Statistics and Its Interface Volume 2 (2009) 349–360

Multi-class AdaBoost*

Ji Zhu†‡, Hui Zou§, Saharon Rosset and Trevor Hastie¶

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 develop a new algorithm that directly extends the AdaBoost algorithm to the multi-class case without reducing it to multiple two-class problems. We show that the proposed multi-class AdaBoost algorithm is equivalent to a forward stagewise additive modeling algorithm that minimizes a novel exponential loss for multi-class classification. Furthermore, we show that the exponential loss is a member of a class of Fisher-consistent loss functions for multi-class classification. As shown in the paper, the new algorithm is extremely easy to implement and is highly competitive in terms of misclassification error rate.

AMS 2000 subject classifications: Primary 62H30.
Keywords and phrases: boosting, exponential loss, multi-class classification, stagewise modeling.

1. INTRODUCTION

Boosting has been a very successful technique for solving the two-class classification problem. It was first introduced by [8], 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. [8], [19], and [21]. The ways to extend AdaBoost from two-class to multi-class depend on the interpretation or view of the success of AdaBoost in binary classification, which still remains controversial. Much theoretical work on AdaBoost has been based on the margin analysis, for example, see [20] and [13]. Another view on boosting, which is popular in the statistical community, regards AdaBoost as a functional gradient descent algorithm [6, 11, 17]. In [11], AdaBoost has been shown to be equivalent to a forward stagewise additive modeling algorithm that minimizes the exponential loss. [11] suggested that the success of AdaBoost can be understood by the fact that the population minimizer of exponential loss is one-half of the log-odds. Based on this statistical explanation, [11] derived a multi-class logit-boost algorithm.

The multi-class boosting algorithm by [11] looks very different from AdaBoost, hence it is not clear if the statistical view of AdaBoost still works in the multi-class case. To resolve this issue, we think it is desirable to derive an AdaBoost-like multi-class boosting algorithm by using the exact same statistical explanation of AdaBoost. 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. Surprisingly, the new algorithm is almost identical to AdaBoost but with a simple yet critical modification, and 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. We show that the proposed multi-class AdaBoost algorithm is equivalent to a forward stagewise additive modeling algorithm that minimizes a novel exponential loss for multi-class classification. Furthermore, we show that the exponential loss is a member of a class of Fisher-consistent loss functions for multi-class classification. Combined with forward stagewise additive modeling, these loss functions can be used to derive various multi-class boosting algorithms. We believe this paper complements [11].

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 [8]. Suppose we are given a set of training data $(\boldsymbol{x}_1, c_1), \dots, (\boldsymbol{x}_n, c_n)$, where the input (prediction 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. $\{1, 2, \dots, K\}$. K is the number of classes. Usually it is assumed that the training data are independently and identically distributed samples from an unknown probability distribution $\text{Prob}(X, C)$. 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 $\{1, \dots, K\}$. Under the 0/1 loss, the misclassification error rate of a classifier $C(\boldsymbol{x})$ is given by $1 - \sum_{k=1}^{K} \text{E}_X \left[\mathbb{I}_{C(X)=k}\text{Prob}(C = k|X)\right]$. 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 $1 - \text{E}_X \max_k \text{Prob}(C = k|X)$. This classifier is

[*] We thank the AE and a referee for their helpful comments and suggestions which greatly improved our paper.
[†] Corresponding author.
[‡] Zhu was partially supported by NSF grant DMS-0705532.
[§] Zou was partially supported by NSF grant DMS-0706733.
[¶] Hastie was partially supported by NSF grant DMS-0204162.

known as the *Bayes classifier*, and its error rate is the *Bayes error rate*.

The AdaBoost algorithm is an iterative procedure that tries to approximate the Bayes classifier $C^*(\boldsymbol{x})$ by combining many *weak* classifiers. Starting with the unweighted training sample, the AdaBoost builds a classifier, for example a classification tree [5], 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 [8]*

```
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
       $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$
   (d) Set
       $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$
       for $i = 1, 2, \dots, n$.
   (e) Re-normalize $w_i$.
3. Output
   $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$
```

When applied to two-class classification problems, AdaBoost has been proved to be extremely successful in producing accurate classifiers. In fact, [1] 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 [8]. Note that the theory of [8] assumes that the error of each weak classifier $err^{(m)}$ is less than 1/2 (or equivalently $\alpha^{(m)} > 0$), with respect to the distribution on which it was trained. This assumption is easily satisfied for two-class classification problems, because the error rate of random guessing is 1/2. However, it is much harder to achieve in the multi-class case, where the random guessing error rate is $(K-1)/K$. As pointed out by the inventors of AdaBoost, the main disadvantage of AdaBoost is that it is unable to handle weak learners with an error rate greater than 1/2. As a result, AdaBoost may easily fail in the multi-class case. 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. As we can see (upper left panel), the test error of AdaBoost actually starts to increase after 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 ($\alpha^{(m)} < 0$), 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.

This illustrative example may help explain why AdaBoost is never used for multi-class problems. Instead, for multi-class classification problems, [21] proposed the AdaBoost.MH algorithm which combines AdaBoost and the one-versus-all strategy. There are also several other multi-class extensions of the boosting idea, for example, the ECOC in [19] and the logit-boost in [11].

