Journal of Machine Learning Research 6 (2005) 189–210 Submitted 6/03; Revised 7/04; Published 2/05

# Multiclass Boosting for Weak Classifiers

**Günther Eibl** GUENTHER.EIBL@UIBK.AC.AT
**Karl–Peter Pfeiffer** KARL-PETER.PFEIFFER@UIBK.AC.AT
*Department of Biostatistics*
*University of Innsbruck*
*Schöpfstrasse 41, 6020 Innsbruck, Austria*

**Editor:** Robert Schapire

## Abstract

AdaBoost.M2 is a boosting algorithm designed for multiclass problems with weak base classifiers. The algorithm is designed to minimize a very loose bound on the training error. We propose two alternative boosting algorithms which also minimize bounds on performance measures. These performance measures are not as strongly connected to the expected error as the training error, but the derived bounds are tighter than the bound on the training error of AdaBoost.M2. In experiments the methods have roughly the same performance in minimizing the training and test error rates. The new algorithms have the advantage that the base classifier should minimize the confidence-rated error, whereas for AdaBoost.M2 the base classifier should minimize the pseudo-loss. This makes them more easily applicable to already existing base classifiers. The new algorithms also tend to converge faster than AdaBoost.M2.

**Keywords:** boosting, multiclass, ensemble, classification, decision stumps

## 1. Introduction

Most papers about boosting theory consider two-class problems. Multiclass problems can be either reduced to two-class problems using error-correcting codes (Allwein et al., 2000; Dietterrich and Bakiri, 1995; Guruswami and Sahai, 1999) or treated more directly using base classifiers for multiclass problems. Freund and Schapire (1996 and 1997) proposed the algorithm AdaBoost.M1 which is a straightforward generalization of AdaBoost using multiclass base classifiers. An exponential decrease of an upper bound of the training error rate is guaranteed as long as the error rates of the base classifiers are less than 1/2. For more than two labels this condition can be too restrictive for weak classifiers like decision stumps which we use in this paper. Freund and Schapire overcame this problem with the introduction of the pseudo-loss of a classifier $h : X \times Y \rightarrow [0,1]$ :

$$\varepsilon_t = \frac{1}{2}\left(1 - h_t(x_i,y_i) + \frac{1}{|Y|-1}\sum_{y \neq y_i} h_t(x_i,y)\right).$$

In the algorithm AdaBoost.M2, each base classifier has to minimize the pseudo-loss instead of the error rate. As long as the pseudo-loss is less than 1/2, which is easily reachable for weak base classifiers as decision stumps, an exponential decrease of an upper bound on the training error rate is guaranteed.

In this paper, we will derive two new direct algorithms for multiclass problems with decision stumps as base classifiers. The first one is called GrPloss and has its origin in the gradient descent framework of Mason et al. (1998, 1999). Combined with ideas of Freund and Schapire (1996, 1997) we get an exponential bound on a performance measure which we call pseudo-loss error. The second algorithm was motivated by the attempt to make AdaBoost.M1 work for weak base classifiers. We introduce the maxlabel error rate and derive bounds on it. For both algorithms, the bounds on the performance measures decrease exponentially under conditions which are easy to fulfill by the base classifier. For both algorithms the goal of the base classifier is to minimize the confidence-rated error rate which makes them applicable for a wide range of already existing base classifiers.

Throughout this paper $S = \{(x_i, y_i);\ i = 1, \ldots, N)\}$ denotes the training set where each $x_i$ belongs to some instance or measurement space $X$ and each label $y_i$ is in some label set $Y$. In contrast to the two-class case, $Y$ can have $|Y| \geq 2$ elements. A boosting algorithm calls a given weak classification algorithm $h$ repeatedly in a series of rounds $t = 1, \ldots, T$. In each round, a sample of the original training set $S$ is drawn according to the weighting distribution $D_t$ and used as training set for the weak classification algorithm $h$. $D_t(i)$ denotes the weight of example $i$ of the original training set $S$. The final classifier $H$ is a weighted majority vote of the $T$ weak classifiers $h_t$ where $\alpha_t$ is the weight assigned to $h_t$. Finally, the elements of a set $M$ that maximize and minimize a function $f$ are denoted $\arg\max_{m \in M} f(m)$ and $\arg\min_{m \in M} f(m)$ respectively.

## 2. Algorithm GrPloss

In this section we will derive the algorithm GrPloss. Mason et al. (1998, 1999) embedded AdaBoost in a more general theory which sees boosting algorithms as gradient descent methods for the minimization of a loss function in function space. We get GrPloss by applying the gradient descent framework especially for minimizing the exponential pseudo-loss. We first consider slightly more general exponential loss functions. Based on the gradient descent framework, we derive a gradient descent algorithm for these loss functions in a straight forward way in Section 2.1. In contrast to the general framework, we can additionally derive a simple update rule for the sampling distribution as it exists for AdaBoost.M1 and AdaBoost.M2. Gradient descent does not provide a special choice for the “step size” $\alpha_t$. In Section 2.2, we define the pseudo-loss error and derive $\alpha_t$ by minimization of an upper bound on the pseudo-loss error. Finally, the algorithm is simplified for the special case of decision stumps as base classifiers.

### 2.1 Gradient Descent for Exponential Loss Functions

First we briefly describe the gradient descent framework for the two-class case with $Y = \{-1, +1\}$. As usual a training set $S = \{(x_i, y_i);\ i = 1, \ldots, N)\}$ is given. We are considering a function space $\mathcal{F} = \mathrm{lin}(\mathcal{H})$ consisting of functions $f : X \to \mathbb{R}$ of the form

$$f(x; \vec{\alpha}, \vec{\beta}) = \sum_{t=1}^{T} \alpha_t h_t(x; \beta_t), \qquad h_t : X \to \{\pm 1\}$$

with $\vec{\alpha} = (\alpha_1, \ldots, \alpha_T) \in \mathbb{R}^T$, $\vec{\beta} = (\beta_1, \ldots, \beta_T)$ and $h_t \in \mathcal{H}$. The parameters $\beta_t$ uniquely determine $h_t$ therefore $\vec{\alpha}$ and $\vec{\beta}$ uniquely determine $f$. We choose a loss function

$$L(f) = E_{y,x}[l(f(x), y)] = E_x[E_y[l(yf(x))]] \qquad l : \mathbb{R} \to \mathbb{R}_{\geq 0}$$

where for example the choice of $l(f(x),y) = e^{-yf(x)}$ leads to

$$L(f) = \frac{1}{N}\sum_{i=1}^{N} e^{y_i f(x_i)}.$$

The goal is to find $f^* = \arg\min_{f\in\mathcal{F}} L(f)$.

