A Smoothed Boosting Algorithm Using Probabilistic Output Codes

Rong Jin RONGJIN@CSE.MSU.EDU
Dept. of Computer Science and Engineering, Michigan State University, MI 48824, USA

Jian Zhang JIAN.ZHANG@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, PA 15213, USA

Abstract

AdaBoost.OC has shown to be an effective method in boosting "weak" binary classifiers for multi-class learning. It employs the Error Correcting Output Code (ECOC) method to convert a multi-class learning problem into a set of binary classification problems, and applies the AdaBoost algorithm to solve them efficiently. In this paper, we propose a new boosting algorithm that improves the AdaBoost.OC algorithm in two aspects: 1) It introduces a smoothing mechanism into the boosting algorithm to alleviate the potential overfitting problem with the AdaBoost algorithm, and 2) It introduces a probabilistic coding scheme to generate binary codes for multiple classes such that training errors can be efficiently reduced. Empirical studies with seven UCI datasets have indicated that the proposed boosting algorithm is more robust and effective than the AdaBoost.OC algorithm for multi-class learning.

1. Introduction

AdaBoost has been shown to be an effective method for improving the classification accuracy of weak classifiers (Freund & Schapire, 1996; Freund & Schapire, 1995; Schapire, 1997; Grove & Schuurmans, 1998; Bauer & Kohavi, 1999; Ratsch et al., 1999; Ratsch et al., 2000; Schapire, 1999; Lebanon & Lafferty, 2001; Jin et al., 2003). It works by repeatedly running a weak learner on various training examples sampled from the original training pool, and combining multiple instances of the week learner into a single composite classifier. AdaBoost.OC (Schapire, 1997) extends the AdaBoost algorithm to multi-class learning by combining it with the method of Error-Correcting Output Code (ECOC) (Dietterich & Bakiri, 1995). It reduces a multi-class learning problem into a set of binary-class learning problems by encoding each class into a binary vector. A binary classifier is learned for every bit in the binary representation of classes, and is linearly combined to make predictions for test examples. Empirical studies have shown that AdaBoost.OC is effective in boosting binary classifiers for multi-class learning.

Despite its success, there are two problems with AdaBoost.OC that have not been well addressed in the past:

- *The overfitting problem* Previous studies have shown that the AdaBoost algorithm tends to overfit training examples when they are noisy (Ratsch et al., 1999; Dietterich, 2000; Jin et al., 2003; Jiang, 2001). Since AdaBoost.OC is based on the AdaBoost algorithm, we expect that it will suffer from the overfitting problem as AdaBoost does. Although there have been many studies on alleviating the overfitting problem with AdaBoost (Ratsch et al., 1999; Schapire, 1999; Ratsch et al., 2000; Lebanon & Lafferty, 2001; Jin et al., 2003; Ratsch et al., 1998), no research has been done for its multi-class version (i.e., AdaBoost.OC).
- *Effective coding schemes.* AdaBoost.OC employs the ECOC method to reduce a multi-class learning problem into a sequence of binary classification problems. Hence, the choice of coding schemes can have a significant impact on the performance of AdaBoost.OC. In (Schapire, 1997), the author examined several coding schemes for AdaBoost.OC. But, none of them provided no-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

ticeable improvement over a simple random code.

In this paper, we present a new boosting algorithm that explicitly addresses the above two problems:

- To alleviate the overfitting problem, a smoothing mechanism is introduced into the boosting algorithm. In particular, a *bounded* weight is assigned to each individual example during every iteration. This is in contrast to AdaBoost.OC, where weights assigned to individual examples are unbounded and noisy data points can be overly emphasized during the iterations when they are assigned with extremely large weights.
- To convert a multi-class learning problem into a set of binary classification problems, a probabilistic coding scheme is introduced to generate binary codes for multiple classes. Compared to a simple random code, the new coding scheme is advantageous in that it automatically adapts to the performance of basis classifiers and therefore is more efficient in reducing training errors. Unlike the deterministic coding schemes (Schapire, 1997) whose success heavily depends on their assumption of basis classifiers, the probabilistic coding scheme is robust even when its assumption about basis classifiers is violated.

The rest of this paper is arranged as follows: Section 2 reviews the related work; Section 3 describes the smoothed boosting algorithm for multi-class learning and the probabilistic output code; Section 4 presents the experiment results; Section 5 concludes this paper with the future work.

