Additive Logistic Regression: a Statistical View of Boosting

JEROME FRIEDMAN [*] [†]
TREVOR HASTIE [*] [‡]
ROBERT TIBSHIRANI [*] [‡]

August 20, 1998
Revision April 24, 1999
Second Revision November 30, 1999

Abstract

Boosting (Schapire 1990, Freund & Schapire 1997) 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 multi-class generalizations based on multinomial likelihood are derived that exhibit performance comparable to other recently proposed multi-class 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 & 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 & Schapire 1996*b*), which we call *Discrete AdaBoost*[1]. 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

[*]Department of Statistics, Sequoia Hall, Stanford University, California 94305. {jhf,hastie,tibs}@stat.stanford.edu
[†]Stanford Linear Accelerator Center, Stanford, CA94305
[‡]Division of Biostatistics, Department of Health, Research and Policy, Stanford University, Stanford CA94305.
[1]Essentially the same as AdaBoost.M1 for binary data (Freund & Schapire 1996*b*)

$F(x) = \sum_1^M c_m f_m(x)$ where each $f_m(x)$ is a classifier producing values ± 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.

Discrete AdaBoost(Freund & Schapire 1996*b*)

1. Start with weights $w_i = 1/N$, $i = 1, \dots, N$.
2. Repeat for $m = 1, 2, \dots, M$:
   (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 $w_i \leftarrow w_i \exp[c_m \cdot 1_{(y_i \neq f_m(x_i))}]$, $i = 1, 2, \dots N$, 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 $w = (w_1, w_2, \dots w_n)$, 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.*

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 (NIPS workshop, 1996) called AdaBoost with trees the “best off-the-shelf classifier in the world” (see also Breiman (1998*b*)). 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 & Stone 1984) as the base classifier. This adaptation grows fixed-size trees in a “best-first” manner (see Section 8, page 26). 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.

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:

- 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.
- “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 (Fig. 1[top-right panel].)