1.2 Multi-class AdaBoost

We introduce a new multi-class generalization of AdaBoost for multi-class classification. 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:

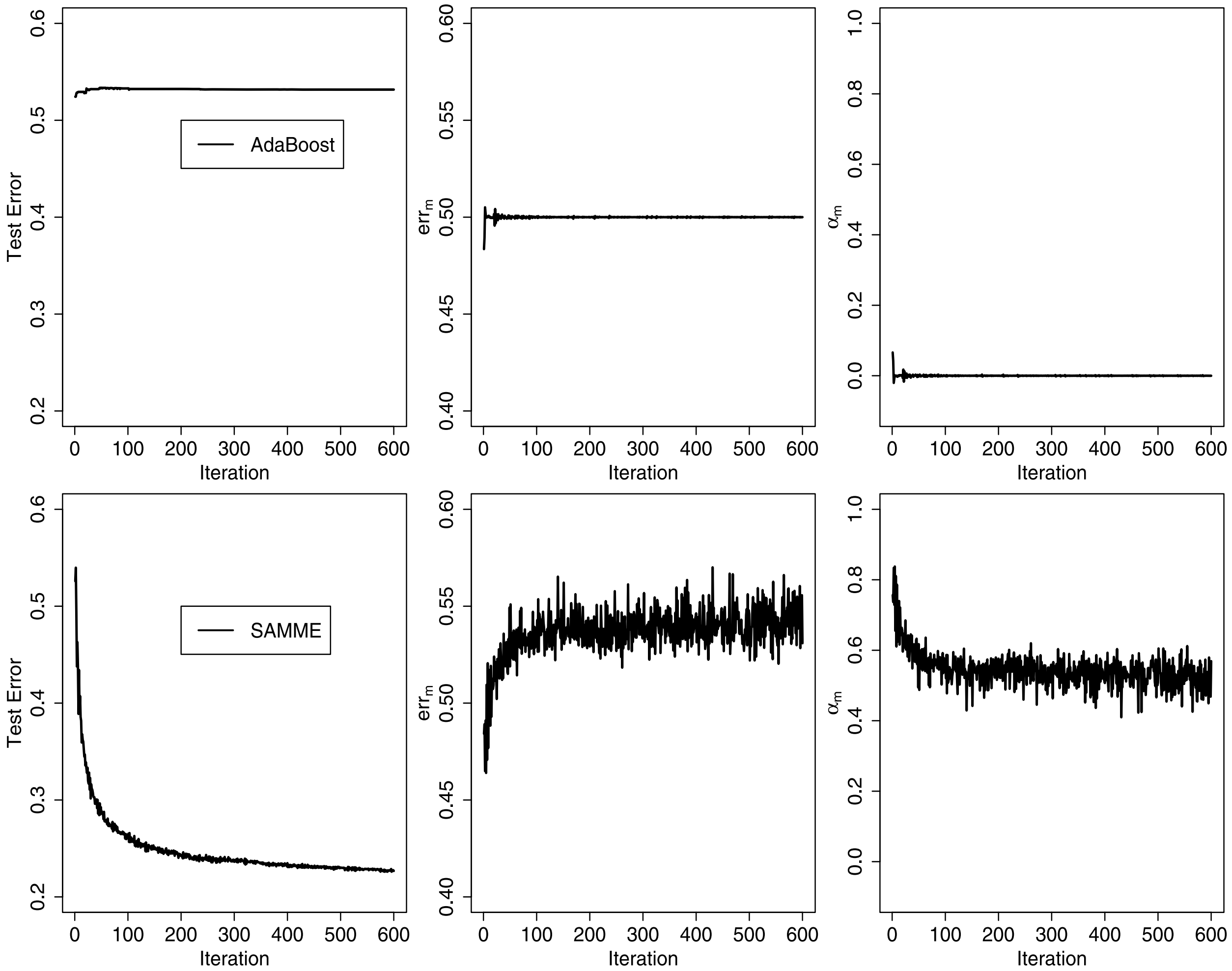

Figure 1. Comparison of AdaBoost and the new algorithm SAMME on a simple three-class simulation example. The training sample size is 3000, and the testing sample size is 10000. Ten-terminal node trees are used as weak classifiers. The upper row is for AdaBoost and the lower row is for SAMME.

Algorithm 2. *SAMME*

```
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
        (1)    $\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1).$
    (d) Set
        $w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$
        for $i = 1, \ldots, n$.
    (e) Re-normalize $w_i$.
3. Output
        $C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$
```

Note that Algorithm 2 (SAMME) shares the same simple modular structure of AdaBoost with a *simple but subtle* difference in (1), specifically, the extra term $\log(K-1)$. Obviously, when $K = 2$, SAMME reduces to AdaBoost. However, the term $\log(K-1)$ in (1) is critical in the multi-class case ($K > 2$). One immediate consequence is that now in order for $\alpha^{(m)}$ to be positive, we only need $(1-err^{(m)}) > 1/K$, or the accuracy of each weak classifier to be better than random guessing rather than 1/2. To appreciate its effect, we apply SAMME to the illustrative example in Section 1.1. As can be seen from Fig. 1, the test error of SAMME quickly decreases to a low value and keeps decreasing even after 600 iterations, which is exactly what we could expect from a successful boosting algorithm. In Section 2, we shall show that the term $\log(K-1)$ is not artificial, it follows naturally from the multi-class generalization of the exponential loss in the binary case.