The gradient in function space is defined as:

$$\nabla L(f)(x) := \frac{\partial L(f+e1_x)}{\partial e}\Big|_{e=0} = \lim_{e\to 0}\frac{L(f+e1_x)-L(f)}{e}$$

where for two arbitrary tuples $v$ and $\tilde{v}$ we denote

$$1_v(\tilde{v}) = \begin{cases} 1 & \tilde{v} = v \\ 0 & \tilde{v} \neq v. \end{cases}$$

A gradient descent method always makes a step in the "direction" of the negative gradient $-\nabla L(f)(x)$. However $-\nabla L(f)(x)$ is not necessarily an element of $\mathcal{F}$, so we replace it by an element $h_t$ of $\mathcal{F}$ which is as parallel to $-\nabla L(f)(x)$ as possible. Therefore we need an inner product $\langle\, ,\, \rangle : \mathcal{F}\times\mathcal{F}\to\mathbb{R}$, which can for example be chosen as

$$\langle f, \tilde{f}\rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)\tilde{f}(x_i).$$

This inner product measures the agreement of $f$ and $\tilde{f}$ on the training set. Using this inner product we can set

$$\beta_t := \arg\max_{\beta}\langle -\nabla L(f_{t-1}), h(\cdot\,;\beta)\rangle$$

and $h_t := h(\cdot\,;\beta_t)$. The inequality $\langle -\nabla L(f_{t-1}), h(\beta_t)\rangle \leq 0$ means that we can not find a good "direction" $h(\beta_t)$, so the algorithm stops, when this happens. The resulting algorithm is given in Figure 1.

```
Input: training set $S$, loss function $l$, inner product $\langle\, ,\, \rangle : \mathcal{F}\times\mathcal{F}\to\mathbb{R}$, starting value $f_0$.
$t := 1$
Loop: while $\langle -\nabla L(f_{t-1}), h(\beta_t)\rangle > 0$
    • $\beta_t := \arg\max_{\beta}\langle -\nabla L(f_{t-1}), h(\beta)\rangle$
    • $\alpha_t := \arg\min_{\alpha}(L(f_{t-1} + \alpha h_t(\beta_t)))$
    • $f_t = f_{t-1} + \alpha_t h_t(\beta_t)$
Output: $f_t, L(f_t)$
```

Figure 1: Algorithm gradient descent in function space

Now we go back to the multiclass case and modify the gradient descent framework in order to treat classifiers $f$ of the form $f : X\times Y\to\mathbb{R}$, where $f(x,y)$ is a measure of the confidence, that an object with measurements $x$ has the label $y$. We denote the set of possible classifiers with $\mathcal{F}$. For gradient descent we need a loss function and an inner product on $\mathcal{F}$. We choose

$$\langle f, \hat{f} \rangle := \frac{1}{N} \sum_{i=1}^{N} \sum_{y=1}^{|Y|} f(x_i, y) \hat{f}(x_i, y),$$

which is a straightforward generalization of the definition for the two-class case. The goal of the classification algorithm GrPloss is to minimize the special loss function

$$L(f) := \frac{1}{N} \sum_i l(f, i) \quad \text{with} \quad l(f, i) := \exp\left[\frac{1}{2}\left(1 - f(x_i, y_i) + \sum_{y \neq y_i} \frac{f(x_i, y)}{|Y| - 1}\right)\right]. \tag{1}$$

The term

$$-f(x_i, y_i) + \sum_{y \neq y_i} \frac{f(x_i, y)}{|Y| - 1}$$

compares the confidence to label the example $x_i$ correctly with the mean confidence of choosing one of the wrong labels. Now we consider slightly more general exponential loss functions

$$l(f, i) = \exp\left[v(f, i)\right] \quad \text{with exponent} - \text{loss } v(f, i) = v_0 + \sum_y v_y(i) f(x_i, y) \ ,$$

where the choice

$$v_0 = \frac{1}{2} \text{ and } v_y(i) = \begin{cases} -\frac{1}{2} & y = y_i \\ \frac{1}{2(|Y|-1)} & y \neq y_i \end{cases}$$

leads to the loss function (1). This choice of the loss function leads to the algorithm given in Figure 2. The properties summarized in Theorem 1 can be shown to hold on this algorithm.

```
Input: training set $S$, maximum number of boosting rounds $T$
Initialisation: $f_0 := 0, t := 1, \forall i: D_1(i) := \frac{1}{N}$.
Loop: For $t = 1, \ldots, T$ do
    • $h_t = \arg\min_h \sum_i D_t(i) v(h, i)$
    • If $\sum_i D_t(i) v(h_t, i) \geq v_0 :\ T := t - 1$, goto output.
    • Choose $\alpha_t$.
    • Update $f_t = f_{t-1} + \alpha_t h_t$ and $D_{t+1}(i) = \frac{1}{Z_t} D_t(i) l(\alpha_t h_t, i)$
Output: $f_T, L(f_T)$
```

Figure 2: Gradient descent for exponential loss functions

**Theorem 1** *For the inner product*

$$\langle f, h \rangle = \frac{1}{N} \sum_{i=1}^{N} \sum_{y=1}^{|Y|} f(x_i, y) h(x_i, y)$$

*and any exponential loss functions $l(f,i)$ of the form*

$$l(f,i) = \exp[v(f,i)] \quad \text{with} \quad v(f,i) = v_0 + \sum_y v_y(i) f(x_i, y)$$

*where $v_0$ and $v_y(i)$ are constants, the following statements hold:*
*(i) The choice of $h_t$ that maximizes the projection on the negative gradient*

$$h_t = \arg\max_h \langle -\nabla L(f_{t-1}), h \rangle$$

*is equivalent to that minimizing the weighted exponent-loss*

$$h_t = \arg\min_h \sum_i D_t(i) v(h,i)$$

*with respect to the sampling distribution*

$$D_t(i) := \frac{l(f_{t-1}, i)}{\sum_{i'} l(f_{t-1}, i')} = \frac{l(f_{t-1}, i)}{Z'_{t-1}}.$$

*(ii) The stopping criterion of the gradient descent method*

$$\langle -\nabla L(f_{t-1}), h(\beta_t) \rangle \leq 0$$

*leads to a stop of the algorithm, when the weighted exponent-loss gets positive*

$$\sum_i D_t(i) v(h_t, i) \geq v_0.$$

*(iii) The sampling distribution can be updated in a similar way as in AdaBoost using the rule*

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) l(\alpha_t h_t, i),$$

*where we define $Z_t$ as a normalization constant*

$$Z_t := \sum_i D_t(i) l(\alpha_t h_t, i),$$

*which ensures that the update $D_{t+1}$ is a distribution.*

In contrast to the general framework, the algorithm uses a simple update rule for the sampling distribution as it exists for the original boosting algorithms. Note that the algorithm does not specify the choice of the step size $\alpha_t$, because gradient descent only provides an upper bound on $\alpha_t$. We will derive a special choice for $\alpha_t$ in the next section.

**Proof**. The proof basically consists of three steps: the calculation of the gradient, the choice for base classifier $h_t$ together with the stopping criterion and the update rule for the sampling distribution.
(i) First we calculate the gradient, which is defined by

$$\nabla L(f)(x,y) := \lim_{k\to 0} \frac{L(f+k1_{(x,y)}) - L(f)}{k}$$

for $1_{(x,y)}(x',y') = \begin{cases} 1 & (x,y)=(x',y') \\ 0 & (x,y)\neq(x',y') \end{cases}$.

