Multi-class AdaBoost

Ji Zhu[*]
Department of Statistics
University of Michigan
Ann Arbor, MI 48109

Saharon Rosset
Data Analytics Group
IBM Research Center
Yorktown Heights, NY 10598

Hui Zou
School of Statistics
University of Minnesota
Minneapolis, MN 55455

Trevor Hastie
Department of Statistics
Stanford University
Stanford, CA 94305

January 12, 2006

Abstract

Boosting has been a very successful technique for solving the two-class classification problem. In going from two-class to multi-class classification, most algorithms have been restricted to reducing the multi-class classification problem to multiple two-class problems. In this paper, we propose a new algorithm that naturally extends the original AdaBoost algorithm to the multi-class case without reducing it to multiple two-class problems. Similar to AdaBoost in the two-class case, this new algorithm combines weak classifiers and only requires the performance of each weak classifier be better than random guessing (rather than 1/2). We further provide a statistical justification for the new algorithm using a novel multi-class exponential loss function and forward stage-wise additive modeling. As shown in the paper, the new algorithm is extremely easy to implement and is highly competitive with the best currently available multi-class classification methods.

Keywords: Boosting; Exponential loss; Multi-class classification; Stagewise modeling

[*]Address for correspondence: Ji Zhu, 439 West Hall, 1085 South University, Ann Arbor, MI 48109-1107. E-mail: jizhu@umich.edu.

1 Introduction

Boosting has been a very successful technique for solving the two-class classification problem. It was first introduced by Freund & Schapire (1997), with their AdaBoost algorithm. In going from two-class to multi-class classification, most boosting algorithms have been restricted to reducing the multi-class classification problem to multiple two-class problems, e.g. Freund & Schapire (1997), Schapire (1997), Schapire & Singer (1999), Allwein, Schapire & Singer (2000), Friedman, Hastie & Tibshirani (2000), Friedman (2001).

In this paper, we develop a new algorithm that directly extends the AdaBoost algorithm to the multi-class case without reducing it to multiple two-class problems. Similar to AdaBoost in the two-class case, this new algorithm combines weak classifiers and only requires the performance of each weak classifier be better than random guessing (rather than 1/2).

We believe this new algorithm covers a significant gap in the literature, as it is statistically motivated by a novel multi-class exponential loss function. In addition, similar to the original AdaBoost algorithm for the two-class classification, it fits a forward stagewise additive model.

As we will see, the new algorithm is extremely easy to implement, and is highly competitive with the best currently available multi-class classification methods, in terms of both practical performance and computational cost.

1.1 AdaBoost

Before delving into the new algorithm for multi-class boosting, we briefly review the multi-class classification problem and the AdaBoost algorithm (Freund & Schapire 1997). Suppose we are given a set of training data $(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_n, c_n)$, where the input (predictor variable) $\boldsymbol{x}_i \in \mathbb{R}^p$, and the output (response variable) c_i is qualitative and assumes values in a finite set, e.g. $\{1, 2, \ldots, K\}$. K is the number of classes. The goal is to find a classification rule $C(\boldsymbol{x})$ from the training data, so that when given a new input $\boldsymbol{x}$, we can assign it a class label c from $\{1, \ldots, K\}$.

The question, then, is what is the *best* possible classification rule. To answer this question, we need to define what we mean by *best*. A common definition of *best* is to achieve the lowest misclassification error rate. Usually it is assumed that the training data are independently and identically distributed samples from an unknown probability distribution $\text{Prob}(X, C)$. Then the misclassification error rate for $C(\boldsymbol{x})$ is:

$$\begin{aligned} \mathrm{E}_{\boldsymbol{X},C}\mathbb{I}_{C(\boldsymbol{X})\neq C} &= \mathrm{E}_{\boldsymbol{X}}\text{Prob}(C(\boldsymbol{X}) \neq C|\boldsymbol{X}) \\ &= 1 - \mathrm{E}_{\boldsymbol{X}}\text{Prob}(C(\boldsymbol{X}) = C|\boldsymbol{X}) \\ &= 1 - \sum_{k=1}^{K} \mathrm{E}_{\boldsymbol{X}}\left[\mathbb{I}_{C(\boldsymbol{X})=k}\text{Prob}(C = k|\boldsymbol{X})\right]. \end{aligned}$$

It is clear that

$$C^*(\boldsymbol{x}) = \arg\max_k \text{Prob}(C = k|X = \boldsymbol{x})$$

will minimize this quantity with the misclassification error rate equal to $1 - \mathrm{E}_X \max_k \text{Prob}(C = k|X)$. This classifier is known as the *Bayes classifier*, and the error rate it achieves is the *Bayes error rate*.

The AdaBoost algorithm is an iterative procedure that combines many *weak* classifiers to approximate the Bayes classifier $C^*(\boldsymbol{x})$. Starting with the unweighted training sample, the AdaBoost builds a classifier, for example a classification tree (Breiman, Friedman, Olshen & Stone 1984), that produces class labels. If a training data point is misclassified, the weight of that training data point is increased (boosted). A second classifier is built using the new weights, which are no longer equal. Again, misclassified training data have their weights boosted and the procedure is repeated. Typically, one may build 500 or 1000 classifiers this way. A score is assigned to each classifier, and the final classifier is defined as the linear combination of the classifiers from each stage. Specifically, let $T(\boldsymbol{x})$ denote a weak multi-class classifier that assigns a class label to $\boldsymbol{x}$, then the AdaBoost algorithm proceeds as follows:

Algorithm 1 *AdaBoost (Freund & Schapire 1997)*

```
1. Initialize the observation weights $w_i = 1/n,\ i = 1, 2, \ldots, n$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $T^{(m)}(\boldsymbol{x})$ to the training data using weights $w_i$.
    (b) Compute
        $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
    (c) Compute
        $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$
    (d) Set
        $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, 2, \ldots, n.$$
    (e) Re-normalize $w_i$.
3. Output
    $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$
```

When applied to two-class classification problems, AdaBoost has been proved to be extremely successful in producing accurate classifiers. In fact, Breiman (1996) called AdaBoost with trees the "best off-the-shelf classifier in the world." However, it is not the case for multi-class problems, although AdaBoost was also proposed to be used in the multi-class case (Freund & Schapire 1997). Notice that in order for the misclassified training data to be boosted, it is required that the error of each weak classifier $err^{(m)}$ be less than 1/2 (with respect to the distribution on which it was trained), otherwise $\alpha^{(m)}$ will be negative and the weights of the training samples will be updated in the wrong direction in Algorithm 1 (2d). For two-class classification problems, this requirement is about the same as random guessing, but when $K > 2$, accuracy 1/2 may be much harder to achieve than the random guessing accuracy rate $1/K$. Hence, AdaBoost may fail if the weak classifier $T(\boldsymbol{x})$ is not chosen appropriately.

To illustrate this point, we consider a simple three-class simulation example. Each input $\boldsymbol{x} \in \mathbb{R}^{10}$, and the ten input variables for all training examples are randomly drawn from a ten-dimensional standard normal distribution. The three classes are defined as:

$$c = \begin{cases} 1, & \text{if } 0 \leq \sum x_j^2 < \chi^2_{10,1/3}, \\ 2, & \text{if } \chi^2_{10,1/3} \leq \sum x_j^2 < \chi^2_{10,2/3}, \\ 3, & \text{if } \chi^2_{10,2/3} \leq \sum x_j^2, \end{cases}$$

where $\chi^2_{10,k/3}$ is the $(k/3)100\%$ quantile of the χ^2_{10} distribution, so as to put approximately equal numbers of observations in each class. In short, the decision boundaries separating successive classes are nested concentric ten-dimensional spheres. The training sample size is 3000 with approximately 1000 training observations in each class. An independently drawn test set of 10000 observations is used to estimate the error rate.

Figure 1 (upper row) shows how AdaBoost breaks using ten-terminal node trees as weak classifiers. The results are averaged over ten independently drawn training-test set combinations. As we can see (upper left panel), the test error of AdaBoost decreases for a few iterations, then levels off around 0.53. What has happened can be understood from the upper middle and upper right panels: the $err^{(m)}$ starts below 0.5; after a few iterations, it overshoots 0.5 ($\alpha^{(m)}$ below 0), then quickly hinges onto 0.5. Once $err^{(m)}$ is equal to 0.5, the weights of the training samples do not get updated ($\alpha^{(m)} = 0$), hence the same weak classifier is fitted over and over again but is not added to the existing fit, and the test error rate stays the same.