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

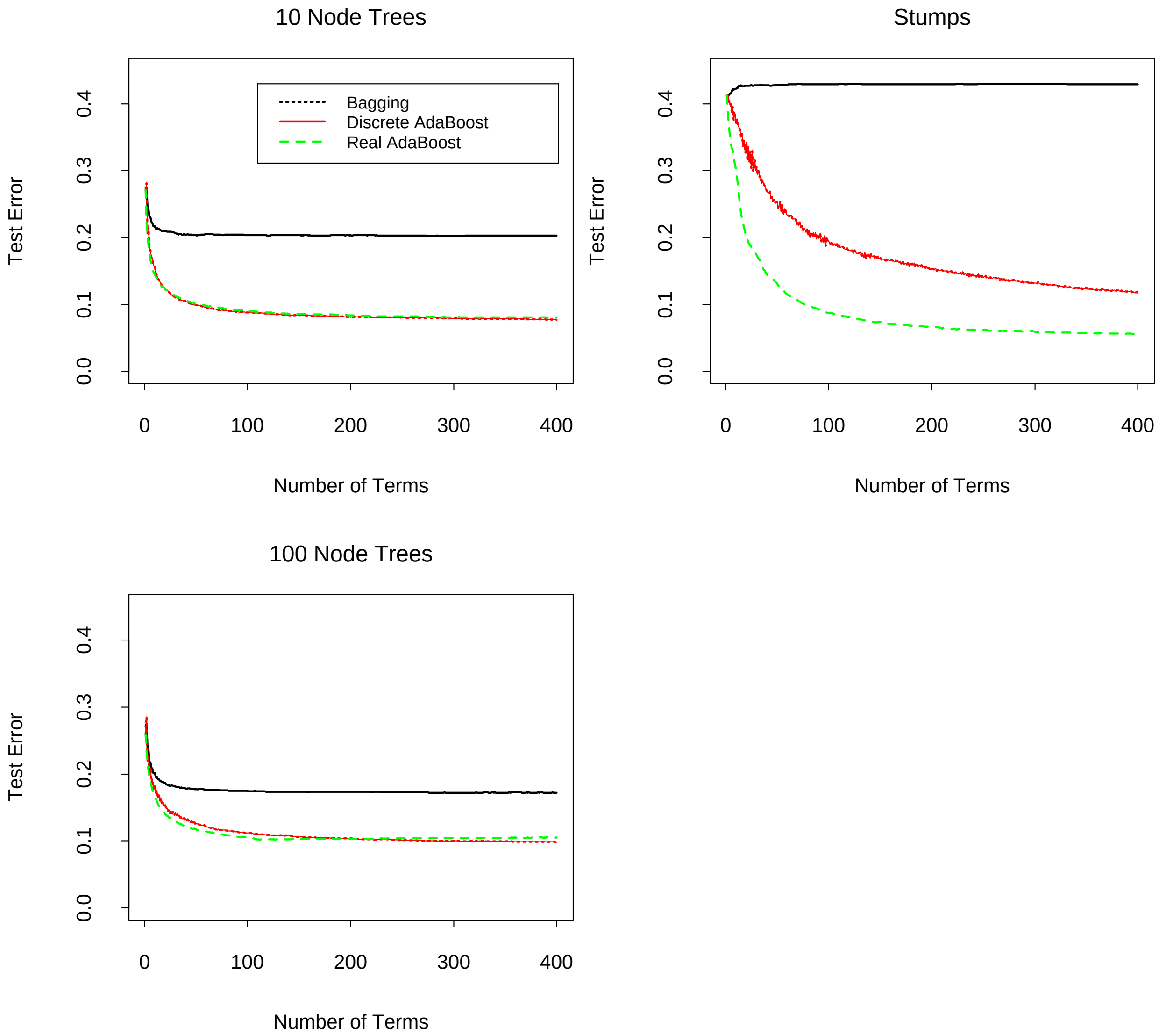

Figure 1: *Test error for Bagging, Discrete AdaBoost and Real AdaBoost on a simulated two-class nested spheres problem (see Section 6 on page 20.) There are 2000 training data points in 10 dimensions, and the Bayes error rate is zero. All trees are grown "best-first" without pruning. The left-most iteration corresponds to a single tree.*

The *base classifier* in Discrete AdaBoost produces a classification rule $f_m(x) : \mathcal{X} \mapsto \{-1, 1\}$, where $\mathcal{X}$ is the domain of the predictive features x. Freund & Schapire (1996*b*), Breiman (1998*a*) and Schapire & Singer (1998) have suggested various modifications to improve the boosting algorithms. A generalization of Discrete AdaBoost appeared in Freund & Schapire (1996*b*), and was developed further in Schapire & 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 $f_m(x) : \mathcal{X} \mapsto R$; 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 Fig. 1, especially with stumps, although we see with 100 node trees Discrete AdaBoost overtakes Real AdaBoost after 200 iterations.

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 $f_m(x) \leftarrow \frac{1}{2} \log \frac{p_m(x)}{1 - p_m(x)} \in R$.
   (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 $f_m(x)$.*

In this paper we analyze the AdaBoost procedures from a statistical perspective. The main result of our paper re-derives 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.

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 $f_1(x) \ldots f_{m-1}(x)$ 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 & Tibshirani 1990).

However squared error loss is not a good choice for classification (see Figure 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 & 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,

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

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 require the weak learner to produce a classifier with a fixed error rate. This led to the more adaptive and realistic AdaBoost (Freund & Schapire 1996*b*) and its offspring, where this assumption was dropped.

Freund & Schapire (1996*b*) and Schapire & 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 & Schapire 1996*a*, Breiman 1997), and VC theory (Schapire, Freund, Bartlett & 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 towards 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

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

Here 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, pre-specified subset of the input variables. The *backfitting algorithm* (Friedman & Stuetzle 1981, Buja, Hastie & Tibshirani 1989) is a convenient modular "Gauss-Seidel" algorithm for fitting additive models. A backfitting update is

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

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 et al. 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 back–fitting 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

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

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 & 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 back–fitting 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 & 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 et al. (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 & Nelder 1989, for example) and additive logistic regression (Hastie & 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 & Tibshirani (1990), can be used to fit the additive logistic model by maximum likelihood. Starting with guesses $f_1(x_1)\dots f_p(x_p)$, $F(x) = \sum f_k(x_k)$ 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 stage-wise 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 stage-wise 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 $F(x)$.[2] 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

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

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

[2] 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.

Proof

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

$$\begin{aligned} E\left(e^{-yF(x)}\middle| x\right) &= P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)} \\ \frac{\partial E\left(e^{-yF(x)}\middle| x\right)}{\partial F(x)} &= -P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)} \end{aligned}$$

The result follows by setting the derivative to zero.

□

This exponential criterion appeared in Schapire & 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 & Schapire (1996*b*), can be motivated as iterative algorithms for optimizing the (population based) exponential criterion. The results share the same format:

- given an imperfect $F(x)$, an update $F(x) + f(x)$ is proposed based on the population version of the criterion.
- 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 & 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 stage-wise 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 $E(e^{-yF(x)})$.*

Derivation

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:

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

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

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

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 Newton-like algorithm.

Parts of this derivation for AdaBoost can be found in Breiman (1997) and Schapire & 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 & 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 & Schapire 1996*b*, Schapire & 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 stage-wise and approximate optimization of* $J(F) = E[e^{-yF(x)}]$

Derivation
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

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

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.

