Mach Learn (2007) 67: 207–227
DOI 10.1007/s10994-007-5005-y

Multi-Class Learning by Smoothed Boosting

Rong Jin · Jian Zhang

Received: 30 June 2005 / Revised: 17 January 2007 /
Accepted: 24 January 2007 / Published online: 4 April 2007

Abstract AdaBoost.OC has been 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. One of the main drawbacks with the AdaBoost.OC algorithm is that it is sensitive to the noisy examples and tends to overfit training examples when they are noisy. In this paper, we propose a new boosting algorithm, named "MSmoothBoost", which introduces a smoothing mechanism into the boosting procedure to explicitly address the overfitting problem with AdaBoost.OC. We proved the bounds for both the empirical training error and the marginal training error of the proposed boosting algorithm. Empirical studies with seven UCI datasets and one real-world application have indicated that the proposed boosting algorithm is more robust and effective than the AdaBoost.OC algorithm for multi-class learning.

Keywords Boosting · Smoothing · Regularization · Multi-class learning

1 Introduction

AdaBoost has been shown to be an effective method for improving the classification accuracy of weak classifiers (Freund and Schapire 1996, 1997; Schapire 1997, 1999; Grove and Schuurmans 1998; Schapire and Singer 1999; Bauer and Kohavi 1999; Ratsch et al. 1999, 2000; Lebanon and Lafferty 2001; Jin et al. 2003). It works by repeatedly running a weak

Editor: Nicolo Cesa-Bianchi

R. Jin (✉)
Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA
e-mail: rongjin@cse.msu.edu

J. Zhang
Department of Statistics, Purdue University, West Lafayette, IN 47907, USA
e-mail: jianzhan@stat.purdue.edu

learner on various training examples sampled from the original training pool, and combining multiple instances of the weak learner into a single composite classifier. A straightforward way to extend the boosting algorithm to multi-class learning is to use AdaBoost as the base binary classifier when decomposing a multi-class learning problem into a set of binary ones. For example, in the AdaBoost.MO algorithm (Schapire and Singer 1999), the Error-Correcting Output Codes (ECOC) (Dietterich and Bakiri 1995) is first applied to reduce a multi-class learning problem into a set of binary classification problems, which are then solved by the AdaBoost algorithm. One problem with this approach is that the boosting procedure is completely independent from the partitioning of multiple classes, which could lead to undesirable results. To see this, consider a coding scheme that divides ten classes into two groups: the positive group that includes nine classes, and the negative group that includes only one single class. Evidently, it is much more likely for a binary classifier to misclassify a data point from a positive group into a negative group since the positive group is significantly more popular than the negative group. Therefore, if we follow the weights of binary boosting algorithm, the examples in the negative group will be weighted significantly higher than the examples in the positive group. But, this situation is simply an artifact of the coding scheme, and it does not imply that examples in the class of the negative group are more difficult to be classified correctly than the examples in other classes. Hence, a better designed boosting algorithm should take into account the partitioning of multiple classes when it computes the weight of training examples. This leads to the AdaBoost.OC (Schapire 1997), which extends the AdaBoost algorithm. Similar to the AdaBoost.MO algorithm, AdaBoost.OC extends AdaBoost to multi-class learning by reducing a multi-class learning problem into a set of binary ones using the ECOC method. But, unlike AdaBoost.MO where the boosting procedure is independent from the codewords that are generated by the ECOC method, AdaBoost.OC takes into account both the classification accuracy of the base binary classifier and the codewords that are generated by the ECOC method, which makes it more effective for multi-class learning as indicated in (Schapire 1997).