2. Related Work

The most relevant work is the AdaBoost.OC algorithm (Schapire, 1997), which combines the AdaBoost algorithm with the ECOC method for multi-class learning. Figure 1 describes the detailed steps of the AdaBoost.OC algorithm. It iteratively refines a multi-class classifier $H(x)$. At the t-th iteration, a coding function $f_t(y)$ is first generated to map any class label y into the binary set $\{-1, +1\}$. Based on the coding function $f_t(y)$, a binary classifier $h_t(x)$ is trained over examples that are weighted by the distribution $D_t(i)$. The learned classifier $h_t(x)$ is then linearly combined with the binary classifiers that are learned in previous iterations to make prediction for test examples. The combination parameter α_t is computed based on the classification error ϵ_t. Finally, the weight distribution $D_t(i, y)$ is updated using the combination parameter α_t.

Given: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
Initialize $D_1(i, y) = [y \neq y_i]/(m(|\mathcal{Y}| - 1))$

For $t = 1, ..., T$

1. Generate coding function $f_t(y) : \mathcal{Y} \rightarrow \{0, 1\}$
2. Let $U_t = \sum_{i=1}^{m} \sum_{y \in \mathcal{Y}} D_t(i, y)[f_t(y) \neq f_t(y_i)]$, and $D_t(i) = \sum_{y \in \mathcal{Y}} D_t(i, y)[f_t(y) \neq f_t(y_i)]/U_t$
3. Train a weak classifier $h_t(x) : \mathcal{X} \rightarrow \{0, 1\}$ on examples weighted by D_t
4. Let $\epsilon_t = \sum_{i=1}^{m} D_t(i)[f_t(y_i) \neq h_t(x_i)]$
5. Let $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6. Update the weight distribution

$$D_{t+1}(i, y) = \frac{D_t(i, y) \exp(\alpha_t([f_t(y_i) \neq h_t(x_i)] + [f_t(y) = h_t(x_i)]))}{Z_{t+1}}$$

where Z_{t+1} is a normalization factor (chosen so that $D_{t+1}(i, y)$ is sum to 1).

Output: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \alpha_t [h_t(x) = f_t(y)]$

Figure 1. Description of the AdaBoost.OC algorithm

One problem with AdaBoost.OC is that it could overfit the training data. Note that weight $D_t(i, y)$ in Figure 1 can be arbitrarily close to 1. Given a data point is noisy, it may be misclassified multiple times. As a result, its weight can be considerably larger than the weights of other data points, which could lead AdaBoost.OC to overfit training examples.

Another problem with AdaBoost.OC is how to determine the coding function $f_t(y)$ in each iteration. In (Schapire, 1997), the author proved that the training error of AdaBoost.OC is upper bounded by $\prod_t \sqrt{1 - 4(\gamma_t U_t)^2}$ where $\gamma_t = 1/2 - \epsilon_t$, and the expression for U_t and ϵ_t are presented in Figure 1. Based on this analysis, the author suggested that a good coding function $f_t(y)$ should maximize U_t. However, this analysis ignores the fact that both factors U_t and γ_t are influenced by the choice of the coding function $f_t(y)$. As a result, a coding function $f_t(y)$ that maximizes U_t can result in a small value for γ_t, which may lead to a loose upper bound for the training error.

This work is also related to previous studies on preventing AdaBoost from overfitting training examples. In the past, many approaches have been proposed, including the smoothing approaches (Schapire, 1999; Jin et al., 2003), the regularization approaches (Ratsch

et al., 1999; Lebanon & Lafferty, 2001), and the maximum margin approaches (Grove & Schuurmans, 1998; Ratsch et al., 2000). However, most of them target on binary-class classification problems, which may not be the most effective solution for multi-class learning.

3. A Smoothed Boosting Algorithm Using the Probabilistic Output Code

In this section, we first describe a smoothed boosting algorithm that combines multiple binary classifiers for multi-class learning, followed by the description of a probabilistic coding scheme that automatically generates coding functions during the iterations of boosting.

3.1. A Smoothed Boosting Algorithm For Multi-class Learning

Similar to AdaBoost.OC, the proposed smoothed boosting algorithm applies an iterative procedure to boost binary classifiers for multi-class learning. In each iteration, a new coding function is first generated to map multiple classes into a binary set. Then, a binary classifier is learned based on the given coding function. The final multi-class classifier is a linear combination of the binary-class classifiers that are learned in the iterations.