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

2. STATISTICAL 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. Our arguments are in line with [11] who developed a statistical perspective on the original two-class AdaBoost algorithm, viewing the two-class AdaBoost algorithm as forward stagewise additive modeling using the exponential loss function

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

where $y = (\mathbb{I}(c=1) - \mathbb{I}(c=2)) \in \{-1,1\}$ in a two-class classification setting. A key argument is to show that the population minimizer of this exponential loss function is one half of the logit transform

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

Therefore, the Bayes optimal classification rule agrees with the sign of $f^*(\boldsymbol{x})$. [11] recast AdaBoost as a functional gradient descent algorithm to approximate $f^*(\boldsymbol{x})$. We note that besides [11], [2] and [21] 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 SAMME as forward stagewise additive modeling

We now show that Algorithm 2 is equivalent to forward stagewise additive modeling using a multi-class exponential loss function.

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. 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. $\boldsymbol{y} = (y_1,\ldots,y_K)^\intercal$, and:

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

[14] and [16] used the same coding for the multi-class support vector machine. Given the training data, we wish to find $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}),\ldots,f_K(\boldsymbol{x}))^\intercal$ such that

$$\min_{\boldsymbol{f}(\boldsymbol{x})} \quad \sum_{i=1}^n L(\boldsymbol{y}_i, \boldsymbol{f}(\boldsymbol{x}_i)) \tag{3}$$

$$\text{subject to} \quad f_1(\boldsymbol{x}) + \cdots + f_K(\boldsymbol{x}) = 0. \tag{4}$$

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 $\boldsymbol{x}$, $\boldsymbol{g}(\boldsymbol{x})$ maps $\boldsymbol{x}$ 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:

$$\mathcal{Y} = \left\{ \begin{array}{c} \left(1, -\frac{1}{K-1}, \ldots, -\frac{1}{K-1}\right)^\intercal, \\ \left(-\frac{1}{K-1}, 1, \ldots, -\frac{1}{K-1}\right)^\intercal, \\ \vdots \\ \left(-\frac{1}{K-1}, \ldots, -\frac{1}{K-1}, 1\right)^\intercal \end{array} \right\}. \tag{5}$$

Forward stagewise modeling approximates the solution to (3)–(4) 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

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

Now, we consider using the multi-class exponential loss function

$$\begin{aligned} L(\boldsymbol{y}, \boldsymbol{f}) =& \exp\left(-\frac{1}{K}(y_1 f_1 + \cdots + y_K f_K)\right) \\ =& \exp\left(-\frac{1}{K}\boldsymbol{y}^\intercal \boldsymbol{f}\right), \end{aligned}$$

in the above forward stagewise modeling algorithm. The choice of the loss function will be clear in Section 2.2 and Section 2.3. Then in step (2a), we need to find $\boldsymbol{g}^{(m)}(\boldsymbol{x})$ (and $\beta^{(m)}$) to solve:

$$\begin{aligned} & (\beta^{(m)}, \boldsymbol{g}^{(m)}) \\ = & \arg\min_{\beta, \boldsymbol{g}} \\ (6) \quad & \sum_{i=1}^{n} \exp\left(-\frac{1}{K}\boldsymbol{y}_i^\intercal(\boldsymbol{f}^{(m-1)}(\boldsymbol{x}_i) + \beta \boldsymbol{g}(\boldsymbol{x}_i))\right) \\ = & \arg\min_{\beta, \boldsymbol{g}} \\ (7) \quad & \sum_{i=1}^{n} w_i \exp\left(-\frac{1}{K}\beta \boldsymbol{y}_i^\intercal \boldsymbol{g}(\boldsymbol{x}_i)\right), \end{aligned}$$

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

Notice that every $\boldsymbol{g}(\boldsymbol{x})$ as in (5) has a one-to-one correspondence with a multi-class classifier $T(\boldsymbol{x})$ in the following way:

$$(8) \qquad T(\boldsymbol{x}) = k, \quad \text{if } g_k(\boldsymbol{x}) = 1,$$

and vice versa:

$$(9) \qquad g_k(\boldsymbol{x}) = \begin{cases} 1, & \text{if } T(\boldsymbol{x}) = k, \\ -\frac{1}{K-1}, & \text{if } T(\boldsymbol{x}) \neq k. \end{cases}$$

Hence, solving for $\boldsymbol{g}^{(m)}(\boldsymbol{x})$ in (7) 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 (7) is*

$$\begin{aligned} (10) \quad & T^{(m)}(\boldsymbol{x}) \\ = & \arg\min \sum_{i=1}^{n} w_i \mathbb{I}(c_i \neq T(\boldsymbol{x}_i)), \\ (11) \quad & \beta^{(m)} \\ = & \frac{(K-1)^2}{K}\left(\log\frac{1 - err^{(m)}}{err^{(m)}} + \log(K-1)\right), \end{aligned}$$

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

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

Based on Lemma 1, 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

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

This is equal to

$$\begin{aligned} (12) \quad & w_i \cdot e^{-\frac{(K-1)^2}{K^2}\alpha^{(m)}\boldsymbol{y}_i^\intercal \boldsymbol{g}^{(m)}(\boldsymbol{x}_i)} \\ & = \begin{cases} w_i \cdot e^{-\frac{K-1}{K}\alpha^{(m)}}, & \text{if } c_i = T(\boldsymbol{x}_i), \\ w_i \cdot e^{\frac{1}{K}\alpha^{(m)}}, & \text{if } c_i \neq T(\boldsymbol{x}_i), \end{cases} \end{aligned}$$

where $\alpha^{(m)}$ is defined as in (1) with the extra term $\log(K-1)$, and the new weight (12) is equivalent to the weight updating scheme in Algorithm 2 (2d) after normalization.