At 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 stage-wise fashion, fixing the earlier components. Our term "stage-wise" refers to a similar approach in Statistics:

- Variables are included sequentially in a stepwise regression.
- 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

- Schapire & Singer (1998) motivate $e^{-yF(x)}$ as a differentiable upper-bound to misclassification error $1_{[yF<0]}$ (see Fig. 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

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

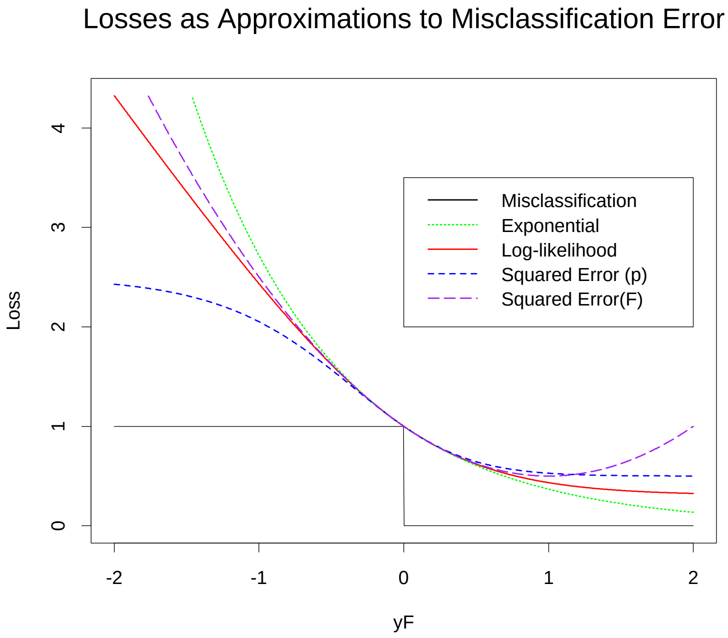

Figure 2: *A variety of loss functions for estimating a function $F(x)$ for classification. The horizontal axis is yF, which is negative for errors and positive for correct classifications. All the loss functions are monotone in yF, and are centered and scaled to match e^{-yF} at $F = 0$. The curve labeled "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:

- The population minimizers of $-E\ell(y^*, p(x))$ 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 $F = 0$:

  $$-\ell(y^*, p) \approx \exp(-yF) + \log(2) - 1 \quad (28)$$

  Graphs of $\exp(-yF)$ and $\log(1 + e^{-2yF(x)})$ are shown in Fig. 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 ± 1.

- 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 stage-wise 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 non-monotonicity of squared error loss (Fig. 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 stage-wise 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.

Result 3 *The LogitBoost algorithm (2 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

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

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

$$\begin{aligned} s(x) &= \frac{\partial E\ell(F(x)+f(x))}{\partial f(x)}\Big|_{f(x)=0} \\ &= 2E(y^* - p(x)|x) \qquad (32) \\ H(x) &= \frac{\partial^2 E\ell(F(x)+f(x))}{\partial f(x)^2}\Big|_{f(x)=0} \\ &= -4E(p(x)(1-p(x))|x) \qquad (33) \end{aligned}$$

LogitBoost (2 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
   $$\begin{aligned} z_i &= \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))} \\ w_i &= p(x_i)(1 - p(x_i)) \end{aligned}$$
   (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 $p(x) \leftarrow \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$.
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.*

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

$$\begin{aligned} F(x) &\leftarrow F(x) - H(x)^{-1} s(x) \\ &= F(x) + \frac{1}{2} \frac{E(y^* - p(x)|x)}{E(p(x)(1 - p(x))|x)} && (34) \\ &= F(x) + \frac{1}{2} E_w \left(\frac{y^* - p(x)}{p(x)(1 - p(x))} |x \right) && (35) \end{aligned}$$

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

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

□

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 et al. 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 led to the following crucial implementation protections:

- If $y^* = 1$, then compute $z = \frac{y^* - p}{p(1-p)}$ as $\frac{1}{p}$. Since this number can get large if p is small, threshold this ratio at *zmax*. The particular value chosen for *zmax* is not crucial; we have found empirically that $zmax \in [2, 4]$ works well. Likewise, if $y^* = 0$, compute $z = \frac{-1}{(1-p)}$ with a lower threshold of $-zmax$.

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

4.4 Optimizing $Ee^{-yF(x)}$ by Newton stepping

The population version of the Real AdaBoost procedure (Algorithm 2) optimizes $Ee^{-y(F(x)+f(x))}$ 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.

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 $w_i \leftarrow w_i e^{-y_i f_m(x_i)}$ 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*

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

Derivation

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

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

$\square$

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 log-ratio as in (24): $f_m(x) = \frac{1}{2}\log\frac{P_w(y=1|x)}{P_w(y=-1|x)}$. 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 $p(x) = \frac{e^{F(x)}}{e^{F(x)}+e^{-F(x)}}$. 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 & 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,

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

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 & Singer (1998) provide several generalizations of AdaBoost for the multiclass case, and also refer to other proposals (Freund & Schapire 1997, Schapire 1997); we describe their *AdaBoost.MH* algorithm (see boxed 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:

- 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.
- The same is almost true for a tree-based algorithm. We see this because

AdaBoost.MH (Schapire & 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 $F : \mathcal{X} \times (1, \ldots, J) \mapsto R$; $F(x, j) = \sum_m f_m(x, j)$.

