. Having multiple interpretations can only increase the chance of obtaining new insights.

We don't see a great deal of contradiction between our view and that of Freund and Schapire. "Provable bounds" can be derived for LogitBoost (Nigel Duffy and David Helmbold, private communication). Also, we don't see how our "assumptions" are necessarily stronger (or weaker) than those motivating their interpretation. It is hard to imagine a binary random variable that is not Bernoulli distributed. We do not assume "a log-linear relationship between the features and the labels." The function class describing this relationship is determined by the weak learner employed. AdaBoost, LogitBoost, or any other boosting method using the same learner is making similar assumptions concerning the label–feature relationship. As was pointed out in the paper, and amplified by Bühlmann and Yu and Buja, "The question is not *whether* but *how much* a weak learner can be boosted, or, in statistical terminology, how well an additive model can approximate an underlying response surface."

The underlying response surface is the unknown label–feature relationship. The weak learner provides the additive components. Different weak learners will perform differently with different response surfaces.

Freund and Mason's (1999) "alternating decision tree" approach is quite interesting. It shares many common elements with MARS [Friedman (1991)], substituting exponential loss for squared error loss, and using zero-order rather than first-order splines. Both of these modifications may be beneficial in the classification context.

Freund's (1999) suggestion of removing outliers is a good one, but not "radical." This approach is at the center of the field of robust statistics. The source of AdaBoost's lack of robustness is its use of exponential loss as the implementing loss function, which increases exponentially with large negative margin. Negative log-likelihood increases only linearly with large negative margin, thereby being much more robust. Still, sometimes performance can be further improved a little by completely removing the influence of observations with very large negative margins. This is accomplished as a beneficial side effect of the weight trimming strategy described in Section 9 of the paper, when used with LogitBoost. The LogitBoost weights are $w_i = p(x_i)(1 - p(x_i))$. Observations with the smallest weights at the current iteration are removed from consideration for that iteration. Observations with large negative margin have either $p(x_i) \simeq 0$ (for $y_i = 1$) or $p(x_i) \simeq 1$ (for $y_i = -1$), thereby being deleted.

Breiman views boosting as an "ensemble" method for combining classifiers. He seeks to incorporate boosting and bagging into a common framework based on "random forests." This is an intriguing concept. However, it seems to us that boosting operates differently than random forests. From the description, a random forests appears to be an ensembles of learners that are i.i.d. given the data. Hence the bias of the ensemble is the same as that of each individual learner. By contrast, in boosting the individual members are not independent given the data: the construction of each depends explicitly on those members that have been previously entered. The bias of the ensemble is typically much less than that of each constituent learner. Boosting turns a weak learner to strong ensemble, whereas random forests combine strong learners in an i.i.d. fashion in order to reduce variance.

Viewing boosting as stagewise additive regression does however suggest Breiman's conclusion that "if boosting succeeds, it is because it gives low correlation between classifiers of reasonable strength." In stagewise regression "each step decouples itself from the previous step" leading to "low correlation between pairs of classifiers." In general, a greedy search strategy will not successively add highly correlated terms, since that would not produce very much local reduction in the criterion being minimized. Also, there will be a tendency to add terms of "reasonable strength," since that will tend to reduce the criterion. Thus, one could conclude that AdaBoost owes its success to these particular aspects of stagewise additive regression. This is consistent with Breiman's explanation, although motivated somewhat differently. Unfortunately, this conclusion appears to be in conflict with one empirical fact,

namely, the success of shrinking (1) in increasing accuracy with all boosting methods, including AdaBoost [Friedman (1999a)].

Shrinking strongly inhibits the natural tendency of greedy search strategies to add low correlation terms of reasonable strength. For smaller values of the shrinkage parameter ν, the strength of each added term $\nu \cdot f_m(\mathbf{x})$ becomes smaller, and more highly correlated with recently entered terms. Yet, the evidence so far indicates that the smaller the value of ν, the higher the overall accuracy, as long as there are enough iterations.

The source of this mystery in under investigation. It is but one of the many aspect of this general class of procedures that are not yet well understood. Whether these procedures are referred to as "boosting," or more prosaically as "stagewise additive regression," there are still many problems left to solve that can challenge us for some time to come.

REFERENCES

BREIMAN, L. (1999). Using adaptive bagging to debias regressions. Technical Report 547, Dept. Statistics, Univ. California, Berkeley.

FREUND, Y. (1999). An adaptive version of boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*.

FREUND, Y. and MASON, L. (1999). The alternating decision tree learning algorithm. In *Machine Learning: Proceedings of the Sixteenth International Conference* 124–133.

FRIEDMAN, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Statist.* **19** 1–141.

FRIEDMAN, J. H. (1999a). Greedy function approximation: a gradient boosting machine. *Ann. Statist.* To appear.

FRIEDMAN, J. H. (1999b). Stochastic gradient boosting. Technical report, Dept. Statistics, Stanford Univ.

FRIEDMAN, J. H. and HALL, P. (1999). On bagging and nonlinear estimation. *J. Comput. Graph. Statist.* To appear.

GROVE, A. and SCHUURMANS, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*.

QUINLAN, J. (1996). Boosting first order learning. In *Proceedings of the Seventh International Workshop on Algorithmic Learning Theory* (S. Arikawa and A. Sharma, eds.) *Lecture Notes in Artificial Intelligence* **1160** 143–155. Springer, Berlin.

RATSCH, G. (1998). Ensemble learning methods for classification. Masters thesis, Dept. Computer Science, Univ. Potsdam.

RATSCH, G., ONODA, T. and MULLER, K. R. (2000). Soft margins for AdaBoost. *Machine Learning* 1–35.

RIDGEWAY, G. (1999). The state of boosting. In *Proceedings of the Thirty-first Symposium on the Interface* 172–181.

DEPARTMENT OF STATISTICS
SEQUOIA HALL
370 SERRA MALL
STANFORD UNIVERSITY
STANFORD CALIFORNIA 94305-4065
E-MAIL: hastie@stat.stanford.edu