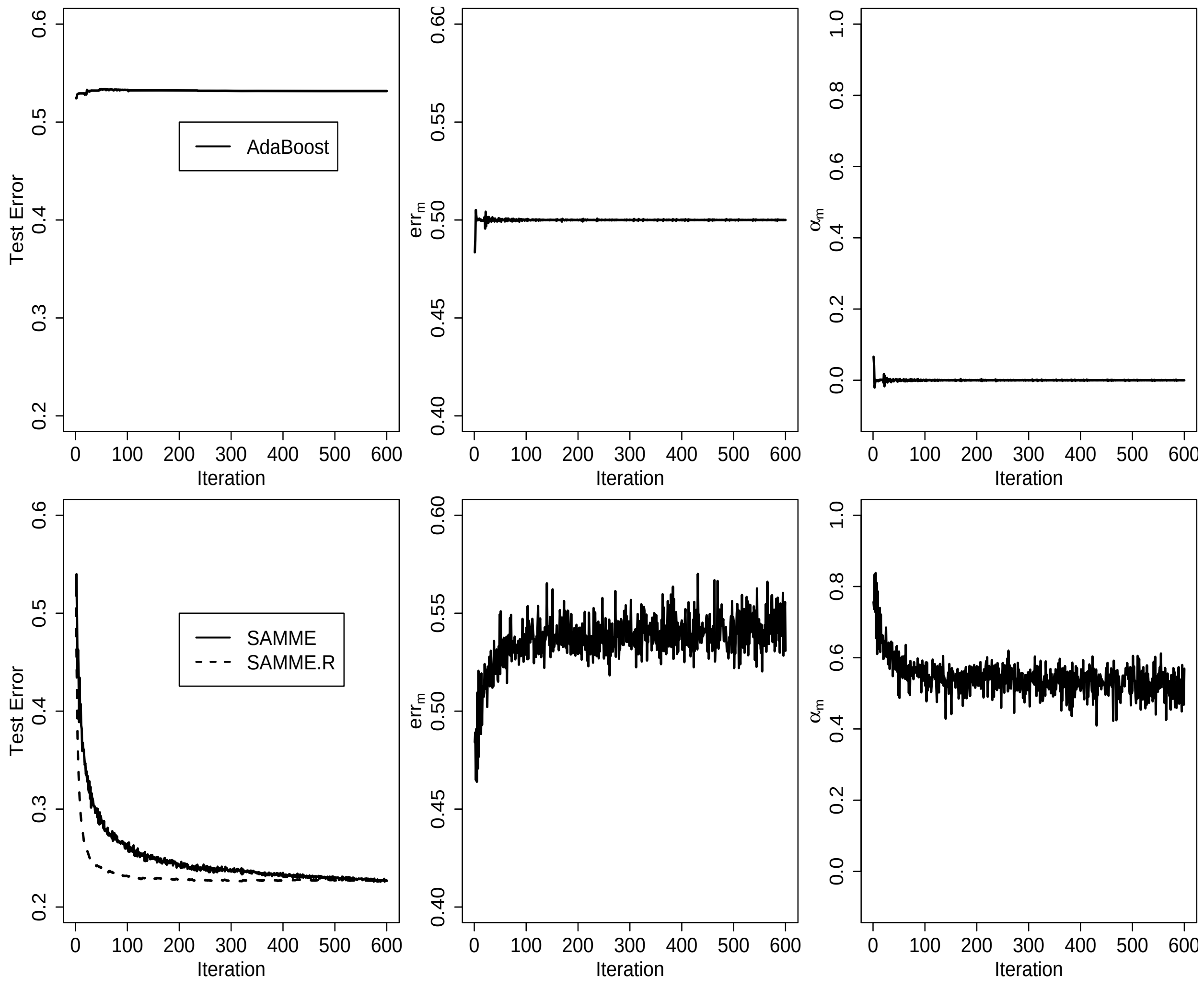

Figure 1: Comparison of AdaBoost and the new algorithm SAMME on a simple three-class simulation example. The training sample size is 3000, and the testing sample size is 10000. Ten-terminal node trees are used as weak classifiers. The results are averaged over ten independently drawn training-test set combinations. The upper row is for AdaBoost and the lower row is for SAMME.

1.2 Multi-class AdaBoost

Before delving into technical details, we propose our new algorithm for multi-class boosting and compare it with AdaBoost. We refer to our algorithm as *SAMME* — Stagewise Additive Modeling using a Multi-class Exponential loss function — this choice of name will be clear in Section 2.

Given the same setup as that of AdaBoost, SAMME proceeds as follows:

Algorithm 2 *SAMME*

```
1. Initialize the observation weights $w_i = 1/n,\ i = 1, 2, \dots, n$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $T^{(m)}(\boldsymbol{x})$ to the training data using weights $w_i$.
    (b) Compute
        $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
    (c) Compute
        $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1). \quad (1)$$
    (d) Set
        $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, \dots, n.$$
    (e) Re-normalize $w_i$.
3. Output
    $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$
```

Notice Algorithm 2 (SAMME) is very similar to AdaBoost with a major difference in (1). Now in order for $\alpha^{(m)}$ to be positive, we only need $(1-err^{(m)}) > 1/K$, or the accuracy of each weak classifier be better than random guessing rather than 1/2. As a consequence, the new algorithm puts more weight on the misclassified data points in (2d) than AdaBoost, and the new algorithm also combines weak classifiers a little differently from AdaBoost, i.e. differ by $\log(K-1)\sum_{m=1}^{M}\mathbb{I}(T^{(m)}(\boldsymbol{x}) = k)$. It is worth noting that when $K = 2$, SAMME reduces to AdaBoost. As we will see in Section 2, the extra term $\log(K-1)$ in (1) is not artificial; it makes the new algorithm equivalent to fitting a forward stagewise additive model using a multi-class exponential loss function. The difference between AdaBoost and SAMME (when $K = 3$) is also illustrated in Figure 1. As we have seen (upper left panel), AdaBoost breaks after the $err^{(m)}$ goes above 1/2. However this is not the case for SAMME (lower row): although $err^{(m)}$ can be bigger than 1/2 (or equal to 1/2), the $\alpha^{(m)}$ is still positive; hence, the mis-classified training samples get more weights, and the test error keeps decreasing even after 600 iterations.

The rest of the paper is organized as follows: In Section 2 and 3, we give theoretical justification for our new algorithm SAMME. In Section 4, we present numerical results on both simulation and real-world data. Summary and discussion regarding the implications of the new algorithm are in Section 5.

2 Theoretical Justification

In this section, we are going to show that the extra term $\log(K-1)$ in (1) is not artificial; it makes Algorithm 2 equivalent to fitting a forward stagewise additive model using a multi-class exponential loss function. Friedman et al. (2000) developed a statistical perspective on the original two-class AdaBoost algorithm, which ultimately leads to viewing the two-class AdaBoost algorithm as forward stagewise additive modeling using the exponential loss function

$$L(y, f) = e^{-yf(\boldsymbol{x})},$$

where $y = (\mathbb{I}(c=1) - \mathbb{I}(c=2)) \in \{-1, 1\}$ in a two-class classification setting. It is not difficult to show that the population minimizer of this exponential loss function is one half of the logit transform

$$\begin{aligned} f^*(\boldsymbol{x}) &= \arg\min_{f(\boldsymbol{x})} L(y, f) \\ &= \frac{1}{2}\log\frac{\text{Prob}(c=1|\boldsymbol{x})}{\text{Prob}(c=2|\boldsymbol{x})}. \end{aligned}$$

The Bayes optimal classification rule then agrees with the sign of $f^*(\boldsymbol{x})$, i.e.

$$\arg\max_y \text{Prob}(Y = y|\boldsymbol{X} = \boldsymbol{x}) = \text{sign}(f^*(\boldsymbol{x})).$$

We note that besides Friedman et al. (2000), Breiman (1999) and Schapire & Singer (1999) also made connections between the original two-class AdaBoost algorithm and the exponential loss function. We acknowledge that these views have been influential in our thinking for this paper.