3. Output the classifier $\text{argmax}_j F(x, j)$.

Algorithm 5: *The AdaBoost.MH algorithm converts the J class problem into that of estimating a 2 class classifier on a training set J times as large, with an additional "feature" defined by the set of class labels.*

1. If the first split is on C — either a J-nary split if permitted, or else $J-1$ binary splits — then the sub-trees are identical to separate trees grown to each of the J groups. This will always be the case for the first tree.
2. 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/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 $\sum_{j=1}^{J} E e^{-y_j F_j(x)}$

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.*

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*

Derivation

- We first give the score and Hessian for the population Newton algorithm corresponding to a standard multi-logit parametrization

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

LogitBoost (J classes)

1. Start with weights $w_{ij} = 1/N$, $i = 1, \dots, N$, $j = 1, \dots, J$, $F_j(x) = 0$ and $p_j(x) = 1/J \ \forall j$.
2. Repeat for $m = 1, 2, \dots, M$:
   (a) Repeat for $j = 1, \dots, J$:
      i. Compute working responses and weights in the jth class
      $$\begin{aligned} z_{ij} &= \frac{y_{ij}^* - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))} \\ w_{ij} &= p_j(x_i)(1 - p_j(x_i)) \end{aligned}$$
      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 $\text{argmax}_j F_j(x)$

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

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

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

$$\begin{aligned} s_j(x) &= E(y_j^* - p_j(x)|x), \ j = 1, \dots, J-1 \\ H_{j,k}(x) &= -p_j(x)(\delta_{jk} - p_k(x)), \ j, k = 1, \dots, J-1 \end{aligned}$$

- Our quasi-Newton update amounts to using a diagonal approximation to the Hessian, producing updates:
$$g_j(x) \leftarrow \frac{E(y_j^* - p_j(x)|x)}{p_j(x)(1 - p_j(x))}, \ j = 1, \dots, J-1$$
- To convert to the symmetric parametrization, we would note that $g_J = 0$, and set $f_j(x) = g_j(x) - \frac{1}{J}\sum_{k=1}^{J} g_k(x)$. 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 10000 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 that 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