Despite its success, the AdaBoost.OC algorithm can potentially suffer from the overfitting problem, particularly when the training examples are noisy. Previous studies of overfitting with boosting algorithms mainly focus on the binary classifiers (Ratsch et al. 1998, 1999, 2000; Schapire 1999; Lebanon and Lafferty 2001; Jin et al. 2003). Empirical results have shown that the AdaBoost algorithm tends to overfit the noisy training examples. Since AdaBoost.OC is an extension of the AdaBoost algorithm for multi-class learning, it is similar to AdaBoost in that it reduces the exponential loss function by a greedy search. As a result, we expect that it will suffer from the overfitting problem as AdaBoost does. Although there have been many studies on how to prevent AdaBoost from overfitting training examples, none of them target on multi-class learning problems. As already argued before, applying binary boosting algorithms directly to multi-class learning may not be desirable given that they do not take into account the distribution of multi-class membership in the binary classes.

In this paper, we present a new boosting algorithm, named "MSmoothBoost", that explicitly addresses the overfitting problem of AdaBoost.OC for multi-class learning. In particular, 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 to prevent any single training example from dominating over other examples in any iteration. In contrast, in AdaBoost.OC, unbounded weights are assigned to individual examples. As a result, noisy training examples can be overly emphasized during the iterations when they are assigned with extremely large weights. It is important to note that the proposed "MSmoothBoost" is different from the smoothing approach for AdaBoost in (Schapire and

Singer 1999) where smoothing is introduced to prevent the linear combination weight α from being too large.

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

2 Related Work

Multi-class classification is one of the basic learning problems (Hsu and Lin 2002; Lee et al. 2002; Zhang 2003). Most approaches for multi-class learning can be divided into two categories. In the first category, a multi-class learning problem is decomposed into multiple binary classification problems. Each binary classification problem is learned separately and combined to make multi-class prediction. A common approach within this category is one-against-all, which builds a different binary classifier to distinguish every class from the rest of classes. Another commonly used approach is one-against-one (Hastie and Tibshirani 1998), which builds a binary classifier for every pair of classes. In (Platt et al. 2000), the authors proposed a different approach for combining binary SVM classifiers to make multi-class prediction based on the Direct Acyclic Graph (DAG). Furthermore, unlike the one-against-all or DAG approach where each binary classification model is solved independently, in (Vapnik 1998; Weston and Watkins 1999; Bredensteiner and Bennett 1999), several multi-class support vector machines have been proposed to solve all the binary classification problems simultaneously. Approaches in the second category target multi-class learning problems directly without converting them into a set of binary classification problems. Classification models like Naive Bayes (McCallum and Nigam 1998) and logistic regression (Zhu and Hastie 2001) can naturally be extended from binary classification problems to multi-class classification problems. In (Crammer and Singer 2000), the authors presented a maximum margin based framework for multi-class multi-label learning. In this paper, we are interested in combining AdaBoost with ECOC for multi-class learning problems, which belongs to the first category of approaches.

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. Note that function $[x]$ outputs 1 when the input logic variable x is true and 0 otherwise. AdaBoost.OC 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 $\{0, +1\}$. 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 predictions 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 estimated combination parameter α_t.

One problem with AdaBoost.OC is that it could overfit the training data. Note that in Fig. 1 $D_t(i, y)$ is updated by multiplying with an exponential factor, and thus may be very large (close to 1). If 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. In the later experimental section, we will demonstrate the overfitting problem with AdaBoost.OC for multi-class learning problems.

In the past, a number of studies have been devoted to prevent boosting algorithms from overfitting training examples, including the smoothing method (Schapire and Singer 1999),