So we get for $x = x_i$:

$$L(f+k1_{x_iy}) = \frac{1}{N}\exp\left[v_0 + \sum_{y'} v_{y'}(i) f(x_i,y') + kv_y(i)\right] = \frac{1}{N} l(f,i)e^{kv_y(i)}.$$

Substitution in the definition of $\nabla L(f)$ leads to

$$\nabla L(f)(x_i,y) = \lim_{k\to 0} \frac{l(f,i)(e^{kv_y(i)}-1)}{k} = l(f,i)v_y(i).$$

Thus

$$\nabla L(f)(x,y) = \begin{cases} 0 & x \neq x_i \\ l(f,i)v_y(i) & x = x_i \end{cases}. \tag{2}$$

Now we insert (2) into $\langle -\nabla L(f_{t-1}), h_t\rangle$ and get

$$\langle -\nabla L(f_{t-1}), h_t\rangle = -\frac{1}{N}\sum_i \sum_y l(f_{t-1},i)v_y(i)h(x_i,y) = -\frac{1}{N}\sum_i l(f_{t-1},i)(v(h,i)-v_0). \tag{3}$$

If we define the sampling distribution $D_t$ up to a positive constant $C_{t-1}$ by

$$D_t(i) := C_{t-1} l(f_{t-1},i), \tag{4}$$

we can write (3) as

$$\langle -\nabla L(f_{t-1}), h_t\rangle = -\frac{1}{C_{t-1}N}\sum_i D_t(i)(v(h,i)-v_0) = -\frac{1}{C_{t-1}N}\left(\sum_i D_t(i)v(h,i) - v_0\right). \tag{5}$$

Since we require $C_{t-1}$ to be positive, we get the choice of $h_t$ of the algorithm

$$h_t = \arg\max_h \langle -\nabla L(f_{t-1}), h\rangle = \arg\min_h \sum_i D_t(i)v(h,i).$$

(ii) One can verify the stopping criterion of Figure 2 from (5)

$$\langle -\nabla L(f_{t-1}), h_t\rangle \leq 0 \Leftrightarrow \sum_i D_t(i)v(h_t,i) \geq v_0.$$

(iii) Finally, we show that we can calculate the update rule for the sampling distribution $D$.

$$\begin{aligned} D_{t+1}(i) &= C_t l(f_t,i) = C_t l(f_{t-1}+\alpha_t h_t, i) \\ &= C_t l(f_{t-1},i) l(\alpha_t h_t, i) = \frac{C_t}{C_{t-1}} D_t(i) l(\alpha_t h_t, i). \end{aligned}$$

This means that the new weight of example $i$ is a constant multiplied with $D_t(i)l(\alpha_t h_t, i)$. By comparing this equation with the definition of $Z_t$ we can determine $C_t$

$$C_t = \frac{C_{t-1}}{Z_t}.$$

Since $l$ is positive and the weights are positive one can show by induction, that also $C_t$ is positive, which we required before. ■

## 2.2 Choice of $\alpha_t$ and Resulting Algorithm GrPloss

The algorithm above leaves the step length $\alpha_t$, which is the weight of the base classifier $h_t$, unspecified. In this section we define the pseudo-loss error and derive $\alpha_t$ by minimization of an upper bound on the pseudo-loss error.

**Definition**: A classifier $f : X \times Y \to \mathbb{R}$ makes a pseudo-loss error in classifying an example $x$ with label $k$, if

$$f(x,k) < \frac{1}{|Y|-1} \sum_{y \neq k} f(x,y).$$

The corresponding training error rate is denoted by $plerr$:

$$plerr := \frac{1}{N} \sum_{i=1}^{N} I\left( f(x_i,y_i) < \frac{1}{|Y|-1} \sum_{y \neq y_i} f(x_i,y) \right).$$

The pseudo-loss error counts the proportion of elements in the training set for which the confidence $f(x,k)$ in the right label is smaller than the *average* confidence in the remaining labels $\sum_{y \neq k} f(x,y)/(|Y|-1)$. Thus it is a weak measure for the performance of a classifier in the sense that it can be much smaller than the training error.

Now we consider the exponential pseudo-loss. The constant term of the pseudo-loss leads to a constant factor which can be put into the normalizing constant. So with the definition

$$u(f,i) := f(x_i,y_i) - \frac{1}{|Y|-1} \sum_{y \neq y_i} f(x_i,y)$$

the update rule can be written in the shorter form

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) e^{-\alpha_t u(h_t,i)/2}, \text{ with } Z_t := \sum_{i=1}^{N} D_t(i) e^{-\alpha_t u(h_t,i)/2}.$$

We present our next algorithm, GrPloss, in Figure 3, which we will derive and justify in what follows.

(i) Similar to Schapire and Singer (1999) we first bound $plerr$ by the product of the normalization constants

$$plerr \leq \prod_{t=1}^{T} Z_t. \tag{6}$$

To prove (6), we first notice that

$$plerr \leq \frac{1}{N} \sum_i e^{-u(f_T,i)/2}. \tag{7}$$

**Input:** training set $S = \{(x_1,y_1),\ldots,(x_N,y_N); x_i \in X, y_i \in Y\}$,
$Y = \{1,\ldots,|Y|\}$, weak classification algorithm with output $h : X \times Y \to [0,1]$
Optionally $T$: maximal number of boosting rounds

**Initialization:** $D_1(i) = \frac{1}{N}$.

**For** $t = 1,\ldots,T$**:**

- Train the weak classification algorithm $h_t$ with distribution $D_t$, where $h_t$ should maximize $U_t := \sum_i D_t(i)u(h_t,i)$.
- If $U_t \le 0$: goto output with $T := t-1$
- Set
$$\alpha_t = \ln\left(\frac{1+U_t}{1-U_t}\right).$$
- Update D:
$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) e^{-\alpha_t u(h_t,i)/2}.$$
where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ is a distribution)

**Output:** final classifier $H(x)$:
$$H(x) = \arg\max_{y\in Y} f(x,y) = \arg\max_{y\in Y}\left(\sum_{t=1}^{T}\alpha_t h_t(x,y)\right)$$

Figure 3: Algorithm GrPloss

Now we unravel the update rule
$$\begin{aligned}
D_{T+1}(i) &= \frac{1}{Z_T} e^{-\alpha_T u(h_T,i)/2} D_T(i) \\
&= \frac{1}{Z_T Z_{T-1}} e^{-\alpha_T u(h_T,i)/2} e^{-\alpha_{T-1} u(h_{T-1},i)/2} D_{T-1}(i) \\
&= \ldots = D_1(i) \prod_{t=1}^{T} e^{-\alpha_t u(h_t,i)/2} \frac{1}{Z_t} \\
&= \frac{1}{N} \exp\left(-\sum_{t=1}^{T} \alpha_t u(h_t,i)/2\right) \prod_{t=1}^{T} \frac{1}{Z_t} \\
&= \frac{1}{N} e^{-u(f_T,i)/2} \prod_{t=1}^{T} \frac{1}{Z_t}
\end{aligned}$$

where the last equation uses the property that $u$ is linear in $h$. Since
$$1 = \sum_i D_{T+1}(i) = \sum_i \frac{1}{N} e^{-u(f_T,i)/2} \prod_{t=1}^{T} \frac{1}{Z_T}$$

we get Equation (6) by using (7) and the equation above

$$plerr \leq \frac{1}{N}\sum_i e^{-u(f_T,i)/2} = \prod_{t=1}^{T} Z_t.$$

(ii) Derivation of $\alpha_t$:
Now we derive $\alpha_t$ by minimizing the upper bound (6). First, we plug in the definition of $Z_t$

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T}\left(\sum_i D_t(i)e^{-\alpha_t u(h_t,i)/2}\right).$$

Now we get an upper bound on this product using the convexity of the function $e^{-\alpha_t u}$ between $-1$ and +1 (from $h(x,y) \in [0,1]$ it follows that $u \in [-1,+1]$) for positive $\alpha_t$:

$$\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T}\left(\sum_i D_t(i)\frac{1}{2}[(1-u(h_t,i))e^{+\frac{1}{2}\alpha_t} + (1+u(h_t,i))e^{-\frac{1}{2}\alpha_t}]\right). \tag{8}$$