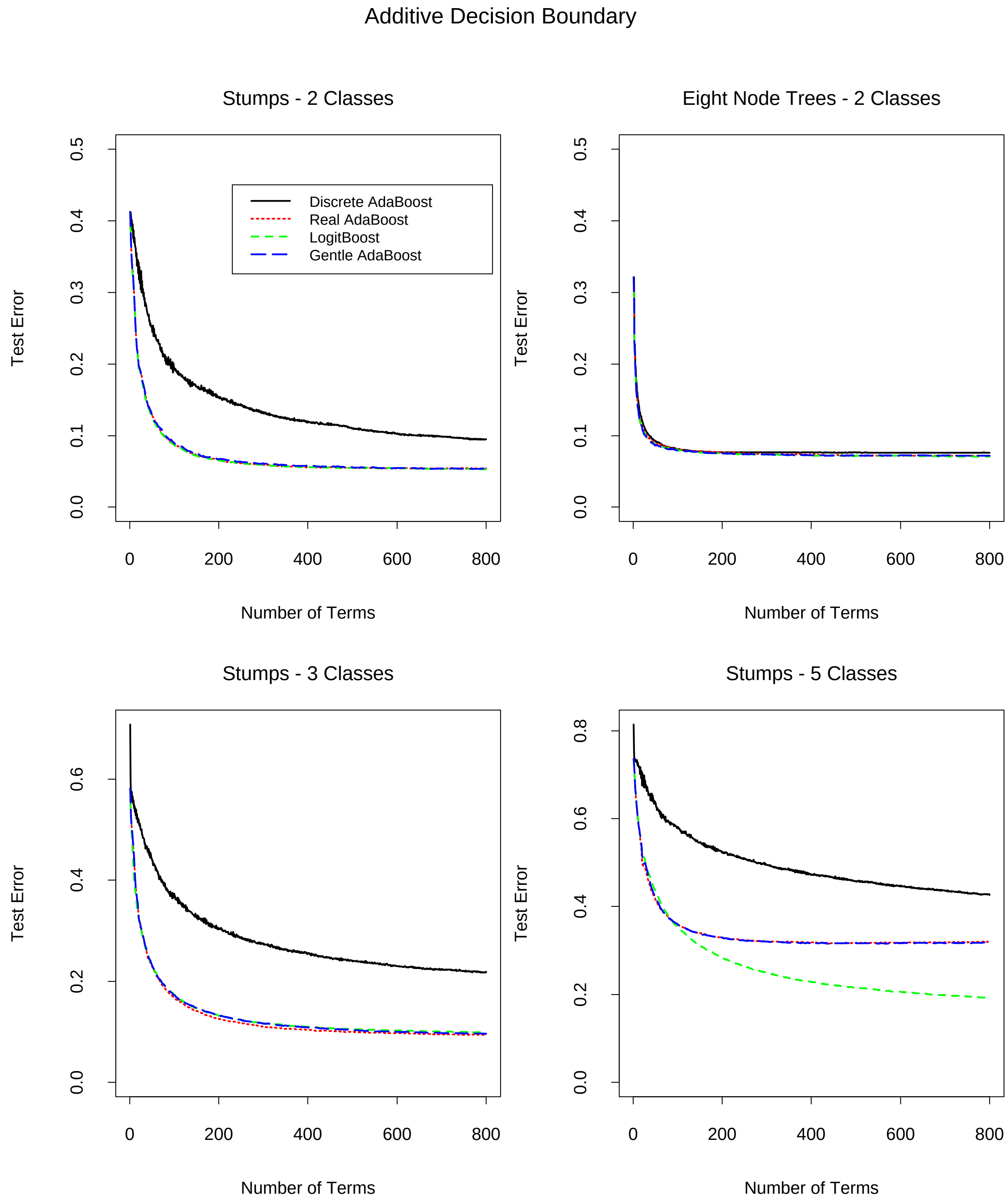

Figure 3: *Test error curves for the simulation experiment with an additive decision boundary, as described in (42) on page 20. In all panels except the the top right, the solid curve (representing Discrete AdaBoost) lies alone above the other three curves.*

and the other 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 Fig. 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 Fig. 3 one sees that for RAB, GAB, and LB the error rate using the bigger trees (.072) is in fact 33% higher than that for stumps (.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:

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

or simply $\{x : B(x) = 0\}$. 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

$$\begin{aligned} f_m(x) &= c_m^L 1_{[x_j \leq t_m]} + c_m^R 1_{[x_j > t_m]} && (44) \\ &= f_m(x_j) && (45) \end{aligned}$$

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 counter productive 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 stage-wise 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 non-additive 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 Fig. 3.

The above discussion also suggests that if the decision boundary separating pairs of classes were inherently *non-additive* 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 10000 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 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

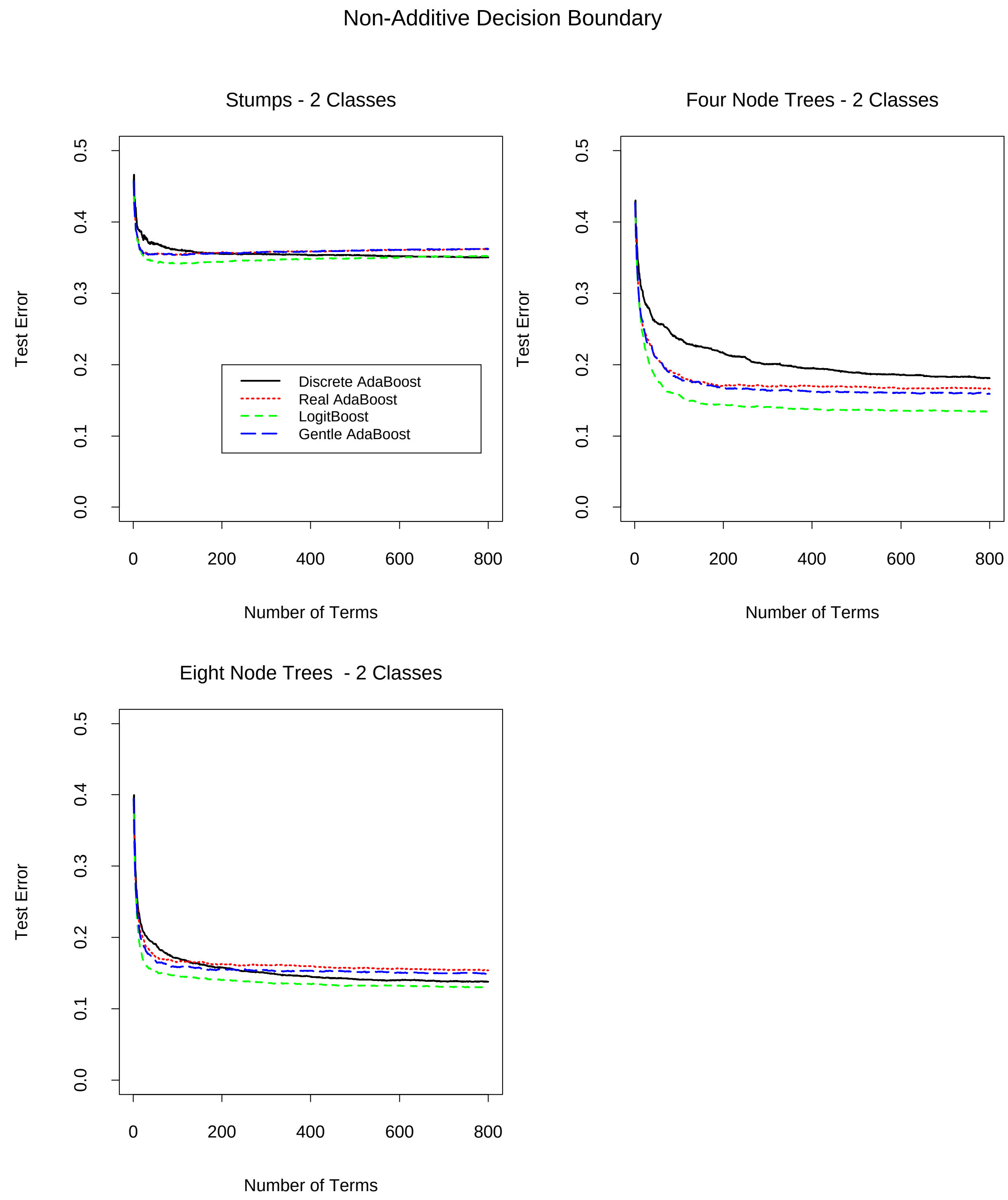

Figure 4: *Test error curves for the simulation experiment with a non-additive decision boundary, as described in (47) on page 23.*