2.1 A new multi-class exponential loss function

In the multi-class classification setting, we can recode the output c with a K-dimensional vector $\boldsymbol{y}$, with all entries equal to $-\frac{1}{K-1}$ except a 1 in position k if $c = k$, i.e. $\boldsymbol{y} = (y_1, \ldots, y_K)^\mathsf{T}$, and:

$$y_k = \begin{cases} 1, & \text{if } c = k, \\ -\frac{1}{K-1}, & \text{if } c \neq k. \end{cases} \tag{2}$$

Lee, Lin & Wahba (2004) used the same coding for a version of the multi-class support vector machine.

A generalization of the exponential loss function to the multi-class case then follows naturally:

$$L(\boldsymbol{y}, \boldsymbol{f}) = \exp\left(-\frac{1}{K}(y_1 f_1 + \cdots + y_K f_K)\right) = \exp\left(-\frac{1}{K}\boldsymbol{y}^\mathsf{T}\boldsymbol{f}\right), \tag{3}$$

where f_k corresponds to class k, and its meaning will be clear shortly. Notice that we need some constraint on $\boldsymbol{f}$ in order for it to be estimable, otherwise adding any constant to all f_k's does not change the loss. We choose to use the symmetric constraint: $f_1 + \cdots + f_K = 0$, so when $K = 2$, this multi-class exponential loss function reduces to the two-class exponential loss.

Similar to the two-class case, it is of interest to investigate what this multi-class exponential loss function estimates. This can be answered by seeking its population minimizer. Specifically, we are interested in

$$\begin{aligned} \arg\min_{\boldsymbol{f}(\boldsymbol{x})} \quad & \text{E}_{\boldsymbol{Y}|\boldsymbol{x}} \exp\left(-\frac{1}{K}(Y_1 f_1 + \cdots + Y_K f_K)\right) \\ \text{subject to} \quad & f_1 + \cdots + f_K = 0. \end{aligned}$$

The Lagrange of this constrained optimization problem can be written as:

$$\exp\left(-\frac{f_1(\boldsymbol{x})}{K-1}\right)\mathrm{Prob}(c=1|\boldsymbol{x})+\cdots+\exp\left(-\frac{f_K(\boldsymbol{x})}{K-1}\right)\mathrm{Prob}(c=K|\boldsymbol{x})-\lambda\left(f_1(\boldsymbol{x})+\cdots+f_K(\boldsymbol{x})\right),$$

where λ is the Lagrange multiplier. Taking derivatives with respect to f_k and λ, we reach

$$\begin{aligned}
-\frac{1}{K-1}\exp\left(-\frac{f_1(\boldsymbol{x})}{K-1}\right)\mathrm{Prob}(c=1|\boldsymbol{x})-\lambda &= 0,\\
\vdots \quad & \quad \vdots\\
-\frac{1}{K-1}\exp\left(-\frac{f_K(\boldsymbol{x})}{K-1}\right)\mathrm{Prob}(c=K|\boldsymbol{x})-\lambda &= 0,\\
f_1(\boldsymbol{x})+\cdots+f_K(\boldsymbol{x}) &= 0.
\end{aligned}$$

Solving this set of equations, we obtain the population minimizer

$$f_k^*(\boldsymbol{x})=(K-1)\left(\log\mathrm{Prob}(c=k|\boldsymbol{x})-\frac{1}{K}\sum_{k'=1}^{K}\log\mathrm{Prob}(c=k'|\boldsymbol{x})\right),\quad k=1,\ldots,K. \tag{4}$$

Thus,

$$\arg\max_k f_k^*(\boldsymbol{x})=\arg\max_k \mathrm{Prob}(c=k|\boldsymbol{x}),$$

which is the Bayes optimal classification rule in terms of minimizing the misclassification error. This justifies the use of this multi-class exponential loss function. Equation (4) also provides a way to recover the class probability $\mathrm{Prob}(c=k|\boldsymbol{x})$ once $f_k^*(\boldsymbol{x})$'s are estimated, i.e.

$$\mathrm{Prob}(c=k|\boldsymbol{x})=\frac{e^{\frac{1}{K-1}f_k^*(\boldsymbol{x})}}{e^{\frac{1}{K-1}f_1^*(\boldsymbol{x})}+\cdots+c^{\frac{1}{K-1}f_K^*(\boldsymbol{x})}},\quad k=1,\ldots,K.$$

2.2 Forward stagewise additive modeling

We now show that Algorithm 2 is equivalent to forward stagewise additive modeling using the loss function (3). We start with the forward stagewise additive modeling using a general loss function $L(\cdot,\cdot)$, then apply it to the multi-class exponential loss function (3).

Given the training data, we wish to find $\boldsymbol{f}(\boldsymbol{x})=(f_1(\boldsymbol{x}),\ldots,f_K(\boldsymbol{x}))^\mathsf{T}$ such that

$$\min_{\boldsymbol{f}(\boldsymbol{x})} \quad \sum_{i=1}^{n} L(\boldsymbol{y}_i,\boldsymbol{f}(\boldsymbol{x}_i)) \tag{5}$$

$$\text{subject to} \quad f_1(\boldsymbol{x})+\cdots+f_K(\boldsymbol{x})=0. \tag{6}$$

We consider $\boldsymbol{f}(\boldsymbol{x})$ that has the following form:

$$\boldsymbol{f}(\boldsymbol{x})=\sum_{m=1}^{M}\beta^{(m)}\boldsymbol{g}^{(m)}(\boldsymbol{x}),$$

where $\beta^{(m)}\in\mathbb{R}$ are coefficients, and $\boldsymbol{g}^{(m)}(\boldsymbol{x})$ are basis functions. We require $\boldsymbol{g}(\boldsymbol{x})$ to satisfy the symmetric constraint:

$$g_1(\boldsymbol{x})+\cdots+g_K(\boldsymbol{x})=0.$$

For example, the $\boldsymbol{g}(\boldsymbol{x})$ that we consider in this paper takes value in one of the K possible K-dimensional vectors in (2); specifically, at a given $\boldsymbol{x}$, $\boldsymbol{g}(\boldsymbol{x})$ mapps $\boldsymbol{x}$ onto $\mathcal{Y}$:

$$\boldsymbol{g} : \boldsymbol{x} \in \mathbb{R}^p \to \mathcal{Y},$$

where $\mathcal{Y}$ is the set containing K K-dimensional vectors:

$$\mathcal{Y} = \left\{ \begin{array}{c} \left(1, -\frac{1}{K-1}, \ldots, -\frac{1}{K-1}\right)^{\mathsf{T}}, \\ \left(-\frac{1}{K-1}, 1, \ldots, -\frac{1}{K-1}\right)^{\mathsf{T}}, \\ \vdots \\ \left(-\frac{1}{K-1}, \ldots, -\frac{1}{K-1}, 1\right)^{\mathsf{T}} \end{array} \right\}. \tag{7}$$

Forward stagewise modeling approximates the solution to (5) – (6) by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added. Specifically, the algorithm starts with $\boldsymbol{f}^{(0)}(\boldsymbol{x}) = 0$, sequentially selecting new basis functions from a dictionary and adding them to the current fit:

Algorithm 3 *Forward stagewise additive modeling*

```
1. Initialize $\boldsymbol{f}^{(0)}(\boldsymbol{x}) = 0$.
2. For $m = 1$ to $M$:
    (a) Compute
        $(\beta^{(m)}, \boldsymbol{g}^{(m)}(\boldsymbol{x})) = \arg\min_{\beta,\boldsymbol{g}} \sum_{i=1}^{n} L(\boldsymbol{y}_i, \boldsymbol{f}^{(m-1)}(\boldsymbol{x}_i) + \beta \boldsymbol{g}(\boldsymbol{x}_i)).$
    (b) Set
        $\boldsymbol{f}^{(m)}(\boldsymbol{x}) = \boldsymbol{f}^{(m-1)}(\boldsymbol{x}) + \beta^{(m)} \boldsymbol{g}^{(m)}(\boldsymbol{x}).$
```

The crucial step in the above algorithm is (2a). Now, using the multi-class exponential loss function (3), one wants to find $\boldsymbol{g}^{(m)}(\boldsymbol{x})$ (and $\beta^{(m)}$) to solve:

$$(\beta^{(m)}, \boldsymbol{g}^{(m)}) = \arg\min_{\beta,\boldsymbol{g}} \sum_{i=1}^{n} \exp\left(-\frac{1}{K}\boldsymbol{y}_i^{\mathsf{T}}(\boldsymbol{f}^{(m-1)}(\boldsymbol{x}_i) + \beta \boldsymbol{g}(\boldsymbol{x}_i))\right) \tag{8}$$

$$= \arg\min_{\beta,\boldsymbol{g}} \sum_{i=1}^{n} w_i \exp\left(-\frac{1}{K}\beta \boldsymbol{y}_i^{\mathsf{T}} \boldsymbol{g}(\boldsymbol{x}_i)\right), \tag{9}$$

where $w_i = \exp\left(-\frac{1}{K}\boldsymbol{y}_i^{\mathsf{T}} \boldsymbol{f}^{(m-1)}(\boldsymbol{x}_i)\right)$ are the unnormalized observation weights.