Let the code functions for the first T iterations denoted by $\vec{f}_T(y) = (f_1(y), ..., f_T(y))$ where each coding function $f_t(y) : \mathcal{Y} \rightarrow \{-1, +1\}$. Let $\vec{g}_T(x) = (\alpha_1 h_1(x), ..., \alpha_T h_T(x))$ be the weighted classifiers obtained in the first T iterations where $h_t(x) : \mathcal{X} \rightarrow \{-1, +1\}$ and α_t is a weight for $h_t(x)$. Using the above notations, the combined classifier $H(x)$ for the first T iterations is rewritten as:

$$H_T(x) = \arg\max_{y \in \mathcal{Y}} \vec{f}_T(y) \cdot \vec{g}_T(x) \tag{1}$$

Then, the training error at the T-th iteration err_T is written as:

$$\begin{aligned} err_T &= \frac{1}{m}\sum_{i=1}^{m}[H_T(x_i) \neq y_i] \\ &= \frac{1}{m}\sum_{i=1}^{m} I\left(\max_{y \neq y_i}(\vec{f}_T(y) - \vec{f}_T(y_i)) \cdot \vec{g}_T(x_i)\right) \end{aligned}$$

where $I(x)$ is an indicator function that outputs 1 when the input x is positive and zero otherwise. To facilitate the computation, we upper bound the training error by the following expression:

$$err_T \leq \frac{1}{m}\sum_{i=1}^{m} \frac{(1+\lambda)\sum_{y \neq y_i} e^{\vec{f}_T(y)\cdot\vec{g}_T(x_i)}}{e^{f_T(y_i)\cdot\vec{g}_T(x_i)} + \lambda \sum_{y \neq y_i} e^{\vec{f}_T(y)\cdot\vec{g}_T(x_i)}} \tag{2}$$

where λ is a smoothing parameter that prevents the overfitting of training examples. Notice that the contribution of each training example to the upper bound in (2) is bounded by factor $1 + 1/\lambda$. As will be shown later, AdaBoost.OC corresponds to the case when $\lambda \rightarrow 0$, which leads to the unbounded contribution from each training example.

As an iterative procedure, the goal of our boosting algorithm is to learn classifier $h_{T+1}(x)$, combination constant α_{T+1}, and coding function $f_{T+1}(y)$ at the $T+1$ iteration given $H_T(x)$ learned from the T-th iteration. To this end, we rewrite the training error at $T+1$ iteration into the following expression:

$$\begin{aligned} err_{T+1} &\leq \frac{1}{m}\sum_{i=1}^{m} \frac{1+\lambda}{\lambda + \frac{\exp(\vec{f}_{T+1}(y_i)\cdot\vec{g}_{T+1}(x_i))}{\sum_{y \neq y_i}\exp(\vec{f}_{T+1}(y)\cdot\vec{g}_{T+1}(x_i))}} \\ &= \frac{1}{m}\sum_{i=1}^{m} \frac{1+\lambda}{\lambda + \frac{\exp(\vec{f}_T(y_i)\cdot\vec{g}_T(x_i)+f(y_i)g(x_i))}{\sum_{y \neq y_i}\exp(\vec{f}_T(y)\cdot\vec{g}_T(x_i)+f(y)g(x_i))}} \end{aligned} \tag{3}$$

In the above, we drop the index $T+1$ and simply write $f_{T+1}(y)$ and $g_{T+1}(x)$ as $f(y)$ and $g(x)$, respectively. Define

$$\mu(y|x_i) = \frac{\exp(\vec{f}_T(y)\cdot\vec{g}_T(x_i))}{\exp(\vec{f}_T(y_i)\cdot\vec{g}_T(x_i)) + \lambda\sum_{y' \neq y_i}\exp(\vec{f}_T(y')\cdot\vec{g}_T(x_i))} \tag{4}$$

Then, the training error in (3) can be rewritten as:

$$err_{T+1} \leq \frac{1}{m}\sum_{i=1}^{m} \frac{(1+\lambda)\sum_{y \neq y_i}\mu(y|x_i)e^{f(y)g(x_i)}}{\mu(y_i|x_i)e^{f(y_i)g(x_i)} + \lambda\sum_{y \neq y_i}\mu(y|x_i)e^{f(y)g(x_i)}}$$