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 Fig. 3 [lower-right] appears to be a consequence of the multi–class 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 multi–class model is better able to take advantage of simple pairwise boundaries when they exist (Hastie & 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 illicit 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 2 or 8 terminal nodes. For comparison, a single decision tree was also fit[3], 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 pre-specified test set. The `waveform` data is simulated, as described in (Breiman et al. 1984). For the others, 5-fold cross-validation was used to estimate the test error.

It is difficult to discern trends on the small data sets (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 performance, 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

[3]using the tree function in Splus

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

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 the 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 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. But, after a sufficient number of boosts, the stump based model achieved superior performance.

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

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

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

Table 2: *Test error rates on small real examples*

Method		2 Terminal Nodes			8 Terminal Nodes		
	Iterations	50	100	200	50	100	200
`Vowel`	CART error= .642						
LogitBoost		.532	.524	.511	.517	.517	.517
Real AdaBoost		.565	.561	.548	.496	.496	.496
Gentle AdaBoost		.556	.571	.584	.515	.496	.496
Discrete AdaBoost		.563	.535	.563	.511	.500	.500
`Breast`	CART error= .045						
LogitBoost		.028	.031	.029	.034	.038	.038
Real AdaBoost		.038	.038	.040	.032	.034	.034
Gentle AdaBoost		.037	.037	.041	.032	.031	.031
Discrete AdaBoost		.042	.040	.040	.032	.035	.037
`Ion`	CART error= .076						
LogitBoost		.074	.077	.071	.068	.063	.063
Real AdaBoost		.068	.066	.068	.054	.054	.054
Gentle AdaBoost		.085	.074	.077	.066	.063	.063
Discrete AdaBoost		.088	.080	.080	.068	.063	.063
`Glass`	CART error= .400						
LogitBoost		.266	.257	.266	.243	.238	.238
Real AdaBoost		.276	.247	.257	.234	.234	.234
Gentle AdaBoost		.276	.261	.252	.219	.233	.238
Discrete AdaBoost		.285	.285	.271	.238	.234	.243
`Sonar`	CART error= .596						
LogitBoost		.231	.231	.202	.163	.154	.154
Real AdaBoost		.154	.163	.202	.173	.173	.173
Gentle AdaBoost		.183	.183	.173	.154	.154	.154
Discrete AdaBoost		.154	.144	.183	.163	.144	.144
`Waveform`	CART error= .364						
LogitBoost		.196	.195	.206	.192	.191	.191
Real AdaBoost		.193	.197	.195	.185	.182	.182
Gentle AdaBoost		.190	.188	.193	.185	.185	.186
Discrete AdaBoost		.188	.185	.191	.186	.183	.183

Table 3: *Test error rates on larger data examples.*

Method	**Terminal Nodes**	**Iterations** 20	50	100	200	**Fraction**
`Satimage`	CART error = .148					
LogitBoost	2	.140	.120	.112	.102	
Real AdaBoost	2	.148	.126	.117	.119	
Gentle AdaBoost	2	.148	.129	.119	.119	
Discrete AdaBoost	2	.174	.156	.140	.128	
LogitBoost	8	.096	.095	.092	.088	
Real AdaBoost	8	.105	.102	.092	.091	
Gentle AdaBoost	8	.106	.103	.095	.089	
Discrete AdaBoost	8	.122	.107	.100	.099	
`Letter`	CART error = .124					
LogitBoost	2	.250	.182	.159	.145	.06
Real AdaBoost	2	.244	.181	.160	.150	.12
Gentle AdaBoost	2	.246	.187	.157	.145	.14
Discrete AdaBoost	2	.310	.226	.196	.185	.18
LogitBoost	8	.075	.047	.036	.033	.03
Real AdaBoost	8	.068	.041	.033	.032	.03
Gentle AdaBoost	8	.068	.040	.030	.028	.03
Discrete AdaBoost	8	.080	.045	.035	.029	.03

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"