Notice that every $\boldsymbol{g}(\boldsymbol{x})$ as in (7) has a one-to-one correspondence with a multi-class classifier $T(\boldsymbol{x})$ in the following way:

$$T(\boldsymbol{x}) = k, \quad \text{if} \quad g_k(\boldsymbol{x}) = 1, \tag{10}$$

and vice versa:

$$g_k(\boldsymbol{x}) = \left\{ \begin{array}{ll} 1, & \text{if } T(\boldsymbol{x}) = k, \\ -\frac{1}{K-1}, & \text{if } T(\boldsymbol{x}) \neq k. \end{array} \right. \tag{11}$$

Hence, solving for $\boldsymbol{g}^{(m)}(\boldsymbol{x})$ in (9) is equivalent to finding the multi-class classifier $T^{(m)}(\boldsymbol{x})$ that can generate $\boldsymbol{g}^{(m)}(\boldsymbol{x})$.

Lemma 1 *The solution to (9) is*

$$
\begin{aligned}
T^{(m)}(\boldsymbol{x}) &= \arg\min \sum_{i=1}^{n} w_i \mathbb{I}(c_i \neq T(\boldsymbol{x}_i)), && (12)\\
\beta^{(m)} &= \frac{(K-1)^2}{K}\left(\log\frac{1-err^{(m)}}{err^{(m)}} + \log(K-1)\right), && (13)
\end{aligned}
$$

where $err^{(m)}$ *is defined as*

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$

Proof First, for any fixed value of $\beta > 0$, using the definition (10), one can express the criterion in (9) as:

$$
\begin{aligned}
&\sum_{c_i = T(\boldsymbol{x}_i)} w_i e^{-\frac{\beta}{K-1}} + \sum_{c_i \neq T(\boldsymbol{x}_i)} w_i e^{\frac{\beta}{(K-1)^2}}\\
&= e^{-\frac{\beta}{K-1}} \sum_i w_i + \left(e^{\frac{\beta}{(K-1)^2}} - e^{-\frac{\beta}{K-1}}\right) \sum_i w_i \mathbb{I}(c_i \neq T(\boldsymbol{x}_i)). \qquad (14)
\end{aligned}
$$

Since only the last sum depends on the classifier $T(\boldsymbol{x})$, we get that (12) holds. Now plugging (12) into (9) and solving for β, we obtain (13) (note that (14) is a convex function of β). $\square$

The model is then updated

$$\boldsymbol{f}^{(m)}(\boldsymbol{x}) = \boldsymbol{f}^{(m-1)}(\boldsymbol{x}) + \beta^{(m)} \boldsymbol{g}^{(m)}(\boldsymbol{x}),$$

and the weights for the next iteration will be

$$w_i \leftarrow w_i \cdot \exp\left(-\frac{1}{K}\beta^{(m)} \boldsymbol{y}_i^{\mathsf{T}} \boldsymbol{g}^{(m)}(\boldsymbol{x}_i)\right).$$

This is equal to

$$w_i \cdot e^{-\frac{(K-1)^2}{K^2}\alpha^{(m)} \boldsymbol{y}_i^{\mathsf{T}} \boldsymbol{g}^{(m)}(\boldsymbol{x}_i)} = \begin{cases} w_i \cdot e^{-\frac{K-1}{K}\alpha^{(m)}}, & \text{if } c_i = T(\boldsymbol{x}_i),\\ w_i \cdot e^{\frac{1}{K}\alpha^{(m)}}, & \text{if } c_i \neq T(\boldsymbol{x}_i), \end{cases} \qquad (15)$$

where $\alpha^{(m)}$ is defined as in (1) with the extra term $\log(K-1)$, and the new weight (15) is equivalent to the weight updating scheme in Algorithm 2 (2d) after normalization.

It is also a simple task to check that $\arg\max_k \boldsymbol{f}^{(m)}(\boldsymbol{x}) = \arg\max_k (f_1^{(m)}(\boldsymbol{x}), \ldots, f_K^{(m)}(\boldsymbol{x}))^{\mathsf{T}}$ is equivalent to the output $C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k)$ in Algorithm 2. Hence, Algorithm 2 can be considered as forward stagewise additive modeling using the multi-class exponential loss function.

2.3 Computational cost

Suppose one uses a classification tree as the weak learner, which is the most popular choice, and the depth of each tree is fixed as d, then the computational cost for building each tree is $O(dpn\log(n))$, where p is the dimension of the input $\boldsymbol{x}$. The computational cost for our SAMME algorithm is then $O(dpn\log(n)M)$ since there are M iterations. As a comparison, the computational cost for either the one vs. rest scheme or the pairwise scheme (Friedman 2004) is $O(dpn\log(n)MK)$, which is K times larger than the computational cost of SAMME.

3 A Variant of the SAMME Algorithm

The SAMME algorithm expects the weak learner to deliver a classifier $T(\boldsymbol{x}) \in \{1, \ldots, K\}$. An alternative is to use real-valued confidence-rated predictions such as weighted probability estimates, to update the additive model, rather than the classifications themselves.

Section 2.2 is based on the empirical loss, this time we consider the population version of the loss, derive the update for the additive model using the weighted probability, and then apply it to data.

Suppose we have a current estimate $\boldsymbol{f}^{(m-1)}(\boldsymbol{x})$ and seek an improved estimate $\boldsymbol{f}^{(m-1)}(\boldsymbol{x})+\boldsymbol{h}(\boldsymbol{x})$ by minimizing the loss at each $\boldsymbol{x}$:

$$\begin{array}{rl} \min\limits_{\boldsymbol{h}(\boldsymbol{x})} & \mathrm{E}\left(\exp\left(-\frac{1}{K}\boldsymbol{Y}^{\mathsf{T}}(\boldsymbol{f}^{(m-1)}(\boldsymbol{x})+\boldsymbol{h}(\boldsymbol{x}))\right)|\boldsymbol{x}\right) \\ \text{subject to} & h_1(\boldsymbol{x})+\cdots+h_K(\boldsymbol{x})=0. \end{array}$$

The above expectation can be re-written as

$$\begin{aligned} & e^{-\frac{h_1(\boldsymbol{x})}{K-1}}\mathrm{E}\left(e^{-\frac{1}{K}\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{f}^{(m-1)}(\boldsymbol{x})}\mathbb{I}_{c=1}|\boldsymbol{x}\right)+\cdots+e^{-\frac{h_K(\boldsymbol{x})}{K-1}}\mathrm{E}\left(e^{-\frac{1}{K}\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{f}^{(m-1)}(\boldsymbol{x})}\mathbb{I}_{c=K}|\boldsymbol{x}\right) \\ = \; & e^{-\frac{h_1(\boldsymbol{x})}{K-1}}\mathrm{Prob}_w(c=1|\boldsymbol{x})+\cdots+e^{-\frac{h_K(\boldsymbol{x})}{K-1}}\mathrm{Prob}_w(c=K|\boldsymbol{x}), \end{aligned}$$

where $w(\boldsymbol{x},\boldsymbol{Y})=\exp(-\frac{1}{K}\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{f}^{(m-1)}(\boldsymbol{x}))$, and

$$\mathrm{Prob}_w(c=k|\boldsymbol{x})=\mathrm{E}\left(e^{-\frac{1}{K}\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{f}^{(m-1)}(\boldsymbol{x})}\mathbb{I}_{c=k}|\boldsymbol{x}\right).$$

Taking the symmetric constraint into consideration and optimizing the Lagrange will lead to the solution (details skipped)

$$h_k(\boldsymbol{x})=(K-1)\left(\log\mathrm{Prob}_w(c=k|\boldsymbol{x})-\frac{1}{K}\sum_{k'=1}^{K}\log\mathrm{Prob}_w(c=k'|\boldsymbol{x})\right).$$

The algorithm as presented would stop after one iteration. In practice, we can use approximations to the conditional expectation, such as probability estimates from decision trees, and hence many steps are required. It naturally leads to a variant of the SAMME algorithm, which we call SAMME.R (R for Real).