It is also a simple task to check that $\arg\max_k(f_1^{(m)}(\boldsymbol{x}), \ldots, f_K^{(m)}(\boldsymbol{x}))^\intercal$ 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.

2.2 The multi-class exponential loss

We now justify the use of the multi-class exponential loss (6). Firstly, we note that when $K = 2$, the sum-to-zero constraint indicates $\boldsymbol{f} = (f_1, -f_1)$ and then the multi-class exponential loss reduces to the exponential loss used in binary classification. [11] justified the exponential loss by showing that its population minimizer is equivalent to the Bayes rule. We follow the same arguments to investigate what is the population minimizer of this multi-class exponential loss function. Specifically, we are interested in

$$\begin{aligned} (13) \quad & \arg\min_{\boldsymbol{f}(\boldsymbol{x})} \\ & \mathrm{E}_{\boldsymbol{Y}|X=\boldsymbol{x}} \exp\left(-\frac{1}{K}(Y_1 f_1(\boldsymbol{x}) + \cdots + Y_K f_K(\boldsymbol{x}))\right) \end{aligned}$$

subject to $f_1(\boldsymbol{x}) + \cdots + f_K(\boldsymbol{x}) = 0$. The Lagrange of this constrained optimization problem can be written as:

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

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

$$\begin{aligned}
f_k^*(\boldsymbol{x}) = (K-1)\log\mathrm{Prob}(c=k|\boldsymbol{x}) - \\
\frac{K-1}{K}\sum_{k'=1}^{K}\log\mathrm{Prob}(c=k'|\boldsymbol{x}),
\end{aligned} \tag{14}$$

for $k=1,\ldots,K$. Thus,

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

which is the multi-class Bayes optimal classification rule. This result justifies the use of this multi-class exponential loss function. Equation (14) 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.

$$\mathrm{Prob}(C=k|\boldsymbol{x}) = \frac{e^{\frac{1}{K-1}f_k^*(\boldsymbol{x})}}{e^{\frac{1}{K-1}f_1^*(\boldsymbol{x})}+\cdots+c^{\frac{1}{K-1}f_K^*(\boldsymbol{x})}}, \tag{15}$$

for $k=1,\ldots,K$.

2.3 Fisher-consistent multi-class loss functions

We have shown that the population minimizer of the new multi-class exponential loss is equivalent to the multi-class Bayes rule. This property is shared by many other multi-class loss functions. Let us use the same notation as in Section 2.1, and consider a general multi-class loss function

$$\begin{aligned}
L(\boldsymbol{y},\boldsymbol{f}) &= \phi\left(-\frac{1}{K}(y_1f_1+\cdots+y_Kf_K)\right)\\
&= \phi\left(-\frac{1}{K}\boldsymbol{y}^{\mathsf{T}}\boldsymbol{f}\right),
\end{aligned} \tag{16}$$

where $\phi(\cdot)$ is a non-negative valued function. The multi-class exponential loss uses $\phi(t)=e^{-t}$. We can use the general multi-class loss function in Algorithm 3 to minimize the empirical loss

$$\frac{1}{n}\sum_{i=1}^{n}\phi\left(-\frac{1}{K}\boldsymbol{y}_i^{\mathsf{T}}\boldsymbol{f}(\boldsymbol{x_i})\right). \tag{17}$$

However, to derive a sensible algorithm, we need to require the $\phi(\cdot)$ function be *Fisher-consistent*. Specifically, we say $\phi(\cdot)$ is Fisher-consistent for K-class classification, if for $\forall\boldsymbol{x}$ in a set of full measure, the following optimization problem

$$\begin{aligned}
&\arg\min_{\boldsymbol{f}(\boldsymbol{x})}\\
&\qquad \mathrm{E}_{\boldsymbol{Y}|X=\boldsymbol{x}}\phi\left(-\frac{1}{K}(Y_1f_1(\boldsymbol{x})+\cdots+Y_Kf_K(\boldsymbol{x}))\right)
\end{aligned} \tag{18}$$

subject to $f_1(\boldsymbol{x})+\cdots+f_K(\boldsymbol{x})=0$, has a unique solution $\hat{\boldsymbol{f}}$, and

$$\arg\max_k \hat{f}_k(\boldsymbol{x}) = \arg\max_k \mathrm{Prob}(C=k|\boldsymbol{x}). \tag{19}$$

We use the sum-to-zero constraint to ensure the existence and uniqueness of the solution to (18).

Note that as $n\to\infty$, the empirical loss in (17) becomes

$$\mathrm{E}_X\left\{\mathrm{E}_{C|X=\boldsymbol{x}}\phi\left(-\frac{1}{K}(Y_1f_1(\boldsymbol{x})+\cdots+Y_Kf_K(\boldsymbol{x}))\right)\right\}. \tag{20}$$

Therefore, the multi-class Fisher-consistent condition basically says that with infinite samples, one can exactly recover the multi-class Bayes rule by minimizing the multi-class loss using $\phi(\cdot)$. Thus our definition of Fisher-consistent losses is a multi-class generalization of the binary Fisher-consistent loss function discussed in [15].

In the following theorem, we show that there are a class of convex functions that are Fisher-consistent for K-class classification, for all $K\geq 2$.