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

For $t = 1, \ldots, T$

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

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

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

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

Fig. 1 Description of the AdaBoost.OC algorithm

Gentle Boost (Friedman et al. 2000), BrownBoost (Freund 2001), the linear programming approach (Ratsch et al. 1999), the quadratic programming approach (Ratsch et al. 1999), the Weight Decay method (Ratsch et al. 1998, 2000), and the smooth boosting algorithm (Servedio 2003). The main ideas of these methods can be summarized into two groups: one is to change the cost function used for boosting algorithms, and the other is to introduce a soft margin. The problem of overfitting for AdaBoost might be linked to the exponential cost function, which makes the weights of the noisy data grow exponentially and leads to the overemphasis of those data patterns. The solution to this issue can be, either to introduce a different cost function, such as a logistic regression function in (Friedman et al. 2000), or to regularize the exponential cost function with a penalty term such as the weight decay method used in (Ratsch et al. 1998), or to introduce a different weighting function, such as BrownBoost (Freund 2001), or limit the weight assigned to training examples whose classification margin does not reach the desired threshold (Servedio 2003). A more general solution is to replace the hard margin in AdaBoost with a soft margin. Similar to the strategy used in the support vector machine (SVM) algorithm (Burges 1998), the boosting algorithm with a soft margin is able to allow a larger margin at the expense of some misclassification errors. This idea leads to works such as the regularized boosting algorithms using both linear programming and quadratic programming (Ratsch et al. 1999). Despite the extensive study on the overfitting issue with the boosting algorithms, none of them target on multi-class learning, which is the focus of this paper. It is worth noting that although the boosting algorithm proposed in (Servedio 2003) is similar to this paper in that both of these two studies try to avoid assigning very large weights to the difficult examples, the strategy used in our paper is very different from the one in (Servedio 2003). Unlike (Servedio 2003) where the difficult examples are assigned with a constant weight, this study allows varied weights to be assigned to the examples that are difficult to classify, which allows our work to address different levels of difficulty in classification by different weights. Furthermore, this study targets on multi-class learning while the study in (Servedio 2003) is focused on binary classification problems.

Finally, the idea of applying the ECOC coding method to reduce a multi-class learning problem into a set of binary ones can be found in a number of studies (Dietterich and Bakiri 1995; Berger 1999; Crammer and Singer 2000; Allwein et al. 2000; Ghani 2000). In (Dietterich and Bakiri 1995), the authors presented a general study of using the ECOC method for multi-class learning, and showed a significant reduction in classification error. In (Berger 1999; Ghani 2000), the authors evaluated both a random code (Berger 1999) and the BCH code (Hill 1986) for multi-category text classification. In (Allwein et al. 2000), an error analysis of reducing multi-class learning to binary class learning is presented. In (Crammer and Singer 2000), the authors showed that finding optimal ECOC codes for specific multi-class classification problems is NP-complete, and introduced the "continuous output codes" to approximate the optimal solution by relaxing the discrete codes to continuous ones. A number of coding schemes were studied in (Schapire 1999) under the context of boosting. In (Masulli and Valentini 2003), the authors presented empirical studies about the ECOC coding for multi-class learning. The results have shown that the performance of ECOC not only depends on the base classifier and coding schemes, but also on a number of other factors, such as the relationship among base classifiers and dependence between codeword bits.

3 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 classifiers that are learned in the iterations.

Let the coding functions for the first T iterations be denoted by $\vec{f}_T(y) = (f_1(y), \ldots, f_T(y))$ where each coding function $f_t(y) : \mathcal{Y} \to \{-1, +1\}$. Let $\vec{g}_T(x) = (\alpha_1 h_1(x), \ldots, \alpha_T h_T(x))$ be the weighted classifiers obtained in the first T iterations where $h_t(x) : \mathcal{X} \to \{-1, +1\}$ 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:

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

where m is the number of training examples. $I(x)$ is an indicator function that outputs 1 when the input x is positive and 0 otherwise. For the convenience of presentation, we define

$$\Phi_T(y, x) = \vec{f}_T(y, x) \cdot \vec{g}_T(y, x). \tag{2}$$

$\Phi_T(y, x)$ defined above indicates the confidence of assigning an instance x to class y in the T-th iteration. The larger the $\Phi_T(y, x)$ is, the more likely that instance x will be assigned to

class y. Using the notation of $\Phi_T(y, x)$, the training error err_T can be rewritten as:

$$err_T = \frac{1}{m}\sum_{i=1}^{m} I\Big(\max_{y\neq y_i}[\Phi_T(y, x_i) - \Phi_T(y_i, x_i)]\Big).$$