Algorithm 4 *SAMME.R*

```
1. Initialize the observation weights $w_i = 1/n,\ i = 1, 2, \ldots, n$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $T^{(m)}(\boldsymbol{x})$ to the training data using weights $w_i$.
    (b) Obtain the weighted class probability estimates
            $p_k^{(m)}(\boldsymbol{x}) = \mathrm{Prob}_w(c = k|\boldsymbol{x}),\ k = 1, \ldots, K.$
    (c) Set
            $h_k^{(m)}(\boldsymbol{x}) \leftarrow (K-1)\left(\log p_k^{(m)}(\boldsymbol{x}) - \frac{1}{K}\sum_{k'}\log p_{k'}^{(m)}(\boldsymbol{x})\right),\ k = 1, \ldots, K.$
```

(d) Set

$$w_i \leftarrow w_i \cdot \exp\left(-\frac{K-1}{K}\boldsymbol{y}_i^{\mathsf{T}} \log \boldsymbol{p}^{(m)}(\boldsymbol{x}_i)\right),\ i = 1, \ldots, n.$$

(e) Re-normalize w_i.

3. Output

$$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} h_k^{(m)}(\boldsymbol{x}).$$

The lower row of Figure 1 (left panel) shows the result of SAMME.R on the simple three-class simulation example. As we can see, for this particular simulation example, SAMME.R converges more quickly than SAMME and also performs slightly better than SAMME.

4 Numerical Results

In this section, we use both simulation data and real-world data to demonstrate our algorithms.

For comparison, a single decision tree (CART; Breiman et al. (1984)) and AdaBoost.MH (Schapire & Singer 1999) are also fit. We have chosen to compare with the AdaBoost.MH algorithm since it seemed to have dominated other proposals in empirical studies (Schapire & Singer 1999). The AdaBoost.MH algorithm converts the K-class problem into that of estimating a two-class classifier on a training set K times as large, with an additional feature defined by the set of class labels. It is essentially the same as the one vs. rest scheme (Friedman et al. 2000), hence the computational cost is $O(dpn\log(n)MK)$, which can be K times larger than the computational cost of SAMME.

We note that there have been many other multi-class extensions of the boosting idea, for example, the ECOC in Schapire (1997), the uniform approach in Allwein et al. (2000), the logit-boost in Friedman et al. (2000), and the MART in Friedman (2001). We do not intend to make a full comparison with all these approaches. We would like to emphasize that the purpose of our numerical experiments is not to argue that SAMME is the ultimate multi-class classification tool, but rather to illustrate it is a sensible algorithm, and it is the natural extension of the AdaBoost algorithm to the multi-class case.

4.1 Simulation

We mimic a popular simulation example found in Breiman et al. (1984). This is a three-class problem with twenty one variables, and it is considered to be a difficult pattern recognition problem with Bayes error equal to 0.140. The predictors are defined by

$$x_j = \begin{cases} u \cdot v_1(j) + (1-u) \cdot v_2(j) + \epsilon_j, & \text{Class 1,} \\ u \cdot v_1(j) + (1-u) \cdot v_3(j) + \epsilon_j, & \text{Class 2,} \\ u \cdot v_2(j) + (1-u) \cdot v_3(j) + \epsilon_j, & \text{Class 3,} \end{cases} \tag{16}$$

where $j = 1, \ldots, 21$, u is uniform on $(0,1)$, ϵ_j are standard normal variables, and the v_ℓ are the shifted triangular waveforms: $v_1(j) = \max(6 - |j - 11|, 0), v_2(j) = v_1(j-4)$ and $v_3(j) = v_1(j+4)$. Figure 2 shows some example waveforms from each class with $\epsilon_j = 0$.

The training sample size is 300 so that approximately 100 training observations are in each class. We use the classification tree as the weak classifier for SAMME and SAMME.R. The trees are built using a greedy, top-down recursive partitioning strategy, and we restrict all trees within

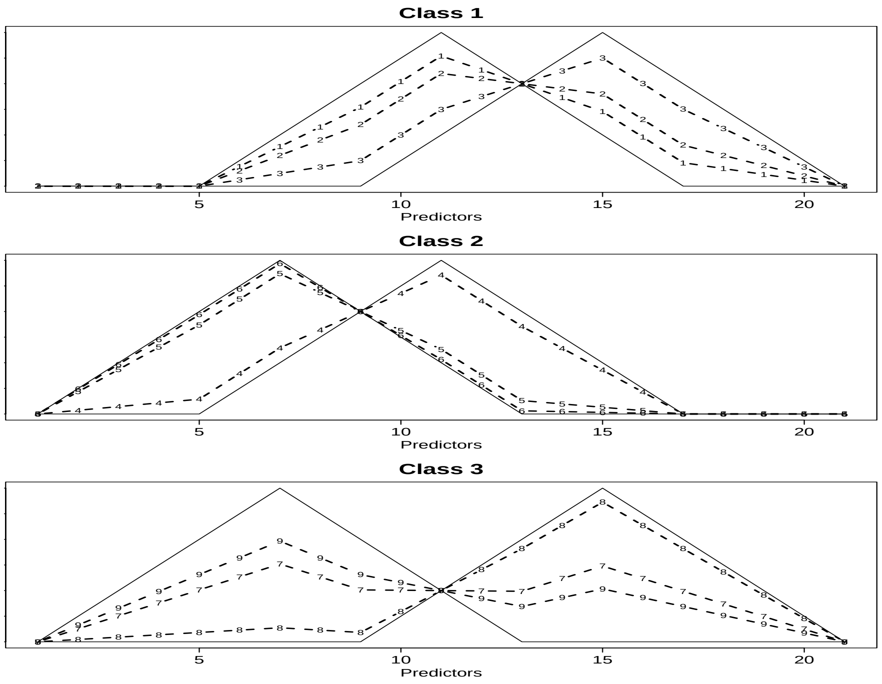

Figure 2: Some examples of the waveforms generated from model (16) before the Gaussian noise is added. The horizontal axis corresponds to the 21 predictor variables, and the vertical axis corresponds to the values of 21 predictor variables.

each method to have the same number of terminal nodes. This number is chosen via five-fold cross-validation. We use an independent test sample of size 5000 to estimate the error rate. Averaged results over ten such independently drawn training-test set combinations are shown in Figure 3 and Table 1.

As we can see, for this particular simulation example, SAMME performs slightly better than the AdaBoost.MH algorithm. A paired t-test across the ten independent comparisons indicates a significant difference with p-value around 0.003. We can also see that the SAMME.R algorithm performs closely to the SAMME algorithm.

Table 1: Test error rates % of different methods on the `waveform` data. The results are averaged over ten independently drawn datasets. For comparison, a single decision tree is also fit.