Theorem 1. *Let $\phi(t)$ be a non-negative twice differentiable function. If $\phi'(0)<0$ and $\phi''(t)>0$ for $\forall t$, then ϕ is Fisher-consistent for K-class classification for $\forall K\geq 2$. Moreover, let $\hat{\boldsymbol{f}}$ be the solution of (18), then we have*

$$\mathrm{Prob}(C=k|\boldsymbol{x}) = \frac{1/\phi'\left(\frac{1}{K-1}\hat{f}_k(\boldsymbol{x})\right)}{\sum_{k'=1}^{K}1/\phi'\left(\frac{1}{K-1}\hat{f}_{k'}(\boldsymbol{x})\right)}, \tag{21}$$

for $k=1,\ldots,K$.

Theorem 1 immediately concludes that the three most popular smooth loss functions, namely, exponential, logit and L_2 loss functions, are Fisher-consistent for all multi-class classification problems regardless the number of classes. The inversion formula (21) allows one to easily construct estimates for the conditional class probabilities. Table 1 shows the explicit inversion formulae for computing the conditional class probabilities using the exponential, logit and L_2 losses.

With these multi-class Fisher-consistent losses on hand, we can use the forward stagewise modeling strategy to derive various multi-class boosting algorithms by minimizing the empirical multi-class loss. The biggest advantage of the exponential loss is that it gives us a simple re-weighting formula. Other multi-class loss functions may not lead to such a simple closed-form re-weighting scheme. One could handle this computation issue by employing the computational

Table 1. The probability inversion formula

	exponential	logit	L_2
	$\phi(t) = e^{-t}$	$\phi(t) = \log(1 + e^{-t})$	$\phi(t) = (1 - t)^2$
$\text{Prob}(C = k\|\boldsymbol{x})$	$\frac{e^{\frac{1}{K-1}\hat{f}_k(\boldsymbol{x})}}{\sum_{k'=1}^{K} e^{\frac{1}{K-1}\hat{f}_{k'}(\boldsymbol{x})}}$	$\frac{1+e^{\frac{1}{K-1}\hat{f}_k(\boldsymbol{x})}}{\sum_{k'=1}^{K}(1+e^{\frac{1}{K-1}\hat{f}_{k'}(\boldsymbol{x})})}$	$\frac{1/(1-\frac{1}{K-1}\hat{f}_k(\boldsymbol{x}))}{\sum_{k'=1}^{K} 1/(1-\frac{1}{K-1}\hat{f}_{k'}(\boldsymbol{x}))}$

trick used in [10] and [6]. For example, [24] derived a multi-class boosting algorithm using the logit loss. A multi-class version of the L_2 boosting can be derived following the lines in [6]. We do not explore these directions in the current paper. To fix ideas, we shall focus on the multi-class AdaBoost algorithm.

3. NUMERICAL RESULTS

In this section, we use both simulation data and real-world data to demonstrate our multi-class AdaBoost algorithm. For comparison, a single decision tree (CART; [5]) and AdaBoost.MH [21] are also fit. We have chosen to compare with the AdaBoost.MH algorithm because it is conceptually easy to understand and it seems to have dominated other proposals in empirical studies [21]. Indeed, [22] also argue that with large samples, AdaBoost.MH has the optimal classification performance. 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 [11].

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 that it is a sensible algorithm, and that it is the natural extension of the AdaBoost algorithm to the multi-class case.

3.1 Simulation

We mimic a popular simulation example found in [5]. 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

$$x_j = \begin{cases} u \cdot v_1(j) + (1-u) \cdot v_2(j) + \epsilon_j, & \text{Class 1,} \\ u \cdot v_1(j) + (1-u) \cdot v_3(j) + \epsilon_j, & \text{Class 2,} \\ u \cdot v_2(j) + (1-u) \cdot v_3(j) + \epsilon_j, & \text{Class 3,} \end{cases} \tag{22}$$

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 $v_3(j) = v_1(j + 4)$.

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. The trees are built using a greedy, top-down recursive partitioning strategy, and we restrict all trees within 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 Fig. 2 and Table 2.

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.

3.2 Real data

In this section, we show the results of running SAMME on a collection of datasets from the UC-Irvine machine learning archive [18]. 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 3. 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 3 compares SAMME and AdaBoost.MH. The test error rates are summarized in Table 5. 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 [12]. The data was collected by [7], 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 Table 4.

Four male and four female speakers were used to train the classifier, and then another four male and three female speakers were used for testing the performance. Each

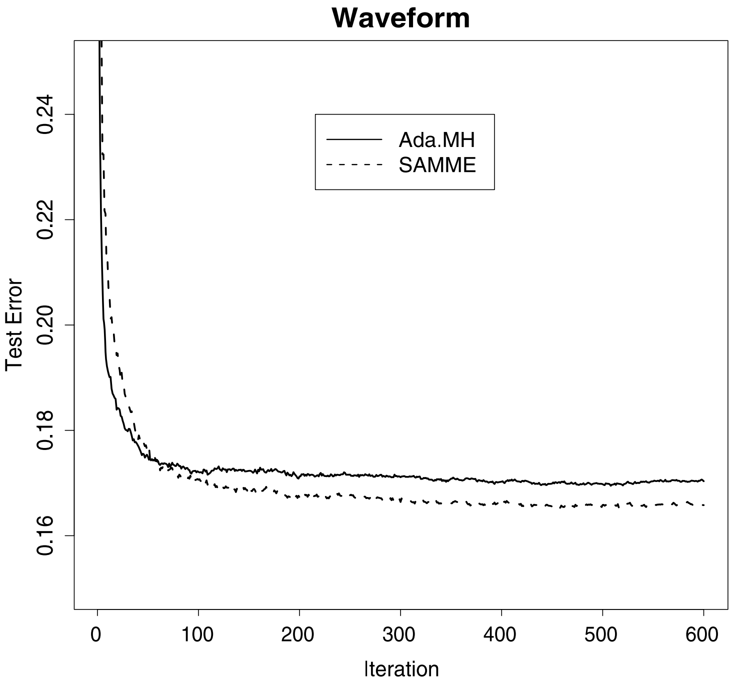

Figure 2. Test errors for SAMME and AdaBoost.MH on the `waveform` simulation example. The training sample size is 300, and the testing sample size is 5000. The results are averages of over ten independently drawn training-test set combinations.