Using inequality $\frac{1}{px+qy} \leq \frac{p}{x} + \frac{q}{y}$ if $p+q=1$ and $x, y, p, q > 0$, the above equation is relaxed as:

$$\begin{aligned} err_{T+1} &\leq \frac{\lambda+1}{m}\sum_{i=1}^{m}\lambda\left(\sum_{y \neq y_i}\mu(y|x_i)\right)^2 \\ &+ \frac{\lambda+1}{m}\sum_{i=1}^{m}\mu(y_i|x_i)\sum_{y \neq y_i}\mu(y|x_i)e^{\alpha h(x_i)(f(y)-f(y_i))} \end{aligned} \tag{5}$$

For the convenience of presentation, we drop the first term in the above expression since it is independent from $f(y)$, α, and $h(x)$. Using the convexity of exponential function, the above expression is then simpli-

fied as:

$$err_{T+1} \leq \frac{1+\lambda}{2m} \times \left(\begin{array}{l} \sum_{i=1}^{m} \mu(y_i|x_i) e^{-2\alpha h(x_i) f(y_i)} \sum_{y} \mu(y|x_i)(1 - f(y) f(y_i)) \\ + \sum_{i=1}^{m} \mu(y_i|x_i) \sum_{y \neq y_i} \mu(y|x_i)(1 + f(y) f(y_i)) \end{array} \right) \quad (6)$$

Define weight distributions $D_{T+1}(i, y)$ and $D_{T+1}(i)$ as:

$$D_{T+1}(i, y) = \frac{\mu(y_i|x_i)\mu(y|x_i)}{Z_{T+1}} \quad (7)$$

$$D_{T+1}(i) = \frac{\sum_y D_{T+1}(i, y)[f(y_i) \neq f(y)]}{U_{T+1}} \quad (8)$$

where Z_{T+1} and U_{T+1} are the normalization factors. Then, Eqn. (6) can be simplified as:

$$err_{T+1} \leq Z_{T+1} U_{T+1} \sum_{i=1}^{m} e^{-2\alpha h(x_i) f(y_i)} D_{T+1}(i) + Z_{T+1} \sum_{i=1}^{m} \sum_{y \neq y_i} D_{T+1}(i, y)(1 + f(y) f(y_i)) \quad (9)$$

To minimize the upper bound in the above expression, we will train a classifier $h(x)$ on the training examples that are weighted by $D_{T+1}(i)$. Furthermore, given a coding function $f(y)$ and a binary classification function $h(x)$, the combination weight α that minimizes the upper bound in Eqn. (9) is:

$$\alpha = \frac{1}{4} \ln \left(\frac{1 - \epsilon_{T+1}}{\epsilon_{T+1}} \right) \quad (10)$$

where

$$\epsilon_{T+1} = \sum_{i=1}^{m} D_{T+1}(i)[f(y_i) \neq h(x_i)] \quad (11)$$

Note that when $\lambda = 0$, our weight distribution $D_{T+1}(i, y)$ in (7) becomes the same expression as the one in Figure (1). This is because when $\lambda = 0$, we have

$$\mu(y|x_i) = \exp((\vec{f}_T(y) - \vec{f}_T(y_i)) \cdot \vec{g}_T(x_i))$$

. Then,

$$D_{T+1}(i, y) = \frac{1}{Z_{T+1}} \mu(y_i|x_i)\mu(y|x_i) = \frac{D_T(i, y)}{Z_{T+1}} e^{2\alpha_T([f_T(y) = h_T(x_i)] + [f_T(y_i) \neq h_T(x_i)])}$$

Thus, the proposed boosting algorithm can be viewed as the generalized version of AdaBoost.OC algorithm. We refer to the proposed smoothed boosting algorithm as "**SmoothBoost**". The detailed steps of the SmoothBoost algorithm is summarized in Figure 2.