To facilitate the computation, we upper bound the training error by the following expression:

$$err_T \leq \frac{1}{m}\sum_{i=1}^{m} \frac{(1+\lambda)\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))}{\exp(\Phi_T(y_i, x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))} \tag{3}$$

where $\lambda \geq 0$ is a smoothing parameter. The above inequality holds for every correctly classified example (x_i, y_i) since $I(\max_{y\neq y_i}[\Phi_T(y, x_i) - \Phi_T(y_i, x_i)]) = 0$. When example x_i is misclassified, the indicator function will take value 1. We also have

$$\exp(\Phi_T(y_i, x_i)) \leq \sum_{y\neq y_i}\exp(\Phi_T(y, x_i)).$$

Thus, the above inequality also holds for misclassified examples.

The introduction of λ in (3) is to prevent overfitting training examples that are difficult to classify. One reason for AdaBoost to overfit training examples is because the noisy training examples can have a dominant contribution to the overall loss function compared with the contribution of other training examples. As a result, most of the optimization effort is put on improving the classification results of the noisy training examples, which could lead to significant degradation in the overall performance. In (3), by introducing smoothing parameter λ, we guarantee that the contribution of each training example to the upper bound in (3) is bounded by $1 + 1/\lambda$. To see this, we have

$$\begin{aligned}&\frac{(1+\lambda)\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))}{\exp(\Phi_T(y_i, x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))}\\ &\quad\leq \frac{(1+\lambda)\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))}{\lambda\sum_{y\neq y_i}\exp(\Phi_T(y, x_i))} = \frac{1+\lambda}{\lambda}.\end{aligned}$$

Hence, the contribution of each training example to the upper bound on training errors would be limited when $\lambda > 0$. It is also interesting to note that when $\lambda \to 0$, the upper bound in (3) becomes:

$$err_T \leq \frac{1}{m}\sum_{i=1}^{m}\sum_{y\neq y_i}\exp(\Phi_T(y, x_i) - \Phi_T(y_i, x_i)),$$

and the contribution of each example to the upper bound becomes unbounded. As we will show later, the above error bound will lead to the AdaBoost.OC algorithm.

At iteration $T + 1$, the goal of our boosting algorithm is to learn classifier $h_{T+1}(x)$, combination constant α_{T+1}, and coding function $f_{T+1}(y)$ given $H_T(x)$ that is learned from the T-th iteration. To shorten our notation, in the rest of the derivation, we drop the index $T + 1$ and simply write $f_{T+1}(y)$, $g_{T+1}(x)$ and α_{T+1} as $f(y)$, $g(x)$ and α, respectively. Using the fact that $\Phi_{T+1}(x, y) = \Phi_T(x, y) + f(y)g(x)$, we rewrite the upper bound of training

error at $T+1$ iteration as follows:

$$err_{T+1} \leq \frac{1}{m}\sum_{i=1}^{m}\frac{(1+\lambda)\sum_{y\neq y_i}\mu_T(y|x_i)\exp(f(y)g(x_i))}{\mu_T(y_i|x_i)\exp(f(y_i)g(x_i))+\lambda\sum_{y\neq y_i}\mu_T(y|x_i)\exp(f(y)g(x_i))} \tag{4}$$

where $\mu_T(y|x_i)$ is defined as:

$$\mu_T(y|x_i)=\frac{\exp(\Phi_T(y,x_i))}{\exp(\Phi_T(y_i,x_i))+\lambda\sum_{y'\neq y_i}\exp(\Phi_T(y',x_i))}. \tag{5}$$

$\mu_T(y|x_i)$ in the above can be interpreted as the confidence of classifying the instance x_i into class y in the T-th iteration. The higher the $\mu_T(y|x_i)$ is, the more likely that instance x_i will be classified into class y. Furthermore, it is important to note that $0<\mu(y_i|x_i)\leq 1$ according to the definition in (5).