	Iterations		
Method	**200**	**400**	**600**
`Waveform`	CART error = 28.4 (1.8)		
Ada.MH	17.1 (0.6)	17.0 (0.5)	17.0 (0.6)
SAMME	16.7 (0.8)	16.6 (0.7)	16.6 (0.6)
SAMME.R	16.8 (0.3)	16.6 (0.4)	16.6 (0.4)

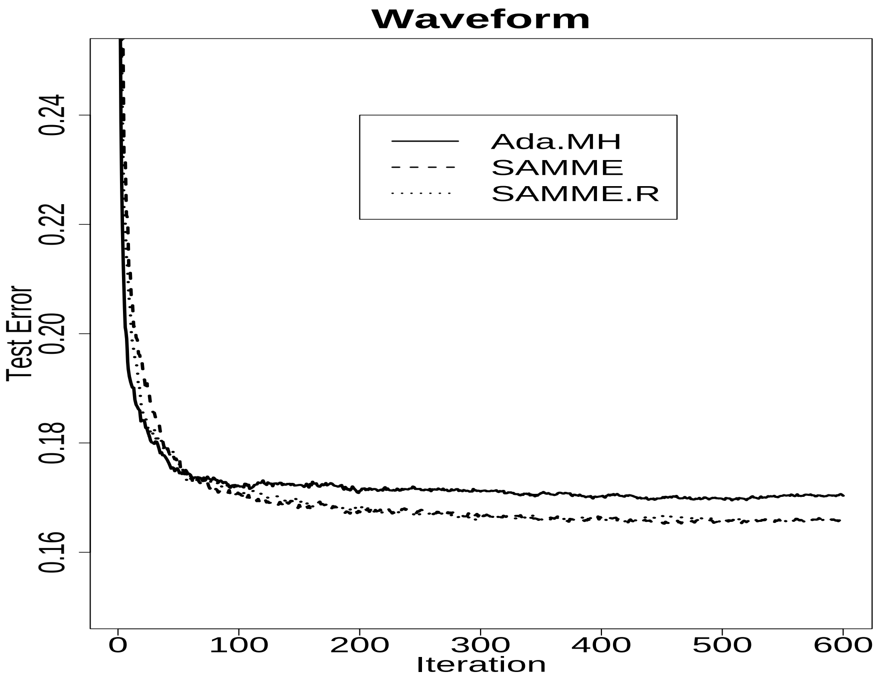

Figure 3: Test errors for SAMME, SAMME.R and AdaBoost.MH on the `waveform` simulation example. The training sample size is 300, and the testing sample size is 5000. The results are averages over ten independently drawn training-test set combinations.

4.2 Real data

In this section, we show the results of running SAMME and SAMME.R on a collection of datasets from the UC-Irvine machine learning archive (Merz & Murphy 1998). Seven datasets were used: `Letter`, `Nursery`, `Pendigits`, `Satimage`, `Segmentation`, `Thyroid` and `Vowel`. These datasets come with pre-specified training and testing sets, and are summarized in Table 2. They cover a wide range of scenarios: the number of classes ranges from 3 to 26, and the size of the training data ranges from 210 to $16,000$ data points. The types of input variables include both numerical and categorical, for example, in the `Nursery` dataset, all input variables are categorical variables. We used a classification tree as the weak classifier in each case. Again, the trees were built using a greedy, top-down recursive partitioning strategy. We restricted all trees within each method to have the same number of terminal nodes, and this number was chosen via five-fold cross-validation.

Figure 4 compares SAMME and AdaBoost.MH. Since the SAMME.R algorithm performs closely to the SAMME algorithm, for exposition clarity, the test error curves for the SAMME.R algorithm are not shown. The test error rates are summarized in Table 3. The standard errors are approximated by $\sqrt{\text{te.err} \cdot (1 - \text{te.err})/\text{n.te}}$, where `te.err` is the test error, and `n.te` is the size of the testing data.

The most interesting result is on the `Vowel` dataset. This is a difficult classification problem, and the best methods achieve around 40% errors on the test data (Hastie, Tibshirani & Friedman 2001). The data was collected by Deterding (1989), who recorded examples of the eleven steady state vowels of English spoken by fifteen speakers for a speaker normalization study. The International Phonetic Association (IPA) symbols that represent the vowels and the words in which the eleven vowel sounds were recorded are given in the following table:

Table 2: Summary of seven benchmark datasets

Dataset	#Train	#Test	#Variables	#Classes
Letter	16000	4000	16	26
Nursery	8840	3790	8	3
Pendigits	7494	3498	16	10
Satimage	4435	2000	36	6
Segmentation	210	2100	19	7
Thyroid	3772	3428	21	3
Vowel	528	462	10	11

vowel	word	vowel	word	vowel	word	vowel	word
i:	heed	O	hod	I	hid	C:	hoard
E	head	U	hood	A	had	u:	who'd
a:	hard	3:	heard	Y	hud		

Four male and four female speakers were used to train the classifier, and another four male and three female speakers were used for testing the performance. Each speaker yielded six frames of speech from eleven vowels. This gave 528 frames from the eight speakers used as the training data and 462 frames from the seven speakers used as the testing data. The ten predictors are derived from the digitized speech in a rather complicated way, but standard in the speech recognition world. As we can see from Figure 4 and Table 3, for this particular dataset, the SAMME algorithm performs almost 15% better than the AdaBoost.MH algorithm.

For other datasets, the SAMME algorithm performs slightly better than the AdaBoost.MH algorithm on `Letter`, `Pendigits`, and `Thyroid`, while slightly worse on `Segmentation`. In the `Segmentation` data, there are only 210 training data points, so the difference might be just due to randomness. It is also worth noting that for the `Nursery` data, both the SAMME algorithm and the AdaBoost.MH algorithm are able to reduce the test error to zero, while a single decision tree has about 0.8% test error rate. Overall, we are comfortable to say that the performance of SAMME is comparable with that of the AdaBoost.MH.

For the purpose of further investigation, we also merged the training and the testing sets, and randomly split them into new training and testing sets. We then applied different methods on these new training and testing sets. The procedure was repeated ten times. The means of the test errors and the corresponding standard errors (in parentheses) are summarized in Table 4 and Figure 5. Again, the performance of SAMME is comparable with that of the AdaBoost.MH. We can see that for the `Vowel` data, the SAMME algorithm performs about 10% better than the AdaBoost.MH algorithm. A paired t-test across the ten independent comparisons indicates a significant difference with p-value around 0.001. For the `Segmentation` data, the test error rates are reversed: SAMME performs slightly better than AdaBoost.MH for this dataset. It is also interesting to note that the test error rate for the `Pendigits` data drops dramatically from 3% to about 0.6%. It turns out that in the original training-test split, the training digits were written by 30 writers, while the testing digits were written by 14 different writers. Unfortunately, who wrote which digits in the original data was not recorded. Therefore, in the re-split data, the 44 writers were mixed in both the training and testing data, which caused the drop in the test error rate.