Now we choose $\alpha_t$ in order to minimize this upper bound by setting the first derivative with respect to $\alpha_t$ to zero. To do this, we define

$$U_t := \sum_i D_t(i)u(h_t,i).$$

Since each $\alpha_t$ occurs in exactly one factor of the bound (8) the result for $\alpha_t$ only depends on $U_t$ and not on $U_s,\ s \neq t$, more specifically

$$\alpha_t = \ln\left(\frac{1+U_t}{1-U_t}\right).$$

Note that $U_t$ has its values in the interval $[-1,1]$, because $u_t \in [-1,+1]$ and $D_t$ is a distribution.
(iii) Derivation of the upper bound of the theorem:
Now we substitute $\alpha_t$ back in (8) and get after some straightforward calculations

$$\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T}\sqrt{1-U_t^2}.$$

Using the inequality $\sqrt{1-x} \leq (1-\frac{1}{2}x) \leq e^{-x/2}$ for $x \in [0,1]$ we can get an exponential bound on $\prod_t Z_t$

$$\prod_{t=1}^{T} Z_t \leq \exp\left[\sum_{t=1}^{T} -U_t^2/2\right].$$

If we assume that each classifier $h_t$ fulfills $U_t \geq \delta$, we finally get

$$\prod_{t=1}^{T} Z_t \leq e^{-\delta^2 T/2}.$$

(iv) Stopping criterion:
The stopping criterion of the slightly more general algorithm directly results in the new stopping criterion to stop, when $U_t \leq 0$. However, note that the bound depends on the square of $U_t$ instead of $U_t$ leading to a formal decrease of the bound even when $U_t > 0$.

We summarize the foregoing argument as a theorem.

**Theorem 2** *If for all base classifiers $h_t : X \times Y \to [0,1]$ of the algorithm GrPloss given in Figure 3*

$$U_t := \sum_i D_t(i) u(h_t, i) \geq \delta$$

*holds for $\delta > 0$ then the pseudo-loss error of the training set fulfills*

$$plerr \leq \prod_{t=1}^{T} \sqrt{1 - U_t^2} \leq e^{-\delta^2 T/2}. \tag{9}$$

### 2.3 GrPloss for Decision Stumps

So far we have considered classifiers of the form $h : X \times Y \to [0,1]$. Now we want to consider base classifiers that have additionally the normalization property

$$\sum_{y \in Y} h(x,y) = 1 \tag{10}$$

which we did not use in the previous section for the derivation of $\alpha_t$. The decision stumps we used in our experiments find an attribute $a$ and a value $v$ which are used to divide the training set into two subsets. If attribute $a$ is continuous and its value on $x$ is at most $v$ then $x$ belongs to the first subset; otherwise $x$ belongs to the second subset. If attribute $a$ is categorical the two subsets correspond to a partition of all possible values of $a$ into two sets. The prediction $h(x,y)$ is the proportion of examples with label $y$ belonging to the same subset as $x$. Since proportions are in the interval $[0,1]$ and for each of the two subsets the sum of proportions is one our decision stumps have both the former and the latter property (10). Now we use these properties to minimize a tighter bound on the pseudo-loss error and further simplify the algorithm.

(i) Derivation of $\alpha_t$:
To get $\alpha_t$ we can start with

$$plerr \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left( \sum_i D_t(i) e^{-\alpha_t u(h_t,i)/2} \right)$$

which was derived in part (i) of the proof of the previous section. First, we simplify $u(h,i)$ using the normalization property and get

$$u(h,i) = \frac{|Y|}{|Y|-1} h(x_i,y_i) - \frac{1}{|Y|-1} \tag{11}$$

In contrast to the previous section, $u(h,i) \in [-\frac{1}{|Y|-1}, 1]$ for $h(x_i,y_i) \in [0,1]$, which we will take into account for the convexity argument:

$$plerr \leq \prod_{t=1}^{T} \sum_{i=1}^{N} D_t(i) \left( h(x_i,y_i)\, \mathrm{e}^{-\alpha_t/2} + (1 - h_t(x_i,y_i))\, \mathrm{e}^{\alpha_t/(2(|Y|-1))} \right) \tag{12}$$

Setting the first derivative with respect to $\alpha_t$ to zero leads to

$$\alpha_t = \frac{2(|Y|-1)}{|Y|} \ln\left(\frac{(|Y|-1)r_t}{1-r_t}\right),$$

where we defined

$$r_t := \sum_{i=1}^{N} D_t(i) h_t(x_i, y_i).$$

(ii) Upper bound on the pseudo-loss error:
Now we plug $\alpha_t$ in (12) and get

$$plerr \leq \prod_{t=1}^{T} \left( r_t \left(\frac{1-r_t}{r_t(|Y|-1)}\right)^{(|Y|-1)/|Y|} + (1-r_t)\left(\frac{r_t(|Y|-1)}{1-r_t}\right)^{1/|Y|}\right). \qquad (13)$$

(iii) Stopping criterion:
As expected for $r_t = 1/|Y|$ the corresponding factor is 1. The stopping criterion $U_t \leq 0$ can be directly translated into $r_t \geq 1/|Y|$. Looking at the first and second derivative of the bound one can easily verify that it has a unique maximum at $r_t = 1/|Y|$. Therefore, the bound drops as long as $r_t > 1/|Y|$. Note again that since $r_t = 1/|Y|$ is a unique maximum we get a formal decrease of the bound even when $r_t > 1/|Y|$.
(iv) Update rule:
Now we simplify the update rule using (11) and insert the new choice of $\alpha_t$ and get

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \mathrm{e}^{-\tilde{\alpha}_t(h_t(x_i,y_i)-1/|Y|)} \quad \text{for} \quad \tilde{\alpha}_t := \ln\left(\frac{(|Y|-1)r_t}{1-r_t}\right)$$

Also the goal of the base classifier can be simplified, because maximizing $U_t$ is equivalent to maximizing $r_t$.

We will see in the next section, that the resulting algorithm is a special case of the algorithm BoostMA of the next chapter with $c = 1/|Y|$.

## 3. BoostMA

The aim behind the algorithm BoostMA was to find a simple modification of AdaBoost.M1 in order to make it work for weak base classifiers. The original idea was influenced by a frequently used argument for the explanation of ensemble methods. Assuming that the individual classifiers are uncorrelated, majority voting of an ensemble of classifiers should lead to better results than using one individual classifier. This explanation suggests that the weight of classifiers that perform better than random guessing should be positive. This is not the case for AdaBoost.M1. In AdaBoost.M1 the weight of a base classifier $\alpha$ is a function of the error rate, so we tried to modify this function so that it gets positive, if the error rate is less than the error rate of random guessing. The resulting classifier AdaBoost.M1W showed good results in experiments (Eibl and Pfeiffer, 2002). Further theoretical considerations led to the more elaborate algorithm which we call BoostMA which uses confidence-rated classifiers and also compares the base classifier with the uninformative rule.

In AdaBoost.M2, the sampling weights are increased for instances for which the pseudo-loss exceeds 1/2. Here we want to increase the weights for instances, where the base classifier $h$ :

$X \times Y \to [0,1]$ performs worse than the uninformative or what we call the maxlabel rule. The maxlabel rule labels each instance as the most frequent label. As a confidence-rated classifier, the uninformative rule has the form

$$\text{maxlabel rule} : X \times Y \to [0,1] : h(x,y) := \frac{N_y}{N},$$

where $N_y$ is the number of instances in the training set with label $y$. So it seems natural to investigate a modification where the update of the sampling distribution has the form

$$D_{t+1}(i) = D_t(i)\frac{e^{-\alpha_t(h_t(x_i,y_i)-c)}}{Z_t}, \text{ with } Z_t := \sum_{i=1}^{N} D_t(i)e^{-\alpha_t(h_t(x_i,y_i)-c)},$$

where $c$ measures the performance of the uninformative rule. Later we will set

$$c := \sum_{y \in Y} \left(\frac{N_y}{N}\right)^2$$

and justify this setting. But up to that point we let the choice of $c$ open and just require $c \in (0,1)$. We now define a performance measure which plays the same role as the pseudo-loss error.