Given: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
Initialize $\mu_1(y|x_i) = 1/(1 + \lambda(|\mathcal{Y}| - 1))$

For $t = 1, ..., T$

1. Compute $D_t(i, y) = \mu_t(y_i|x_i)\mu_t(y|x_i)/Z_t$ where Z_t is a normalization factor.
2. Generate coding function $f_t(y) : \mathcal{Y} \rightarrow \{0, 1\}$
3. Let $U_t = \sum_{i=1}^{m} \sum_{y \in \mathcal{Y}} D_t(i, y)[f_t(y) \neq f_t(y_i)]$
4. Let $D_t(i) = \sum_{y \in \mathcal{Y}} D_t(i, y)[f_t(y) \neq f_t(y_i)]/U_t$
5. Train a weak classifier $h_t(x) : \mathcal{X} \rightarrow \{0, 1\}$ on examples weighted by $D_t(i)$
6. Let $\epsilon_t = \sum_{i=1}^{m} D_t(i)[f_t(y_i) \neq h_t(x_i)]$
7. Let $\alpha_t = \frac{1}{4} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
8. Update $\mu_{t+1}(y|x_i)$ as

$$\mu_{t+1}(y|x_i) = \frac{\mu_t(y|x_i) \exp(\alpha f_t(y) h_t(x_i))}{\mu_t(y_i|x_i) e^{\alpha f_t(y_i) h_t(x_i)} + \lambda \sum_{y' \neq y_i} \mu_t(y'|x_i) e \alpha f_t(y') h_t(x_i)}$$

Output: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{T} \alpha_t [h_t(x) = f_t(y)]$

Figure 2. Description of the SmoothBoost algorithm

3.2. The Probabilistic Output Code

The difficulty in determining coding functions arises from the fact that $h(x)$ and $f(y)$ are coupled with each other through the term $\exp(\alpha(x_i)h(x_i)(f(y) - f(y_i))$ in Eqn. (5). Thus, to identify the optimal coding function $f(y)$, we need to make certain assumption about classifier $h(x)$. In this paper, we assume $h(x)$ to be a random classifier, which outputs 1 with probability ρ and -1 with probability $1 - \rho$. Given the symmetry between $+1$ and -1, we assume $0 \leq \rho \leq 1/2$. Then, the upper bound in (5) is simplified as:

$$err_{T+1} \leq \frac{1+\lambda}{m} \times \sum_{i=1}^{m} \mu(y_i|x_i) \left(\begin{array}{l} \rho \sum_{y \neq y_i} \mu(y|x_i) e^{\alpha[f(y) - f(y_i)]} + \\ (1 - \rho) \sum_{y \neq y_i} \mu(y|x_i) e^{\alpha[f(y_i) - f(y)]} \end{array} \right) \quad (12)$$

The above expression can be relaxed as:

$$err_{T+1} \leq \frac{1+\lambda}{m} \sum_{i=1}^{m} \sum_{y \neq y_i} \mu(y_i|x_i)\mu(y|x_i)(e^{2\alpha} + e^{-2\alpha} + 1)$$

$$+ \alpha(1-2\rho)\frac{1+\lambda}{m}\sum_{i=1}^{m}\sum_{y\neq y_i}\mu(y_i|x_i)\mu(y|x_i)(f(y)-f(y_i))$$

using inequality $e^{\alpha x} \leq e^{\alpha} + e^{-\alpha} + 1 - \alpha x$ for $-1 \leq x \leq 1$. Thus, a coding function $f(y)$ that effectively reduces the upper bound in the above equation should minimize the following expression:

$$\begin{aligned}\Omega(f) &= \frac{1-2\rho}{m}\sum_{i=1}^{m}\sum_{y\neq y_i}\mu(y_i|x_i)\mu(y|x_i)(f(y)-f(y_i)) \\ &= (1-2\rho)\sum_{y} f(y)\phi(y)\end{aligned}$$

where

$$\phi(y) = \frac{1}{m}\sum_{i=1}^{m}\mu(y_i|x_i)\left(\mu(y|x_i) - \delta(y,y_i)\sum_{y}\mu(y|x_i)\right)$$

and $\delta(y,y')$ outputs 1 when $y = y'$ and zero otherwise. Hence, the optimal solution is:

$$f(y) = -I(\phi(y)) \tag{13}$$

We refer to the above expression as the "**Deterministic Output Code**".

One problem with the above approach is that the bound in Eqn. (12) is valid only when the output of classifier $h(x)$ follows the Bernoulli distribution with probability ρ . When this is not the case, the deterministic code will no long be optimal. To relieve the dependence of the optimal coding function on the assumption of the basis classifier, we consider a probabilistic coding scheme. In particular, we view $\Omega(f)$ as a log-likelihood function for $f(y)$, i.e.,

$$\log \Pr(f) \propto (1-2\rho)\sum_{y} f(y)\phi(y)$$

Then, $\Pr(f(y)=1)$ is written as:

$$\Pr(f(y)=1) = \frac{1}{1+\exp\left(-\gamma\phi(y)\right)} \tag{14}$$

where $\gamma \propto 1-2\rho$. To generate a coding function $f(y)$, we will sample the value for $f(y)$ according to the distribution in (14). We refer to this method as the "**Probabilistic Output Code**". Parameter γ is determined by the cross validation method with 20% of training data used for validation.