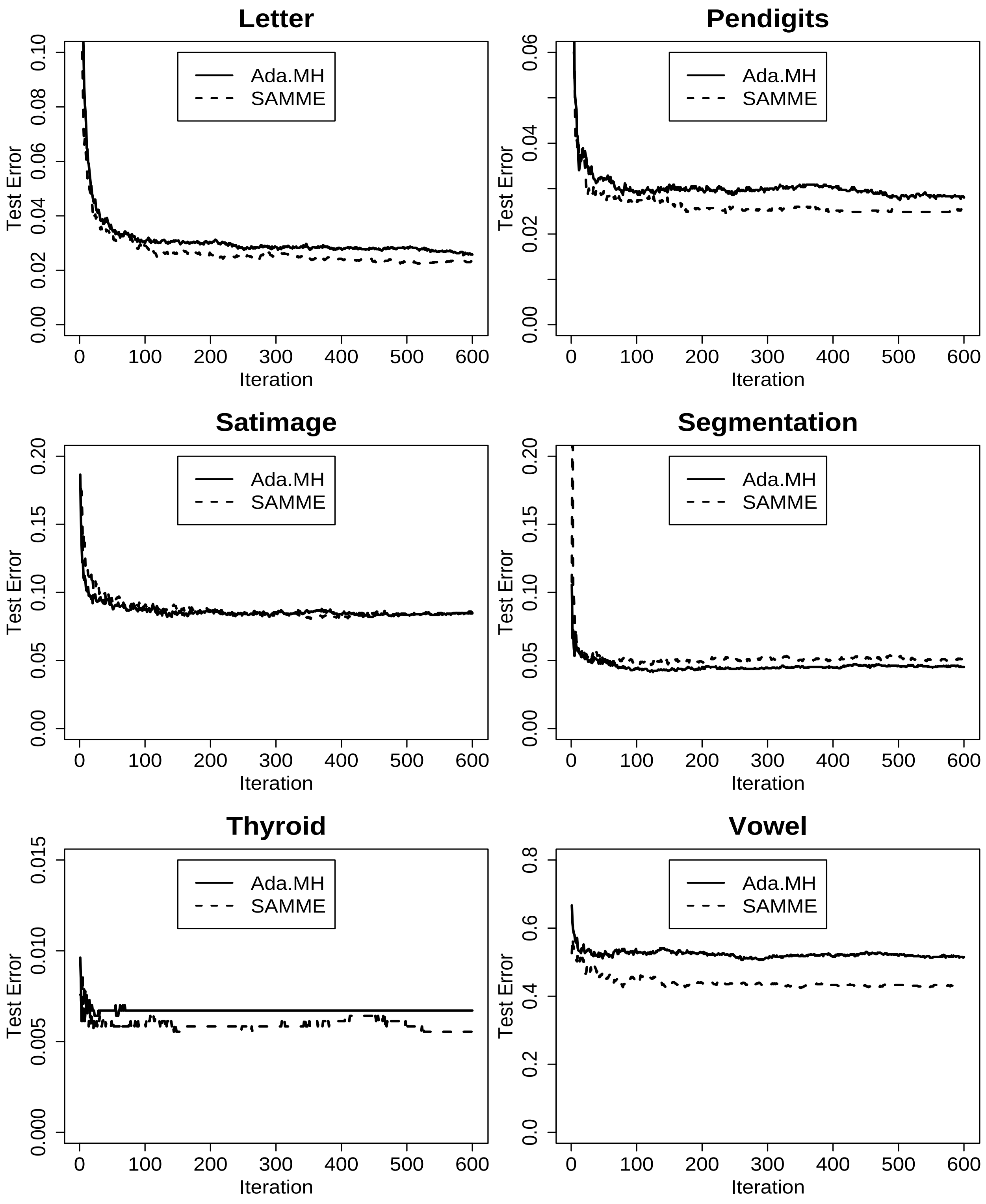

Figure 4: Test errors for SAMME and AdaBoost.MH on six benchmark datasets. These datasets come with pre-specified training and testing splits, and they are summarized in Table 2. The results for the `Nursery` data are not shown for the test error rates are reduced to zero for both methods. For exposition clarity, the results for SAMME.R are not shown because they are very close to SAMME.

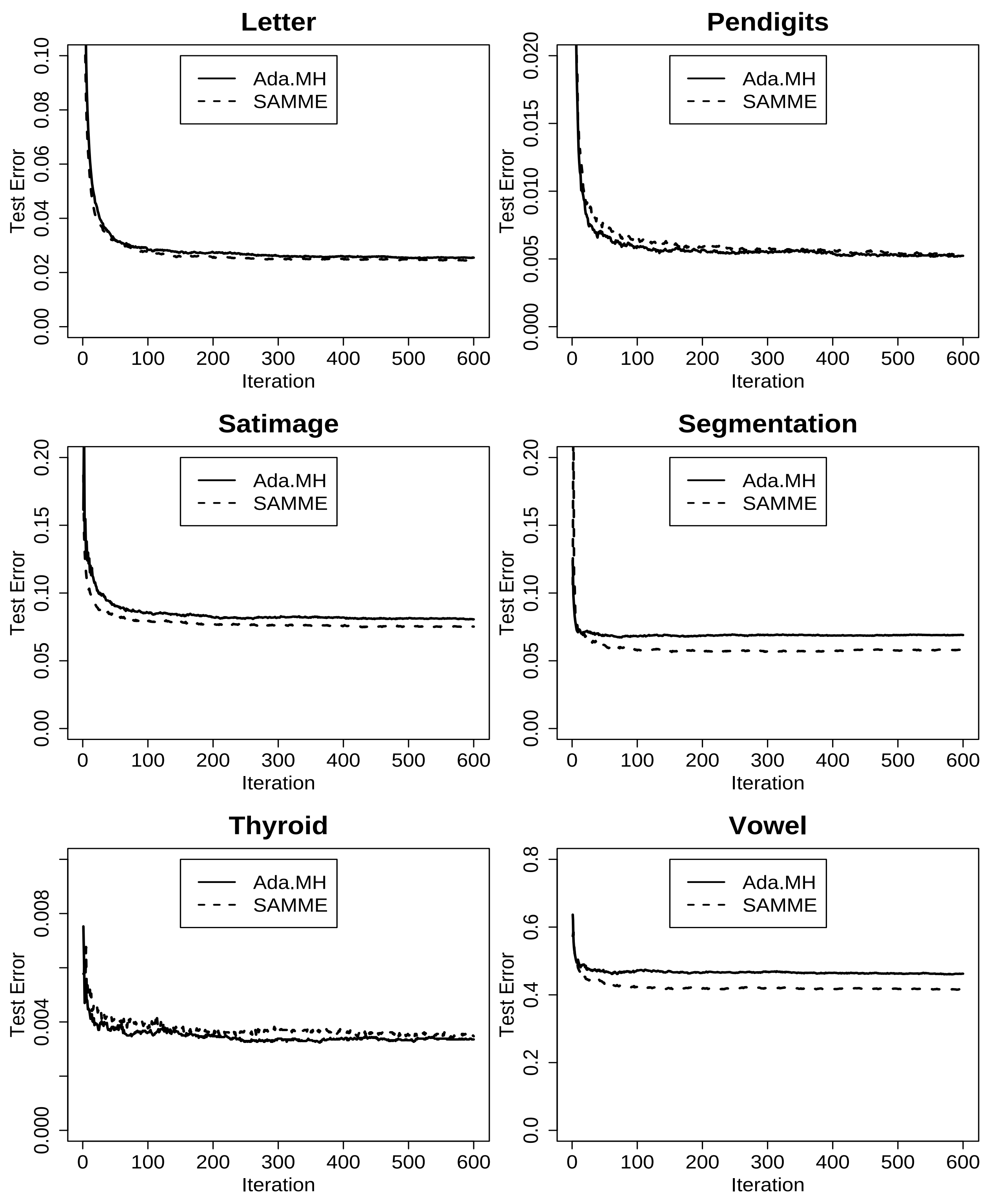

Figure 5: Test error for SAMME and AdaBoost.MH on six benchmark datasets. The results are averages over ten random training-test splits. The results for the `Nursery` data are not shown for the test error rates are reduced to zero for both methods. For exposition clarity, the results for SAMME.R are not shown because they are very close to SAMME.

5 Discussion

The statistical view of boosting, as illustrated in Friedman et al. (2000), shows that the two-class AdaBoost builds an additive model to approximate the two-class Bayes rule. Following the same statistical principle, we have derived SAMME, the natural and clean multi-class extension of the two-class AdaBoost algorithm, and we have shown that

- SAMME adaptively implements the multi-class Bayes rule by fitting a forward stagewise additive model for multi-class problems;
- SAMME follows closely to the philosophy of boosting, i.e. adaptively combining weak *classifiers* (rather than *regressors* as in logit-boost (Friedman et al. 2000) and MART (Friedman 2001)) into a powerful one;
- at each stage, SAMME returns only *one* weighted classifier (rather than K), and the weak classifier only needs to be better than K-class random guessing;
- SAMME shares the same simple modular structure of AdaBoost.

We note that although AdaBoost was proposed to solve both the two-class and the multi-class problems (Freund & Schapire 1997), it sometimes can fail in the multi-class case. AdaBoost.MH is an approach to convert the K-class problem into K two-class problems. As we have seen, AdaBoost.MH in general performs very well on both simulation and real-world data. SAMME's performance is comparable with that of the AdaBoost.MH, and sometimes slightly better. However, we would like to emphasize that our goal is not to argue that SAMME is the ultimate multi-class classification tool, but rather to illustrate it is the natural extension of the AdaBoost algorithm to the multi-class case. We believe our numerical and theoretical results make SAMME an interesting and useful addition to the toolbox of multi-class methods. The SAMME algorithm has been implemented in the R computing environment, and will be publicly available from the first author's website.