**Definition 1** *Let $c$ be a number in $(0,1)$. A classifier $f : X \times Y \to [0,1]$ makes a maxlabel error in classifying an example $x$ with label $k$, if*

$$f(x,k) < c.$$

*The maxlabel error for the training set is called mxerr:*

$$mxerr := \frac{1}{N}\sum_{i=1}^{N} I\left(f(x_i,y_i) < c\right).$$

The maxlabel error counts the proportion of elements of the training set for which the confidence $f(x,k)$ in the right label is smaller than $c$. The number $c$ must be chosen in advance. The higher $c$ is, the higher is the maxlabel error for the *same* classifier $f$; therefore to get a weak error measure we set $c$ very low. For BoostMA we choose $c$ as the accuracy for the uninformative rule. When we use decision stumps as base classifiers we have the property $h(x,y) \in [0,1]$. By normalizing $\alpha_1,\dots,\alpha_T$, so that they sum to one, we ensure $f(x,y) \in [0,1]$ (Equation 15).

We present the algorithm BoostMA in Figure 4 and in what follows we justify and establish some properties about it. As for GrPloss the modus operandi consists of finding an upper bound on *mxerr* and minimizing the bound with respect to $\alpha$.

(i) Bound of *mxerr* in terms of the normalization constants $Z_t$:

Similar to the calculations used to bound the pseudo-loss error we begin by bounding *mxerr* in terms of the normalization constants $Z_t$: We have

$$\begin{aligned} 1 &= \sum_i D_{t+1}(i) = \sum_i D_t(i)\frac{e^{-\alpha_t(h_t(x_i,y_i)-c)}}{Z_t} = \dots \\ &= \frac{1}{\prod_s Z_s}\frac{1}{N}\sum_i \prod_{s=1}^{t} e^{-\alpha_s(h_s(x_i,y_i)-c)} = \frac{1}{\prod_s Z_s}\frac{1}{N}\sum_i e^{-(f(x_i,y_i)-c\sum_s \alpha_s)}. \end{aligned}$$

```
Input: training set $S = \{(x_1,y_1),\ldots,(x_N,y_N); x_i \in X, y_i \in Y\}$,
    $Y = \{1,\ldots,|Y|\}$, weak classification algorithm of the form $h : X \times Y \to [0,1]$.
    Optionally $T$: number of boosting rounds

Initialization: $D_1(i) = \frac{1}{N}$.

For $t = 1,\ldots,T$:

    • Train the weak classification algorithm $h_t$ with distribution $D_t$, where $h_t$ should maximize
          $r_t = \sum_i D_t(i) h_t(x_i,y_i)$

    • If $r_t \leq c$: goto output with $T := t-1$

    • Set
          $\alpha_t = \ln\left(\frac{(1-c)r_t}{c(1-r_t)}\right)$.

    • Update D:
          $D_{t+1}(i) = D_t(i)\frac{e^{-\alpha_t(h_t(x_i,y_i)-c)}}{Z_t}$.
      where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ is a distribution)

Output: Normalize $\alpha_1,\ldots,\alpha_T$ and set the final classifier $H(x)$:
    $H(x) = \arg\max_{y\in Y} f(x,y) = \arg\max_{y\in Y}\left(\sum_{t=1}^{T}\alpha_t h_t(x,y)\right)$
```

Figure 4: Algorithm BoostMA

So we get

$$\prod_t Z_t = \frac{1}{N}\sum_i e^{-(f(x_i,y_i) - c\sum_t \alpha_t)}. \tag{14}$$

Using

$$\frac{f(x_i,y_i)}{\sum_t \alpha_t} < c \quad \Rightarrow \quad e^{-(f(x_i,y_i) - c\sum_t \alpha_t)} > 1 \tag{15}$$

and (14) we get

$$mxerr \leq \prod_t Z_t. \tag{16}$$

(ii) Choice of $\alpha_t$:
Now we bound $\prod_t Z_t$ and then we minimize it, which leads us to the choice of $\alpha_t$. First we use the definition of $Z_t$ and get

$$\prod_t Z_t = \prod_t \left(\sum_i D_t(i) e^{-\alpha_t(h_t(x_i,y_i)-c)}\right). \tag{17}$$

Now we use the convexity of $e^{-\alpha_t(h_t(x_i,y_i)-c)}$ for $h_t(x_i,y_i)$ between 0 and 1 and the definition

$$r_t := \sum_i D_t(i) h_t(x_i,y_i)$$

and get

$$\begin{aligned} mxerr &\leq \prod_t \sum_i D_t(i) \left( h_t(x_i,y_i) e^{-\alpha_t(1-c)} + (1-h_t(x_i,y_i)) e^{\alpha_t c} \right) \\ &= \prod_t \left( r_t e^{-\alpha_t(1-c)} + (1-r_t) e^{\alpha_t c} \right). \end{aligned}$$

We minimize this by setting the first derivative with respect to $\alpha_t$ to zero, which leads to

$$\alpha_t = \ln\left( \frac{(1-c)r_t}{c(1-r_t)} \right).$$

(iii) First bound on *mxerr*:
To get the bound on *mxerr* we substitute our choice for $\alpha_t$ in (17) and get

$$mxerr \leq \prod_t \left( \left( \frac{(1-c)r_t}{c(1-r_t)} \right)^c \sum_i D_t(i) \left( \frac{c(1-r_t)}{(1-c)r_t} \right)^{h_t(x_i,y_i)} \right). \tag{18}$$

Now we bound the term $\left(\frac{c(1-r_t)}{(1-c)r_t}\right)^{h_t(x_i,y_i)}$ by use of the inequality

$$x^a \leq 1 - a + ax \quad \text{for } x \geq 0 \text{ and } a \in [0,1],$$

which comes from the convexity of $x^a$ for $a$ between 0 and 1 and get

$$\left( \frac{c(1-r_t)}{(1-c)r_t} \right)^{h_t(x_i,y_i)} \leq 1 - h_t(x_i,y_i) + h_t(x_i,y_i) \frac{c(1-r_t)}{(1-c)r_t}.$$

Substitution in (18) and simplifications lead to

$$mxerr \leq \prod_t \left( \frac{r_t^c (1-r_t)^{1-c}}{(1-c)^{1-c} c^c} \right). \tag{19}$$

The factors of this bound are symmetric around $r_t = c$ and take their maximum of 1 there. Therefore if $r_t > c$ is valid the bound on *mxerr* decreases.
(iv) Exponential decrease of *mxerr*:
To prove the second bound we set $r_t = c + \delta$ with $\delta \in (0, 1-c)$ and rewrite (19) as

$$mxerr \leq \prod_t \left( 1 - \frac{\delta}{1-c} \right)^{1-c} \left( 1 + \frac{\delta}{c} \right)^c.$$

We can bound both terms using the binomial series: all terms of the series of the first term are negative, we stop after the terms of first order and get

$$\left( 1 - \frac{\delta}{1-c} \right)^{1-c} \leq 1 - \delta.$$

The series of the second term has both positive and negative terms, we stop after the positive term of first order and get

$$\left(1+\frac{\delta}{c}\right)^c \leq 1+\delta.$$

Thus

$$mxerr \leq \prod_t (1-\delta^2).$$

Using $1+x \leq e^x$ for $x \leq 0$ leads to

$$mxerr \leq e^{-\delta^2 T}.$$

We summarize the foregoing argument as a theorem.