4. Experiment

In this experiment, we examine the effectiveness of the SmoothBoost algorithm and the effectiveness of the probabilistic output code, separately. To evaluate the performance of SmoothBoost for multi-class learning, we compare it to three baseline algorithms that also combine binary classification models for multi-class learning. They are: 1) the ECOC method that uses a random code, 2) the AbaBoost.OC algorithm in (Schapire, 1997), and 3) the one-against-all approach that uses the regularized boosting algorithm (Jin et al., 2003) as its binary classifier. We refer to these three baseline models as "*ECOC*", "*AdaBoost.OC*", and "*BinBoost.Reg*". To evaluate the proposed coding scheme, we compare it to a random code. We also compare the probabilistic output code to the deterministic one to see if the probabilistic output code is more robust and effective for multi-class learning.

4.1. Experimental Design

Seven multi-class datasets from the UCI machine learning repository are used. The statistics of these datasets are listed in Table 1. In order to examine the robustness of the proposed algorithms, we conducted experiments with noisy data, which are generated by randomly selecting 20% of training data and assigning them with incorrect labels.

A decision stump is used as the basis classifier throughout the experiment. All boosting algorithms generate 50 binary classifiers and combine them to make prediction. The choice of 50 iterations is based on the previous studies on AdaBoost (e.g., (Dietterich, 2000)). For each dataset, 60% of data are randomly selected for training and the rest is used for testing. Each experiment is repeated 10 times and the average classification error together its variance is used as the evaluation metric.

4.2. Experiment (I): Effectiveness of SmoothBoost

We tested the SmoothBoost algorithm on the UCI datasets. A random code is used for the proposed boosting algorithm. To see the effectiveness of SmoothBoost for multi-class learning, we also evaluate the performance of the three baseline methods using the same datasets. Table 2 summarizes the classification errors of the four methods.

First, compared to the ECOC method, we see that all three boosting algorithms are effective in reducing classification errors. For example, for dataset "ecoli", its classification error is reduced from 14.4% to around 3% by all three boosting algorithms. Second, we see that for most datasets, SmoothBoost is more effective than the other two boosting methods in improving the classification accuracy for multi-class learning. For in-

Table 1. Properties of test datasets

Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast	physiological
#Instance	327	178	2000	154	204	946	1484	2579
#Classes	5	3	10	3	5	4	9	5
#Features	7	13	16	14	10	18	9	13

Table 2. Classification errors (%) for the original UCI datasets.

Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast
ECOC	14.4± 6.1	20.3± 3.5	10.5± 5.7	6.7± 2.2	50.5 ± 3.4	30.3± 4.2	46.7 ± 5.5
AdaBoost.OC	3.5± 2.4	16.0± 3.3	2.7± 1.6	5.2± 2.1	49.7 ± 2.9	26.0± 3.8	36.4 ± 2.6
BinBoost.Reg	3.4± 1.8	13.2± 3.4	3.2± 1.3	4.8± 1.8	47.0 ± 1.9	26.9 ± 1.4	33.6 ± 2.9
SmoothBoost	2.8± 1.6	13.9 ± 2.1	2.3± 1.6	5.2±1.6	43.8 ± 2.8	21.4± 2.4	33.7 ± 3.1

stance, for dataset "vehicle", SmoothBoost reduces the training error to 21.4% while the classification errors of AdaBoost.OC and BinBoost.Reg are over 26%.

In order to see the robustness of SmoothBoost with regard to training noises, we tested both the SmoothBoost algorithm and the three baseline methods on the "corrupted" datasets in which 20% of training data are incorrectly labeled. The classification errors of the four methods are displayed in Table 3.