The multi-class exponential loss function (3) is a natural generalization of the two-class exponential loss function. Using the symmetry in $\boldsymbol{y}$ and $\boldsymbol{f}$, we can further simplify (3) as

$$\exp\left(-\frac{1}{K-1}\sum_{k=1}^{K}\mathbb{I}(c=k)f_k\right) = \exp\left(-\frac{1}{K-1}f_c\right).$$

This naturally leads us to consider a general multi-class loss function that has the form:

$$\sum_{k=1}^{K}\mathbb{I}(c=k)\phi(f_k) \quad \text{or equivalently} \quad \phi(f_c).$$

Hence, there are several interesting directions where our work can be extended: (1) For this general loss function to be sensible, we would like to find conditions on ϕ such that the classification rule given by the population minimizer agrees with the Bayes optimal rule. Lin (2004) studied a similar problem for the two-class classification case, and proposed very general conditions (Theorem 3.1). Along the lines of Lin (2004), we plan to design more multi-class loss functions that can be used in practice. (2) The principal attraction of the multi-class exponential loss function in the context of additive modeling is computational; it leads to our simple modular reweighting SAMME algorithm. We intend to develop algorithms for other members of the multi-class loss functions family as well.

One example is the multi-class logit loss, which can be more robust to outliers and misspecified data. However, other multi-class loss functions may not lead to a simple closed form reweighting scheme. The computational trick used in Friedman (2001) and Bühlmann & Yu (2003) should prove useful here.

Acknowledgments

The authors would like to dedicate this work to the memory of Leo Breiman, who passed away while we were finalizing this manuscript. Leo Breiman has made tremendous contributions to the study of statistics and machine learning. His work has greatly influenced us.

Zhu is partially supported by grant DMS-0505432 from the National Science Foundation. Hastie is partially supported by grant DMS-0204162 from the National Science Foundation.

References

Allwein, E., Schapire, R. & Singer, Y. (2000), 'Reducing multiclass to binary: a unifying approach for margin classifiers', *Journal of Machine Learning Research* **1**, 113–141.

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

Breiman, L. (1999), 'Prediction games and arcing algorithms', *Neural Computation* **7**, 1493–1517.

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

Bühlmann, P. & Yu, B. (2003), 'Boosting with the l_2 loss: regression and classification', *Journal of the American Statistical Association* **98**(462).

Deterding, D. (1989), *Speaker Normalisation for Automatic Speech Recognition*, University of Cambridge. Ph.D. thesis.

Freund, Y. & Schapire, R. (1997), 'A decision theoretic generalization of on-line learning and an application to boosting', *Journal of Computer and System Sciences* **55**(1), 119–139.

Friedman, J. (2001), 'Greedy function approximation: a gradient boosting machine', *Annals of Statistics* **29**(5).

Friedman, J. (2004), 'Another approach to polychotomous classification', *Machine Learning* . To appear.

Friedman, J., Hastie, T. & Tibshirani, R. (2000), 'Additive logistic regression: a statistical view of boosting', *Annals of Statistics* **28**, 337–407.

Hastie, T., Tibshirani, R. & Friedman, J. (2001), *The Elements of Statistical Learning*, Springer-Verlag, New York.

Lee, Y., Lin, Y. & Wahba, G. (2004), 'Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data', *Journal of the American Statistical Association* **99**, 67–81.

Lin, Y. (2004), ‘A note on margin-based loss functions in classification’, *Statistics and Probability Letters* **68**(1), 73–82.

Merz, C. & Murphy, P. (1998), ‘UCI repository of machine learning databases’, *www.ics.uci.edu/˜mlearn/MLRepository.html* .

Schapire, R. (1997), Using output codes to boost multiclass learning problems, *in* ‘Proceedings of the Fourteenth International Conference on Machine Learning’, Morgan Kauffman.

Schapire, R. & Singer, Y. (1999), ‘Improved boosting algorithms using confidence-rated prediction’, *Machine Learning* **37**(3), 297–336.

Table 3: Test error rates % on seven benchmark real datasets. The datasets come with pre-specified training and testing splits. The standard errors (in parentheses) are approximated by $\sqrt{\text{te.err} \cdot (1 - \text{te.err})/\text{n.te}}$, where `te.err` is the test error, and `n.te` is the size of the testing data. For comparison, a single decision tree was also fit, and the tree size was determined by five-fold cross-validation.

	Iterations		
Method	**200**	**400**	**600**
`Letter`	CART error = 13.5 (0.5)		
Ada.MH	3.0 (0.3)	2.8 (0.3)	2.6 (0.3)
SAMME	2.6 (0.3)	2.4 (0.2)	2.3 (0.2)
SAMME.R	2.6 (0.3)	2.5 (0.2)	2.4 (0.2)
`Nursery`	CART error = 0.79 (0.14)		
Ada.MH	0	0	0
SAMME	0	0	0
SAMME.R	0	0	0
`Pendigits`	CART error = 8.3 (0.5)		
Ada.MH	3.0 (0.3)	3.0 (0.3)	2.8 (0.3)
SAMME	2.5 (0.3)	2.5 (0.3)	2.5 (0.3)
SAMME.R	2.8 (0.3)	2.8 (0.3)	2.7 (0.3)
`Satimage`	CART error = 13.8 (0.8)		
Ada.MH	8.7 (0.6)	8.4 (0.6)	8.5 (0.6)
SAMME	8.6 (0.6)	8.2 (0.6)	8.5 (0.6)
SAMME.R	8.8 (0.6)	8.8 (0.6)	8.6 (0.6)
`Segmentation`	CART error = 9.3 (0.6)		
Ada.MH	4.5 (0.5)	4.5 (0.5)	4.5 (0.5)
SAMME	4.9 (0.5)	5.0 (0.5)	5.1 (0.5)
SAMME.R	5.3 (0.5)	4.9 (0.5)	5.0 (0.5)
`Thyroid`	CART error = 0.64 (0.14)		
Ada.MH	0.67 (0.14)	0.67 (0.14)	0.67 (0.14)
SAMME	0.58 (0.13)	0.61 (0.13)	0.58 (0.13)
SAMME.R	0.61 (0.13)	0.70 (0.14)	0.67 (0.14)
`Vowel`	CART error = 53.0 (2.3)		
Ada.MH	52.8 (2.3)	51.5 (2.3)	51.5 (2.3)
SAMME	43.9 (2.3)	43.3 (2.3)	43.3 (2.3)
SAMME.R	43.9 (2.3)	46.3 (2.3)	46.7 (2.3)

Table 4: Test error rates % on seven benchmark real datasets. The results are averages over ten random training-test splits. For comparison, a single decision tree was also fit, and the tree size was determined by five-fold cross-validation.

	Iterations		
Method	**200**	**400**	**600**
`Letter`	CART error = 13.5 (0.7)		
Ada.MH	2.8 (0.3)	2.6 (0.3)	2.5 (0.3)
SAMME	2.5 (0.3)	2.5 (0.2)	2.4 (0.2)
SAMME.R	2.5 (0.3)	2.5 (0.2)	2.4 (0.2)
`Nursery`	CART error = 0.68 (0.21)		
Ada.MH	0	0	0
SAMME	0	0	0
SAMME.R	0	0	0
`Pendigits`	CART error = 4.6 (0.3)		
Ada.MH	0.57 (0.10)	0.54 (0.13)	0.52 (0.12)
SAMME	0.59 (0.12)	0.57 (0.11)	0.54 (0.11)
SAMME.R	0.61 (0.12)	0.57 (0.15)	0.57 (0.15)
`Satimage`	CART error = 13.3 (0.8)		
Ada.MH	8.2 (0.5)	8.2 (0.3)	8.1 (0.3)
SAMME	7.7 (0.5)	7.6 (0.4)	7.5 (0.3)
SAMME.R	8.0 (0.3)	7.9 (0.3)	7.8 (0.3)
`Segmentation`	CART error = 10.9 (2.1)		
Ada.MH	6.9 (1.2)	6.9 (1.2)	6.9 (1.3)
SAMME	5.8 (0.8)	5.7 (0.9)	5.8 (0.9)
SAMME.R	6.1 (0.8)	6.1 (0.8)	6.1 (0.9)
`Thyroid`	CART error = 0.46 (0.05)		
Ada.MH	0.35 (0.12)	0.34 (0.12)	0.34 (0.12)
SAMME	0.37 (0.10)	0.36 (0.10)	0.35 (0.11)
SAMME.R	0.38 (0.07)	0.34 (0.09)	0.34 (0.10)
`Vowel`	CART error = 58.1 (4.2)		
Ada.MH	46.5 (5.5)	46.5 (4.9)	46.2 (4.9)
SAMME	41.8 (3.4)	41.6 (3.3)	41.8 (3.6)
SAMME.R	43.2 (4.2)	43.8 (3.8)	43.0 (3.7)