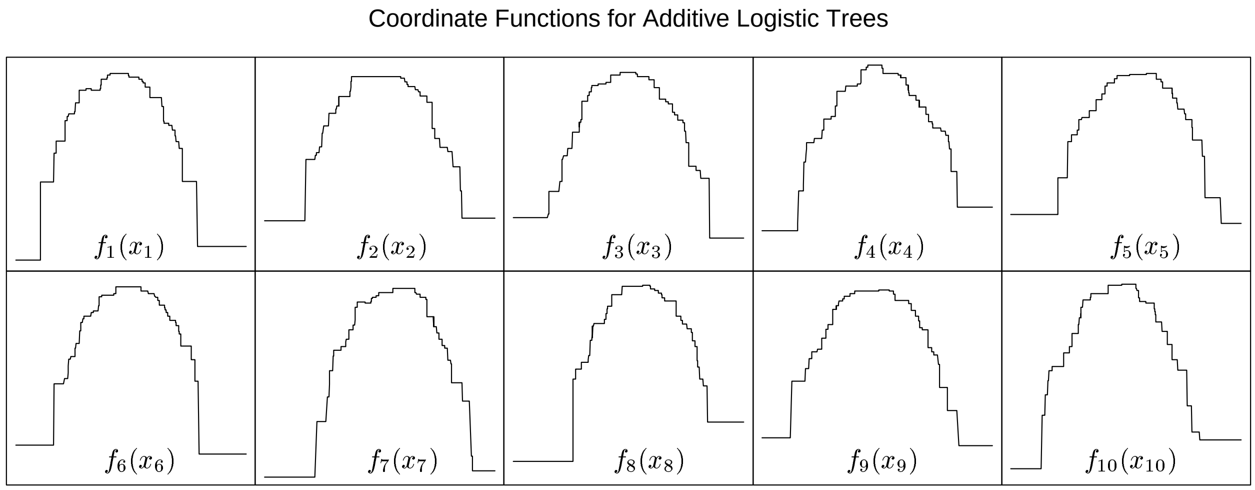

Figure 5: *Coordinate functions for the additive logistic tree obtained by boosting (Logitboost) with stumps, for the two-class nested sphere example from Section 6.*

(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 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 Fig. 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 Fig. 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 Fig. 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 non-interaction (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 influential. 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 percent 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 re-enter 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 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 Fig. 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

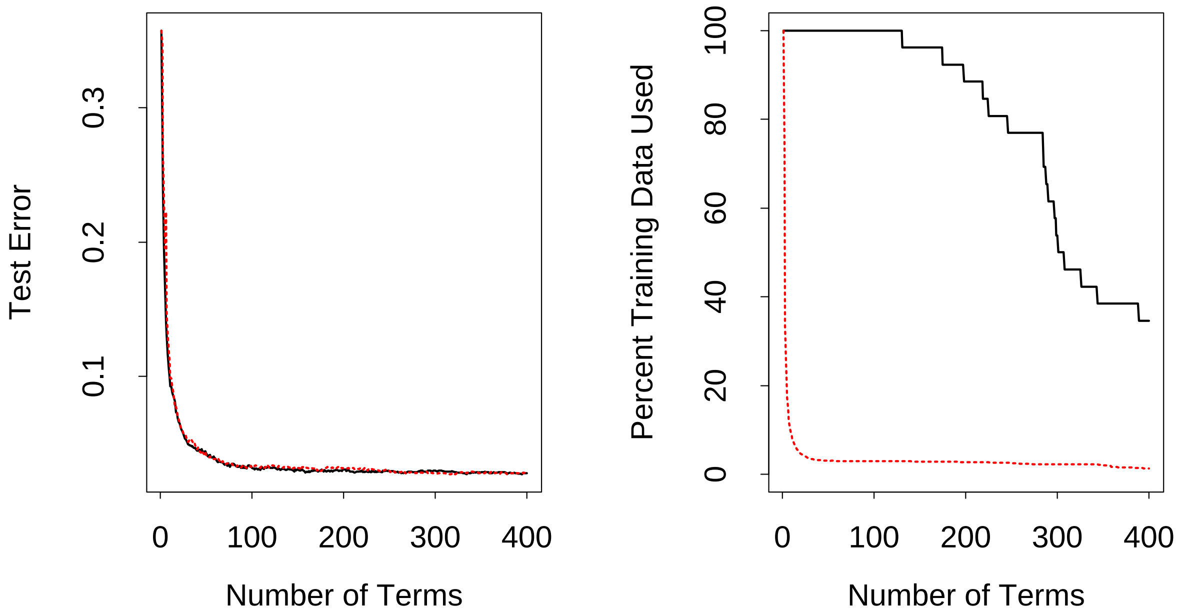

Figure 6: *The left panel shows the test error for the letter recognition problem as a function of iteration number. The black solid curve uses all the training data, the red dashed curve uses a subset based on weight thresholding. The right panel shows the percent of training data used for both approaches.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

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 generalizatons 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. But 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 distribution. 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 multi-class 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 & 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 multi-class 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 1998*a*) when boosting is implemented by finite weighted random sampling instead of weighted optimization. The advantages/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:

- 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.

- The stage-wise 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.