**Theorem 3** *If all base classifiers $h_t$ with $h_t(x,y) \in [0,1]$ fulfill*

$$r_t := \sum_i D_t(i) h_t(x_i,y_i) \geq c+\delta$$

*for $\delta \in (0,1-c)$ (and the condition $c \in (0,1)$) then the maxlabel error of the training set for the algorithm in Figure 4 fulfills*

$$mxerr \leq \prod_t \left(\frac{r_t^c\,(1-r_t)^{1-c}}{(1-c)^{1-c}\,c^c}\right) \leq e^{-\delta^2 T}. \tag{20}$$

Remarks: 1.) Choice of $c$ for BoostMA: since we use confidence-rated base classification algorithms we choose the training accuracy for the confidence-rated uninformative rule for $c$, which leads to

$$c = \frac{1}{N}\sum_{i=1}^{N} \frac{N_{y_i}}{N} = \frac{1}{N}\sum_y \sum_{i;y_i=y} \frac{N_y}{N} = \sum_{y\in Y}\left(\frac{N_y}{N}\right)^2. \tag{21}$$

2.) For base classifiers with the normalization property (10) we can get a simpler expression for the pseudo-loss error. From

$$\sum_{y\neq k} f(x,y) = \sum_{y\neq k}\sum_t \alpha_t h_t(x,y) = \sum_t \alpha_t (1-h_t(x,k)) = \sum_t \alpha_t - f(x,k)$$

we get

$$f(x,k) < \frac{1}{|Y|-1}\sum_{y\neq k} f(x,y) \;\Leftrightarrow\; \frac{f(x,k)}{\sum_t \alpha_t} < \frac{1}{|Y|}. \tag{22}$$

That means that if we choose $c = 1/|Y|$ for BoostMA the maxlabel error is the same as the pseudo-loss error. For the choice (21) of $c$ this is the case when the group proportions are balanced, because then

$$c = \sum_{y\in Y}\left(\frac{N_y}{N}\right)^2 = \sum_{y\in Y}\left(\frac{1}{|Y|}\right)^2 = |Y|\frac{1}{|Y|^2} = \frac{1}{|Y|}.$$

For this choice of $c$ the update rule of the sampling distribution for BoostMA gets

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \mathrm{e}^{-\alpha_t (h_t(x_i,y_i) - 1/|Y|)} \quad \text{and} \quad \alpha_t = \ln\left(\frac{(|Y|-1) r_t}{1-r_t}\right),$$

which is just the same as the update rule GrPloss for decision stumps. Summarizing these two results we can say that for base classifiers with the normalization property, the choice (21) for $c$ of BoostMA and data sets with balanced labels, the two algorithms GrPloss and BoostMA and their error measures are the same.
3.) In contrast to GrPloss the algorithm does not change when the base classifier additionally fulfills the normalization property (10) because the algorithm only uses $h_t(x_i,y_i)$.

## 4. Experiments

In our experiments we focused on the derived bounds and the practical performance of the algorithms.

### 4.1 Experimental Setup

To check the performance of the algorithms experimentally we performed experiments with 12 data sets, most of which are available from the UCI repository (Blake and Merz, 1998). To get reliable estimates for the expected error rate we used relatively large data sets consisting of about 1000 cases or more. The expected classification error was estimated either by a test error rate or 10-fold cross-validation. A short overview of the data sets is given in Table 1.

| Database | N | ♯ Labels | ♯ Variables | Error Estimation | Labels |
|---|---|---|---|---|---|
| car * | 1728 | 4 | 6 | 10-CV | unbalanced |
| digitbreiman | 5000 | 10 | 7 | test error | balanced |
| letter | 20000 | 26 | 16 | test error | balanced |
| nursery * | 12960 | 4 | 8 | 10-CV | unbalanced |
| optdigits | 5620 | 10 | 64 | test error | balanced |
| pendigits | 10992 | 10 | 16 | test error | balanced |
| satimage * | 6435 | 6 | 34 | test error | unbalanced |
| segmentation | 2310 | 7 | 19 | 10-CV | balanced |
| waveform | 5000 | 3 | 21 | test error | balanced |
| vehicle | 846 | 4 | 18 | 10-CV | balanced |
| vowel | 990 | 11 | 10 | test error | balanced |
| yeast * | 1484 | 10 | 9 | 10-CV | unbalanced |

Table 1: Properties of the databases

For all algorithms we used boosting by resampling with decision stumps as base classifiers. We used AdaBoost.M2 by Freund and Schapire (1997), BoostMA with $c = \sum_{y\in Y}(N_y/N)^2$ and the algorithm GrPloss for decision stumps of Section 2.3 which corresponds to BoostMA with $c = 1/|Y|$. For only four databases the proportions of the labels are significantly unbalanced so that GrPloss and BoostMA should have greater differences only for these four databases (marked with a *). As discussed by Bauer and Kohavi (1999) the individual sampling weights $D_t(i)$ can get very small. Similar to was done there, we set the weights of instances which were below $10^{-10}$ to $10^{-10}$.

We also set a maximum number of 2000 boosting rounds to stop the algorithm if the stopping criterion is not satisfied.

### 4.2 Results

The experiments have two main goals. From the theoretical point of view one is interested in the derived bounds. For the practical use of the algorithms, it is important to look at the training and test error rates and the speed of the algorithms.

#### 4.2.1 DERIVED BOUNDS

First we look at the bounds on the error measures. For the algorithm AdaBoost.M2, Freund and Schapire (1997) derived the upper bound

$$(|Y|-1)2^{T-1}\prod_{t=1}^{T}\sqrt{\varepsilon_t(1-\varepsilon_t)} \tag{23}$$

on the training error. We have three different bounds on the pseudo-loss error of Grploss: the term

$$\prod_t Z_t \tag{24}$$

which was derived in the first part of the proof of Theorem 2, the tighter bound (9) of Theorem 2 and the bound (13) for the special case of decision stumps as base classifiers. In Section 3, we derived two upper bounds on the maxlabel error for BoostMA: term (24) and the tighter bound (20) of Theorem 3.

For all algorithms their respective bounds hold for all time steps and for all data sets. Bound (23) on the training error of AdaBoost.M2 is very loose – it even exceeds 1 for eight of the 12 data sets, which is possible due to the factor $|Y|-1$ (Table 2). In contrast to the bound on the training error of AdaBoost.M2, the bounds on the pseudo-loss error of GrPloss and the maxlabel error of BoostMA are below 1 for all data sets and all boosting rounds. In that sense, they are tighter than the bounds on the training error of AdaBoost.M2.

As expected, bound (13) derived for the special case of decision stumps as base classifiers on the pseudo-loss error is smaller than bound (9) of Theorem 2 which doesn't use the normalization property (10) of the decision stumps.

For both GrPloss and BoostMA, bound (24) is the smallest bound since it contains the fewest approximations. For BoostMA, term (24) is a bound on the maxlabel error and for GrPloss term (24) is a bound on the pseudo-loss error. For unbalanced data sets, the maxlabel error is the "harder" error measure than the pseudo-loss error, so for these data sets bound (24) is higher for BoostMA than for GrPloss. For balanced data sets the maxlabel error and the pseudo-loss error are the same. Bound (9) for GrPloss is higher for these data sets than bound (20) of BoostMA. This suggests that bound (9) for GrPloss could be improved by more sophisticated calculations.