Table 2. 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)

Table 3. 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

Table 4. The International Phonetic Association (IPA) symbols that represent the eleven vowels

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		

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. 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 Fig. 3 and Table 5, 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 test sets, and randomly split them into new training and testing sets. The procedure was repeated ten times. Again, the performance of SAMME is comparable with that of the AdaBoost.MH. For the sake of space, we do not present these results.

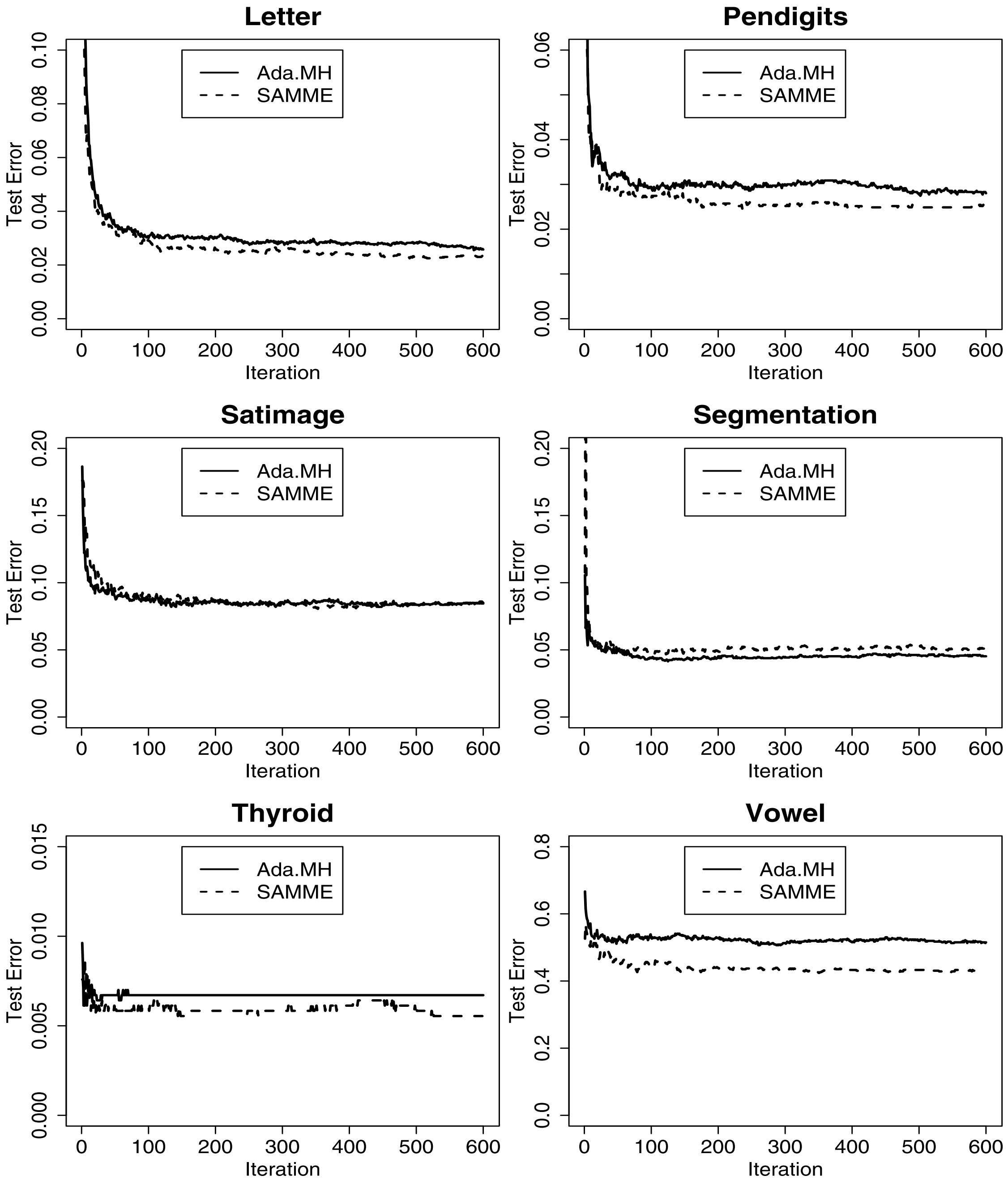

Figure 3. Test errors for SAMME and AdaBoost.MH on six benchmark datasets. These datasets come with pre-specified training and testing splits, and they are summarized in Table 3. The results for the `Nursery` *data are not shown for the test error rates are reduced to zero for both methods.*

4. DISCUSSION

The statistical view of boosting, as illustrated in [11], 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 [11] and MART [10]) into a powerful one;
- At each stage, SAMME returns only *one* weighted clas-

Table 5. 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)
`Nursery`	CART error = 0.79 (0.14)		
Ada.MH	0	0	0
SAMME	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)
`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)
`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)
`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)
`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)

sifier (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.