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

Figure 7 shows a case where boosting does overfit. The data are generated from two 10-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 et al. (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;

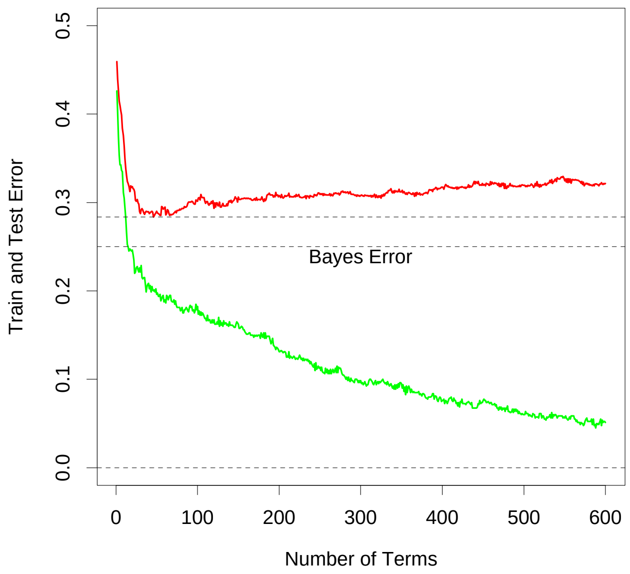

Figure 7: *Real AdaBoost (stumps) on a noisy concentric-sphere problem, with 400 observations per class and Bayes error* 25%. *The test error (upper curve) increases after about fifty iterations.*

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. Jerome Friedman was partially supported by the Department of Energy under contract number DE-AC03-76SF00515 and by grant DMS-9764431 of the National Science Foundation. Trevor Hastie was partially supported by grants DMS-9504495 and DMS-9803645 from the National Science Foundation, and grant ROI-CA-72028-01 from the National Institutes of Health. Robert Tibshirani was supported by the Natural Sciences and Engineering Research Council of Canada.

References

Breiman, L. (1996), 'Bagging predictors', *Machine Learning 26* **24**, 123–140.

Breiman, L. (1997), Prediction games and arcing algorithms, Technical Report Technical Report 504, Statistics Department, University of California, Berkeley. Submitted to Neural Computing.

Breiman, L. (1998*a*), 'Arcing classifiers (with discussion)', *Annals of Statistics* **26**, 801–849.

Breiman, L. (1998*b*), Combining predictors, Technical report, Statistics Department, University of California, Berkeley.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), *Classification and Regression Trees*, Wadsworth, Belmont, California.

Buja, A., Hastie, T. & Tibshirani, R. (1989), 'Linear smoothers and additive models (with discussion)', *Annals of Statistics* **17**, 453–555.

Cover, T. & Hart, P. (1967), 'Nearest neighbor pattern classification', *Proc. IEEE Trans. Inform. Theory* pp. 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', *Information and Computation* **121**(2), 256–285.

Freund, Y. & Schapire, R. (1996*a*), Game theory, on-line prediction and boosting, *in* 'Proceedings of the Ninth Annual Conference on Computational Learning Theory', pp. 325–332.

Freund, Y. & Schapire, R. E. (1996*b*), Experiments with a new boosting algorithm, *in* 'Machine Learning: Proceedings of the Thirteenth International Conference', Morgan Kauffman, San Francisco, pp. 148–156.

Freund, Y. & Schapire, R. E. (1997), 'A decision-theoretic generalization of online learning and an application to boosting', *Journal of Computer and System Sciences* **55**.

Friedman, J. (1991), 'Multivariate adaptive regression splines (with discussion)', *Annals of Statistics* **19**(1), 1–141.

Friedman, J. (1996), Another approach to polychotomous classification, Technical report, Stanford University.

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

Friedman, J. & Stuetzle, W. (1981), 'Projection pursuit regression', *Journal of the American Statistical Association* **76**, 817–823.

Hastie, T. & Tibshirani, R. (1990), *Generalized Additive Models*, Chapman and Hall.

Hastie, T. & Tibshirani, R. (1998), 'Classification by pairwise coupling', *Annals of Statistics* **26**(2).

Hastie, T., Tibshirani, R. & Buja, A. (1994), 'Flexible discriminant analysis by optimal scoring', *Journal of the American Statistical Association* **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', *Annals of Math. Stat.* **53**, 73–101.

Kearns, M. & Vazirani, U. (1994), *An Introduction to Computational Learning Theory*, MIT Press.

Mallat, S. & Zhang, Z. (1993), 'Matching pursuits with time-frequency dictionaries', *IEEE Transactions on Signal Processing* **41**, 3397–3415.

McCullagh, P. & Nelder, J. (1989), *Generalized Linear Models*, Chapman and Hall.

Schapire, R. (1997), Using output codes to boost multiclass learning problems, *in* 'Proceedings of the Fourteenth International Conference on Machine Learning', Morgan Kauffman, San Francisco, pp. 313–321.

Schapire, R. E. (1990), 'The strength of weak learnability', *Machine Learning* **5**(2), 197–227.

Schapire, R. E. & 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. & Lee, W. (1998), 'Boosting the margin: a new explanation for the effectiveness of voting methods', *Annals of Statistics* **26**(5), 1651–1686.

Valiant, L. G. (1984), 'A theory of the learnable', *Communications of the ACM* **27**, 1134–1142.