First, comparing Table 3 to Table 2, we observed substantial degradation in the classification errors for the ECOC method. The most noticeable cases are dataset "wine" and "pendigit", whose classification errors have increased from 20.3% and 10.5% to 27.2% and 15.6%, respectively. Second, compared to the ECOC method, AdaBoost.OC has experienced more severe degradation in its classification errors when training data are noisy. For example, for dataset "ecoli" and "pendigit", the classification errors of AdaBoost.OC have increased dramatically from 3.5% and 2.7% to 12.8% and 13.9%, respectively. Third, compared to AdaBoost.OC, both BinBoost.Reg and SmoothBoost are more robust to training noise. For all datasets, they suffer from less degradation than AdaBoost.OC in classification errors. This is because both BinBoost.Reg and SmoothBoost employ certain mechanism to alleviate the overfitting problem while AdaBoost.OC does not. Finally, comparing the SmoothBoost algorithm to BinBoost.Reg, we found that SmoothBoost is more resilient to training noise. For example, for dataset "pendigit", given 20% training noise, the classification error for SmoothBoost is only 6.4% while the classification error for BinBoost.Reg is over 12%. Although BinBoost.Reg employs the regularized boosting algorithm as its binary classifier, it is less effective than SmoothBoost because the regularized boosting algorithm is designed for binary classification, not for multi-class learning. Based on the above observation, we conclude that SmoothBoost is effective for multi-class learning and is robust to training noise.

4.3. Experiment (II): Effectiveness of Proposed Coding Schemes

In this experiment, we first examine the effectiveness of the probabilistic output code, followed by the comparison between the probabilistic output code and the deterministic code. All experiments discussed in this subsection use the SmoothBoost algorithm.

4.3.1. Effectiveness of the Probabilistic Output Code

In this subsection, we examine the efficiency and the robustness of the probabilistic output code by comparing it to a random code.

To test the efficiency of the probabilistic output code, we run each experiment twice with different number of iterations: a long run with 50 iterations, and a intermediate run with only 20 iterations. The classification errors of the probabilistic code and the random code for both runs are summarized in Table 4. We see that both coding schemes achieve similar performance when the number of iterations is 50. The difference becomes noticeable when the number of iterations is limited to 20. For dataset "wine", "glass", and "vehicle", the

Table 3. Classification errors (%) for the "corrupted" UCI datasets with 20% training noise.

Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast
ECOC	18.7± 4.3	27.2± 6.1	15.6± 4.4	8.8± 4.9	54.7 ± 4.7	32.7± 4.5	46.9 ± 5.8
AdaBoost.OC	12.8 ± 4.9	20.3 ± 4.5	13.9 ± 3.6	9.7 ± 4.5	55.0 ± 6.2	35.0 ± 4.4	42.5 ± 6.2
BinBoost.Reg	9.3± 3.4	21.2 ± 3.0	12.0 ± 3.8	7.1 ± 3.1	49.9 ± 2.8	31.1 ± 2.7	37.5 ± 3.8
SmoothBoost	9.7 ± 3.1	17.1 ± 3.4	6.4 ± 2.5	8.0 ± 3.4	44.5 ± 1.4	23.3 ± 1.9	37.4 ± 4.3

Table 4. Comparison of a random code with the probabilistic output code for original UCI datasets.

#Iteration	Coding Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast
50	Random	2.8± 1.6	13.9 ± 2.1	2.3± 1.6	5.2±1.6	43.8 ± 2.8	21.4± 2.4	33.7 ± 3.1
	Prob.	2.9± 2.2	13.6 ± 2.5	2.1± 1.1	4.2±2.1	44.4 ± 3.0	21.4± 3.4	34.4 ± 3.5
20	Random	5.8± 2.7	18.5 ± 2.0	2.4 ± 1.1	5.5±2.7	47.4 ± 2.2	37.1± 3.1	37.6 ± 3.3
	Prob.	4.9± 2.8	14.9 ± 1.9	2.2± 1.4	4.3±2.7	45.8 ± 2.2	33.0± 2.9	37.2 ± 2.3

classification errors of the probabilistic output code is noticeably lower than the random code. This fact indicates that the probabilistic output code is more efficient than a random code in reducing the classification errors.

To see the robustness of the probabilistic code with regard to training noise, we test it on the corrupted UCI datasets where 20% of the training data are labeled incorrectly. The classification errors of both coding schemes are listed in Table 5. Overall, the probabilistic output code performs slightly better than the random code. The improvement is more noticeable for dataset "ecoli" and "iris". The probabilistic output code outperforms a random code with 6.1% vs. 9.7% for dataset "ecoli" and 5.5% vs. 8.0% for dataset "iris". This result indicates that the probabilistic output code is more resilient to training noise than the random code.