Our numerical experiments have indicated that AdaBoost.MH in general performs very well and 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 that it is the natural extension of the AdaBoost algorithm to the multi-class case. The success of SAMME is used here to demonstrate the usefulness of the forward stagewise modeling view of boosting.

[11] called the AdaBoost algorithm *Discrete* AdaBoost and proposed *Real* AdaBoost and *Gentle* AdaBoost algorithms which combine regressors to estimate the conditional class probability. Using their language, SAMME is also a discrete multi-class AdaBoost. We have also derived the corresponding Real Multi-class AdaBoost and Gentle Multi-class AdaBoost [23, 24]. These results further demonstrate the usefulness of the forward stagewise modeling view of boosting.

It should also be emphasized here that although our statistical view of boosting leads to interesting and useful results, we do not argue it is the ultimate explanation of boosting. Why boosting works is still an open question. Interested readers are referred to the discussions on [11]. [9] mentioned that the forward stagewise modeling view of AdaBoost does not offer a bound on the generalization error as in the original AdaBoost paper [8]. [3] also pointed out that the statistical view of boosting does not explain why AdaBoost is robust against overfitting. Later, his understandings of AdaBoost lead to the invention of random forests [4].

Finally, we discuss the computational cost of SAMME. Suppose one uses a classification tree as the weak learner, 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.

The SAMME algorithm has been implemented in the `R` computing environment, and will be publicly available from the authors' websites.

APPENDIX: PROOFS

Lemma 1. First, for any fixed value of $\beta > 0$, using the definition (8), one can express the criterion in (7) as:

$$\begin{aligned} & \sum_{c_i=T(\boldsymbol{x}_i)} w_i e^{-\frac{\beta}{K-1}} + \sum_{c_i\neq T(\boldsymbol{x}_i)} w_i e^{\frac{\beta}{(K-1)^2}} \\ = \; & e^{-\frac{\beta}{K-1}} \sum_i w_i + \\ & (e^{\frac{\beta}{(K-1)^2}} - e^{-\frac{\beta}{K-1}}) \sum_i w_i \mathbb{I}(c_i \neq T(\boldsymbol{x}_i)). \end{aligned} \tag{23}$$

Since only the last sum depends on the classifier $T(\boldsymbol{x})$, we get that (10) holds. Now plugging (10) into (7) and solving for β, we obtain (11) (note that (23) is a convex function of β). □

Theorem 1. Firstly, we note that under the sum-to-zero constraint,

$$\begin{aligned} & \mathrm{E}_{\boldsymbol{Y}|X=\boldsymbol{x}} \phi\left(\frac{1}{K}(Y_1 f_1(\boldsymbol{x}) + \cdots + Y_K f_K(\boldsymbol{x}))\right) \\ = \; & \phi\left(\frac{f_1(\boldsymbol{x})}{K-1}\right) \mathrm{Prob}(C=1|\boldsymbol{x}) + \\ & \cdots \\ & +\phi\left(\frac{f_K(\boldsymbol{x})}{K-1}\right) \mathrm{Prob}(C=K|\boldsymbol{x}). \end{aligned}$$

Therefore, we wish to solve

$$\begin{aligned} \min_{\boldsymbol{f}} \quad & \phi(\frac{1}{K-1}f_1(\boldsymbol{x}))\mathrm{Prob}(C=1|\boldsymbol{x}) + \\ & \cdots \\ & +\phi(\frac{1}{K-1}f_K(\boldsymbol{x}))\mathrm{Prob}(C=1|\boldsymbol{x}) \\ \text{subject to} \quad & \sum_{k=1}^{K} f_k(\boldsymbol{x}) = 0. \end{aligned}$$

For convenience, let $p_k = \mathrm{Prob}(C = k|\boldsymbol{x})$, $k = 1, 2, \ldots, K$ and we omit $\boldsymbol{x}$ in $f_k(\boldsymbol{x})$. Using the Lagrangian multiplier, we define

$$\begin{aligned} Q(\boldsymbol{f}) = & \ \phi(\frac{1}{K-1}f_1)p_1 + \\ & \cdots + \\ & \phi(\frac{1}{K-1}f_K)p_K + \frac{1}{K-1}\lambda(f_1 + \ldots + f_K). \end{aligned}$$

Then we have

$$\frac{\partial Q(\boldsymbol{f})}{\partial f_k} = \frac{1}{K-1}\phi'(\frac{1}{K-1}f_k)p_k + \frac{1}{K-1}\lambda = 0, \tag{24}$$

for $k = 1, \ldots, K$. Since $\phi''(t) > 0$ for $\forall t$, ϕ' has an inverse function, denoted by ψ. Equation (24) gives $\frac{1}{K-1}f_k = \psi(-\frac{\lambda}{p_k})$. By the sum-to-zero constraint on $\boldsymbol{f}$, we have