#### 4.2.2 COMPARISON OF THE ALGORITHMS

Now we wish to compare the algorithms with one another. Since GrPloss and BoostMA differ only for the four unbalanced data sets, we focus on the comparison of GrPloss with AdaBoost.M2 and make only a short comparison of GrPloss and BoostMA. For the subsequent comparisons we take

| | AdaBoost.M2 training error [%] | | GrPloss pseudo-loss error [%] | | | | BoostMA maxlabel error [%] | | |
|---|---|---|---|---|---|---|---|---|---|
| Database | trerr | BD23 | plerr | BD24 | BD13 | BD9 | mxerr | BD24 | BD20 |
| car * | 0 | 33.9 | 0 | 3.4 | 11.1 | 31.9 | 7.8 | 63.1 | 71.9 |
| digitbreiman | 25.5 | 327.4 | 0.5 | 3.7 | 19.9 | 81.0 | 1.0 | 11.9 | 35.6 |
| letter | 46.1 | 1013.1 | 0.4 | 7.2 | 27.8 | 94.3 | 0.4 | 8.1 | 29.5 |
| nursery * | 14.2 | 78.7 | 0 | 0 | 0.5 | 11.1 | 0 | 0.8 | 7.6 |
| optdigits | 0 | 421.1 | 0 | 0 | 2.0 | 51.4 | 0 | 0 | 0.1 |
| pendigits | 13.8 | 190.2 | 0 | 0 | 0.1 | 42.6 | 0 | 0 | 0.1 |
| satimage * | 15.9 | 118.5 | 0.1 | 1.8 | 13.2 | 62.3 | 3.8 | 26.0 | 50.1 |
| segmentation | 7.5 | 96.2 | 0 | 0.4 | 2.8 | 30.5 | 0 | 0.4 | 3.5 |
| vehicle | 26.5 | 101.2 | 0.1 | 2.8 | 14.7 | 50.0 | 0.1 | 3.3 | 16.5 |
| vowel | 30.9 | 273.8 | 0 | 0 | 0.1 | 40.4 | 0 | 0.1 | 3.0 |
| waveform | 12.5 | 48.4 | 0 | 0.5 | 6.3 | 23.3 | 0 | 0.4 | 6.0 |
| yeast * | 60.2 | 365.0 | 0.4 | 6.6 | 26.0 | 83.6 | 49.2 | 99.2 | 99.6 |

Table 2: Performance measures and their bounds in percent at the boosting round with minimal training error. trerr, BD23: training error of AdaBoost and its bound (23); plerr, BD24 ,BD13 ,BD9: pseudo-loss error of GrPloss and its bounds (24), (13) and (9); mxerr, BD24, BD20: maxlabel error of BoostMA and its bounds (24) and (20).

all error rates at the boosting round with minimal training error rate as was done by Eibl and Pfeiffer (2002).

First we look at the minimum achieved training and test error rates. The theory suggests AdaBoost.M2 to work best in minimizing the training error. However, GrPloss seems to have roughly the same performance with maybe AdaBoost.M2 leading by a slight edge (Tables 3 and 4, Figure 5). The difference in the training error mainly carries over to the difference in the test error. Only for the data sets digitbreiman and yeast do the training and the test error favor different algorithms (Table 4). Both the training and the test error favor AdaBoost.M2 for six data sets and GrPloss for four data sets with two draws (Table 4).

While GrPloss and AdaBoost.M2 were quite close for the training and test error rates, this is not the case for the pseudo-loss error. Here, GrPloss is the clear winner against AdaBoost.M1 with eight wins and four draws (Table 4). The reason for this might be the fact that bound (13) on the pseudo-loss error of GrPloss is tighter than bound (23) on the training error of AdaBoost.M2 (Table 2). For the data set nursery, bound (13) on the pseudo-loss error of GrPloss (0.5%) is smaller than the pseudo-loss error of AdaBoost.M2 (1.9%). So for this data set, bound (13) can explain the superiority of GrPloss in minimizing the pseudo-loss error.

Due to the fact that only four data sets are significantly unbalanced, it is not easy to assess the difference between GrPloss and BoostMA. GrPloss seems to have a lead regarding the training and test error rates (Tables 3 and 5). For the experiments, the constant $c$ of BoostMA was chosen as the training accuracy for the confidence-rated uninformative rule (21). For the unbalanced data sets, this $c$ exceeds $1/|Y|$, which is the corresponding choice for GrPloss (22). A change of $c$ – maybe even adaptively during the run – could possibly improve the performance. We wish to make further

| | training error | | | test error | | |
|---|---|---|---|---|---|---|
| Database | AdaM2 | GrPloss | BoostMA | AdaM2 | GrPloss | BoostMA |
| car * | 0 | 0 | **7.75** | 0 | 0 | **7.75** |
| digitbreiman | 25.49 | 25.63 | 25.63 | 27.51 | 27.13 | 27.38 |
| letter | **46.07** | 40.02 | 40.14 | **47.18** | 41.70 | 41.70 |
| nursery * | *14.16* | 12.37 | 12.63 | *14.27* | 12.35 | 12.67 |
| optdigits | 0 | 0 | 0 | 0 | 0 | 0 |
| pendigits | 13.82 | *17.17* | *17.20* | 18.61 | *20.44* | *20.75* |
| satimage * | 15.85 | 15.69 | 16.87 | 18.25 | 17.80 | 18.90 |
| segmentation | 7.49 | *9.05* | 8.90 | 8.40 | 9.31 | 9.48 |
| vehicle | 26.46 | *30.15* | *30.19* | 35.34 | *38.16* | *36.87* |
| vowel | 30.87 | **41.67** | **42.23** | 54.33 | **67.32** | **67.32** |
| waveform | 12.45 | *14.55* | *14.49* | 16.63 | *18.17* | 17.72 |
| yeast * | 60.18 | 59.31 | 60.61 | 60.65 | 61.99 | *62.47* |

Table 3: Training and test error at the boosting round with minimal training error; bold and italic numbers correspond to high(>5%) and medium(>1.5%) differences to the smallest of the three error rates

| | GrPloss vs. AdaM2 | | | |
|---|---|---|---|---|
| Database | trerr | testerr | plerr | speed |
| car * | o | o | o | + |
| digitbreiman | - | + | + | + |
| letter | + | + | + | + |
| nursery * | + | + | + | + |
| optdigits | o | o | o | - |
| pendigits | - | - | + | + |
| satimage * | + | + | + | + |
| segmentation | - | - | o | + |
| vehicle | - | - | + | + |
| vowel | - | - | o | + |
| waveform | - | - | + | + |
| yeast * | + | - | + | - |
| total | 4-2-6 | 4-2-6 | 8-4-0 | 10-0-2 |

Table 4: Comparison of GrPloss with AdaBoost.M2: win-loss-table for the training error, test error, pseudo-loss error and speed of the algorithm (+/o/-: win/draw/loss for GrPloss)

investigations about a systematic choice of $c$ for BoostMA. Both algorithms seem to be better in the minimization of their corresponding error measure (Table 5). The small differences between GrPloss and BoostMA occurring for the nearly balanced data sets can not only come from the small