4.3.2. The Probabilistic Output Code Vs. The Deterministic Output Code

The classification results of both the probabilistic output code and the deterministic output code for the original UCI datasets are summarized in Table 6. We clearly see that the deterministic output code performs substantially worse than the probabilistic output code. The most noticeable cases are dataset "iris" and "glass". The classification errors of the deterministic output code for these two datsets are 11.5% and 38.4%, while the classification errors of the probabilistic output code are only 4.2% and 11.5%. This result indicates that the probabilistic output code is more effective for multi-class learning than the deterministic output code. As discussed earlier, the deterministic output code is heavily dependent on the assumption of the basis classifier. In contrast, the probabilistic output code is able to alleviate its dependence on the assumption of the basis classifier by probabilistically generating a coding function.

5. Conclusion

In this paper, we propose a new boosting algorithm that is able to boost binary classifiers for multi-class learning problems. It improves the AdaBoost.OC algorithm in two aspects:

- It introduces a smoothing mechanism, named "SmoothBoost", into the boosting algorithm to alleviate the overfitting problem.
- It introduces a probabilistic coding scheme to efficiently reduce the training errors of multi-class learning.

A set of extensive experiments have been conducted to evaluate the effectiveness and the robustness of the proposed boosting algorithm and coding schemes. Our empirical studies have shown that the proposed SmoothBoost algorithm performs better than the AdaBoost.OC algorithm for both clean data and noisy data. Our empirical studies also showed that the proposed probabilistic output code is more effective and

Table 5. Comparison of a random code with the probabilistic output code for the "corrupted" UCI datasets with 20% of the training data incorrectly labeled.

Coding Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast
Random	9.7 ± 3.1	17.1 ± 3.4	6.4 ± 2.5	8.0 ± 3.4	44.5 ± 1.4	23.3 ± 1.9	37.4 ± 4.3
Prob.	6.1 ± 3.5	16.3 ± 3.2	5.2± 2.3	5.5 ±2.9	44.5 ± 1.4	23.3± 1.9	36.4 ± 4.1

Table 6. Comparison of the probabilistic output code with the deterministic output code for the original UCI datasets.

Coding Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast
Deterministic	4.9± 2.6	17.2 ± 3.5	4.9 ± 1.0	11.5 ± 3.9	49.2 ± 4.8	38.4± 10.3	38.0 ± 5.8
Probabilistic	2.9± 2.2	13.6 ± 2.5	2.1± 1.1	4.2±2.1	44.4 ± 3.0	21.4± 3.4	34.4 ± 3.5

more robust for multi-class learning than a random code. In the future, we plan to extend this work to multi-label classification where each data point can be assigned with multiple labels.

References

Bauer, E., & Kohavi, R. (1999). An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants. *Machine Learning*, *36*, 105–139.

Dietterich, T. G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, *40*, 139–157.

Dietterich, T. G., & Bakiri, G. (1995). Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Proceedings of European Conference on Computational Learning Theory* (pp. 23–37).

Freund, Y., & Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 148–156).

Grove, A., & Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles. *Proceedings of the Fifteenth National Conference on Artifical Intelligence* (pp. 692–699).

Jiang, W. (2001). Some theoretical aspects of boosting in the presence of noisy data. *Proceedings of The Eighteenth International Conference on Machine Learning (ICML-2001).*

Jin, R., Liu, Y., Si, L., Carbonell, J., & Hauptmann, A. G. (2003). A New Boosting Algorithm Using Input-Dependent Regularizer. *Proceedings of Twentieth International Conference on Machine Learning (ICML'03).*

Lebanon, G., & Lafferty, J. (2001). Boosting and maximum likelihood for exponential models. *Neural Information Processing Systems (NIPS).*

Ratsch, G., Onoda, T., & Muller, K. (1998). An Improvement of AdaBoost to Avoid Overfitting. *Advances in Neutral Information Processing Systems* (pp. 506–509).

Ratsch, G., Onoda, T., & Muller, K. (1999). Regularizing AdaBoost. *Advances in Neutral Information Processing Systems 11.*

Ratsch, G., Onoda, T., & Muller, K. (2000). Soft Margins for AdaBoost. *Machine Learning*, *42*, 287–320.

Schapire, R. (1999). Theoretical views of boosting and applications. *Proceedings of Tenth International Conference on Algorithmic Learning Theory* (pp. 13–25).

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems. *Proceedings of 14th International Conference on Machine Learning* (pp. 313–321).