$$\sum_{k=1}^{K} \psi\left(-\frac{\lambda}{p_k}\right) = 0. \tag{25}$$

Since ϕ' is a strictly monotone increasing function, so is ψ. Thus the left hand size (LHS) of (25) is a decreasing function of λ. It suffices to show that equation (25) has a root λ^*, which is the unique root. Then it is easy to see that $\hat{f}_k = \psi(-\frac{\lambda^*}{p_k})$ is the unique minimizer of (18), for the Hessian matrix of $Q(\boldsymbol{f})$ is a diagonal matrix and the k-th diagonal element is $\frac{\partial^2 Q(\boldsymbol{f})}{\partial f_k^2} = \frac{1}{(K-1)^2}\phi''(\frac{1}{K-1}f_k) > 0$. Note that when $\lambda = -\phi'(0) > 0$, we have $\frac{\lambda}{p_k} > -\phi'(0)$, then $\psi(-\frac{\lambda}{p_k}) < \psi\left(\phi'(0)\right) = 0$. So the LHS of (25) is negative when $\lambda = -\phi'(0) > 0$. On the other hand, let us define $A = \{a : \phi'(a) = 0\}$. If A is an empty set, then $\phi'(t) \to 0-$ as $t \to \infty$ (since ϕ is convex). If A is not empty, denote $a^* = \inf A$. By the fact $\phi'(0) < 0$, we conclude $a^* > 0$. Hence $\phi'(t) \to 0-$ as $t \to a^*-$. In both cases, we see that $\exists$ a small enough $\lambda_0 > 0$ such that $\psi(-\frac{\lambda_0}{p_k}) > 0$ for all k. So the LHS of (25) is positive when $\lambda = \lambda_0 > 0$. Therefore there must be a positive $\lambda^* \in (\lambda_0, -\phi'(0))$ such that equation (25) holds. Now we show the minimizer $\hat{\boldsymbol{f}}$ agrees with the Bayes rule. Without loss of generality, let $p_1 > p_k$ for $\forall k \neq 1$. Then since $-\frac{\lambda^*}{p_1} > -\frac{\lambda^*}{p_k}$ for $\forall k \neq 1$, we have $\hat{f}_1 > \hat{f}_k$ for $\forall k \neq 1$. For the inversion formula, we note $p_k = -\frac{\lambda^*}{\phi'(\frac{1}{K-1}\hat{f}_k)}$, and $\sum_{k=1}^{K} p_j = 1$ requires $\sum_{k=1}^{K} -\frac{\lambda^*}{\phi'(\frac{1}{K-1}\hat{f}_k)} = 1$. Hence it follows that $\lambda^* = -(\sum_{k=1}^{K} 1/\phi'(\frac{1}{K-1}\hat{f}_k))^{-1}$. Then (21) is obtained. □

ACKNOWLEDGMENTS

We 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.

Received 22 May 2009

REFERENCES

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

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

[3] Breiman, L. (2000). Discussion of "Additive logistic regression: a statistical view of boosting" by Friedman, Hastie and Tibshirani. *Annals of Statistics* **28** 374–377. MR1790002

[4] Breiman, L. (2001). Random forests. *Machine Learning* **45** 5–32.

[5] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees.* Wadsworth, Belmont, CA. MR0726392

[6] Bühlmann, P. and Yu, B. (2003). Boosting with the ℓ_2 loss: regression and classification. *Journal of the American Statistical Association* **98** 324–339. MR1995709

[7] Deterding, D. (1989). *Speaker Normalization for Automatic Speech Recognition.* University of Cambridge. Ph.D. thesis.

[8] Freund, Y. and Schapire, R. (1997). A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55** 119–139. MR1473055

[9] Freund, Y. and Schapire, R. (2000). Discussion of "Additive logistic regression: a statistical view on boosting" by Friedman, Hastie and Tibshirani. *Annals of Statistics* **28** 391–393. MR1790002

[10] Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics* **29** 1189–1232. MR1873328

[11] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics* **28** 337–407. MR1790002

[12] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning.* Springer-Verlag, New York. MR1851606

[13] Koltchinskii, V. and Panchenko, D. (2002). Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics* **30** 1–50. MR1892654

[14] Lee, Y., Lin, Y., and 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. MR2054287

[15] Lin, Y. (2004). A note on margin-based loss functions in classification. *Statistics and Probability Letters* **68** 73–82. MR2064687

[16] Liu, Y. and Shen, X. (2006). Multicategory psi-learning. *Journal of the American Statistical Association* **101** 500–509. MR2256170

[17] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Boosting algorithms as gradient descent in function space. *Neural Information Processing Systems* **12**.

[18] Merz, C. and Murphy, P. (1998). UCI repository of machine learning databases.

[19] SCHAPIRE, R. (1997). Using output codes to boost multiclass learning problems. *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kauffman.

[20] SCHAPIRE, R., FREUND, Y., BARTLETT, P., and LEE, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics* **26** 1651–1686. MR1673273

[21] SCHAPIRE, R. and SINGER, Y. (1999). Improved boosting algorithms using confidence-rated prediction. *Machine Learning* **37** 297–336. MR1811573

[22] ZHANG, T. (2004). Statistical analysis of some multi-category large margin classification methods. *Journal of Machine Learning Research* **5** 1225–1251. MR2248016

[23] ZHU, J., ROSSET, S., ZOU, H., and HASTIE, T. (2005). Multiclass AdaBoost. Technical Report # 430, Department of Statistics, University of Michigan.

[24] ZOU, H., ZHU, J., and HASTIE, T. (2008). The margin vector, admissable loss, and multiclass margin-based classifiers. *Annals of Applied Statistics* **2** 1290–1306.

Ji Zhu
Department of Statistics
University of Michigan
Ann Arbor, MI 48109
USA
E-mail address: jizhu@umich.edu

Hui Zou
School of Statistics
University of Minnesota
Minneapolis, MN 55455
USA
E-mail address: hzou@stat.umn.edu

Saharon Rosset
Department of Statistics
Tel Aviv University
Tel Aviv 69978
Israel
E-mail address: saharon@tau.ac.il

Trevor Hastie
Department of Statistics
Stanford University
Stanford, CA 94305
USA
E-mail address: hastie@stanford.edu