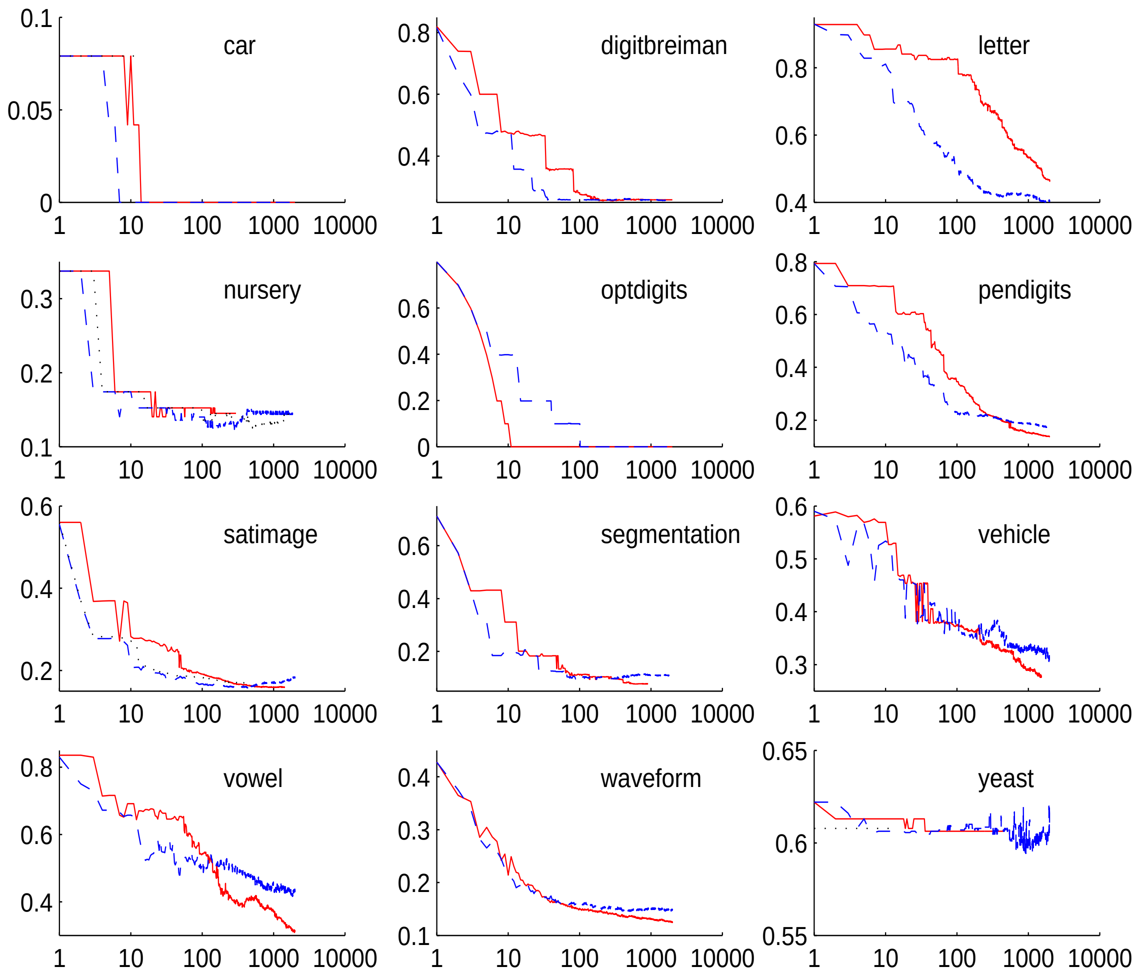

Figure 5: Training error curves: solid: AdaBoost.M2, dashed: GrPloss, dotted: BoostMA

differences in the group proportions, but also from differences in the resampling step and from the partition of a balanced data set into unbalanced training and test sets during cross-validation.

Performing a boosting algorithm is a time consuming procedure, so the speed of an algorithm is an important topic. Figure 5 indicates that the training error rate of GrPloss is decreasing faster than the training error rate of AdaBoost.M2. To be more precise, we look at the number of boosting rounds needed to achieve 90% of the total decrease of the training error rate. For 10 of the 12 data sets, AdaBoost.M2 needs more boosting rounds than GrPloss, so GrPloss seems to lead to a faster decrease in the training error rate (Table 4). Besides the number of boosting rounds, the time for the algorithm is also heavily influenced by the time needed to construct a base classifier. In our program, it took longer to construct a base classifier for AdaBoost.M2 because the minimization of the pseudo-loss which is required for AdaBoost.M2 is not as straightforward as the maximization of $r_t$ required for GrPloss and BoostMA. However, the time needed to construct a base classifier strongly depends on programming details, so we do not wish to over-emphasize this aspect.

| | GrPloss vs. BoostMA | | | | |
|---|---|---|---|---|---|
| Database | trerr | testerr | plerr | mxerr | speed |
| car * | + | + | + | o | - |
| nursery * | + | + | o | o | + |
| satimage * | + | + | + | - | o |
| yeast * | + | + | + | - | - |
| total | 4-0-0 | 4-0-0 | 3-1-0 | 0-2-2 | 1-0-2 |

Table 5: Comparison of GrPloss with BoostMA for the unbalanced data sets: win-loss-table for the training error, test error, pseudo-loss error, maxlabel error and speed of the algorithm (+/o/-: win/draw/loss for GrPloss)

## 5. Conclusion

We proposed two new algorithms GrPloss and BoostMA for multiclass problems with weak base classifiers. The algorithms are designed to minimize the pseudo-loss error and the maxlabel error respectively. Both have the advantage that the base classifier minimizes the confidence-rated error instead of the pseudo-loss. This makes them easier to use with already existing base classifiers. Also the changes to AdaBoost.M1 are very small, so one can easily get the new algorithms by only slight adaption of the code of AdaBoost.M1. Although they are not designed to minimize the training error, they have comparable performance as AdaBoost.M2 in our experiments. As a second advantage, they converge faster than AdaBoost.M2. AdaBoost.M2 minimizes a bound on the training error. The other two algorithms have the disadvantage of minimizing bounds on performance measures which are not connected so strongly to the expected error. However the bounds on the performance measures of GrPloss and BoostMA are tighter than the bound on the training error of AdaBoost.M2, which seems to compensate for this disadvantage.

## References

Erin L. Allwein, Robert E. Schapire, Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Machine Learning*, 1:113–141, 2000.

Eric Bauer, Ron Kohavi. An empirical comparison of voting classification algorithms: bagging, boosting and variants. *Machine Learning*, 36:105–139, 1999.

Catherine Blake, Christopher J. Merz. UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998

Thomas G. Dietterrich, Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2:263–286, 1995.

Günther Eibl, Karl–Peter Pfeiffer. Analysis of the performance of AdaBoost.M2 for the simulated digit-recognition-example. *Machine Learning: Proceedings of the Twelfth European Conference*, 109–120, 2001.

Günther Eibl, Karl–Peter Pfeiffer. How to make AdaBoost.M1 work for weak classifiers by changing only one line of the code. *Machine Learning: Proceedings of the Thirteenth European Conference*, 109–120, 2002.

Yoav Freund, Robert E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference,* 148–56, 1996.

Yoav Freund, Robert E. Schapire. A decision-theoretic generalization of online-learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Venkatesan Guruswami, Amit Sahai. Multiclass learning, boosting, and error-correcting codes. *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 145–155, 1999.

Llew Mason, Peter L. Bartlett, Jonathan Baxter. Direct optimization of margins improves generalization in combined classifiers. *Proceedings of NIPS 98*, 288–294, 1998.

Llew Mason, Peter L. Bartlett, Jonathan Baxter, Marcus Frean. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers*, 221–246, 1999.

Ross Quinlan. Bagging, boosting, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 725–730, 1996.

Gunnar Rätsch, Bernhard Schölkopf, Alex J. Smola, Sebastian Mika, Takashi Onoda, Klaus R. Müller. Robust ensemble learning. *Advances in Large Margin Classifiers*, 207–220, 2000a.

Gunnar Rätsch, Takashi Onoda, Klaus R. Müller. Soft margins for AdaBoost. *Machine Learning* 42(3):287–320, 2000b.

Robert E. Schapire, Yoav Freund, Peter L. Bartlett, Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

Robert E. Schapire, Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning* 37:297-336, 1999.