We will then simplify the upper bound in (4) by using the convexity of reciprocal function, namely

$$\frac{1}{\sum_{i=1}^{n}p_i x_i}\leq\sum_{i=1}^{n}\frac{p_i}{x_i} \tag{$*$}$$

where $p_i, i=1,2,\ldots,n$ is a probability distribution and $x_i>0, i=1,2,\ldots,n$. By defining

$$p(y|x_i)=\begin{cases}\lambda\mu_T(y|x_i), & y\neq y_i,\\ \mu_T(y|x_i), & y=y_i,\end{cases}$$

we rewrite (4) as

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

and now using $(*)$ we get

$$\begin{aligned}err_{T+1}&\leq\frac{1+\lambda}{m}\sum_{i=1}^{m}\Bigg(\sum_{y\neq y_i}\mu_T(y|x_i)\exp(f(y)g(x_i))\Bigg)\Bigg(\sum_y p(y|x_i)\exp(-f(y)g(x_i))\Bigg)\\&\leq\frac{\lambda+1}{m}\sum_{i=1}^{m}\lambda\Bigg(\sum_{y\neq y_i}\mu_T(y|x_i)\Bigg)^2\\&\quad+\frac{\lambda+1}{m}\sum_{i=1}^{m}\mu_T(y_i|x_i)\sum_{y\neq y_i}\mu_T(y|x_i)\exp\big(\alpha h(x_i)(f(y)-f(y_i))\big).\end{aligned} \tag{6}$$

In the last step, we use the definition of $g(x_i)=\alpha h(x_i)$. Using the convexity of exponential function and the fact $f(y)^2=1$, $\exp(\alpha h(x_i)(f(y)-f(y_i)))$ can be upper bounded as:

$$\begin{aligned}&\exp\big(\alpha h(x_i)(f(y)-f(y_i))\big)\\&\quad=\exp\big(\alpha h(x_i)f(y_i)(f(y)f(y_i)-1)\big)\\&\quad=\exp\Bigg(-2\alpha h(x_i)f(y_i)\frac{1-f(y)f(y_i)}{2}+0\times\frac{1+f(y)f(y_i)}{2}\Bigg)\end{aligned}$$

$$\leq \frac{1 - f(y)f(y_i)}{2} \exp(-2\alpha h(x_i) f(y_i)) + \frac{1 + f(y)f(y_i)}{2}.$$

Substituting the second term in (6) using the above inequality, we further relax the bound in (6) as follows:

$$\begin{aligned} err_{T+1} \leq & \frac{\lambda + 1}{m} \sum_{i=1}^{m} \lambda \left(\sum_{y \neq y_i} \mu_T(y|x_i) \right)^2 \\ & + \frac{1 + \lambda}{2m} \sum_{i=1}^{m} \mu_T(y_i|x_i) \exp(-2\alpha h(x_i) f(y_i)) \sum_{y} \mu_T(y|x_i)(1 - f(y)f(y_i)) \\ & + \frac{1 + \lambda}{2m} \sum_{i=1}^{m} \mu_T(y_i|x_i) \sum_{y \neq y_i} \mu_T(y|x_i)(1 + f(y)f(y_i)). \end{aligned} \tag{7}$$

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

$$D_{T+1}(i, y) = \frac{\mu_T(y_i|x_i)\mu_T(y|x_i)}{Z_{T+1}}, \tag{8}$$

$$D_{T+1}(i) = \frac{\sum_y D_{T+1}(i, y)[f(y_i) \neq f(y)]}{U_{T+1}} \tag{9}$$

where Z_{T+1} and U_{T+1} are the normalization factors that are defined as follows:

$$Z_{T+1} = \sum_{i=1}^{m} \sum_{y \neq y_i} \mu_T(y_i|x_i)\mu_T(y|x_i),$$

$$U_{T+1} = \sum_{i=1}^{m} \sum_{y} D_{T+1}(i, y)[f(y_i) \neq f(y)].$$

Using the notation $D_t(i, y)$ and $D_t(i)$, we rewrite (7) as follows:

$$\begin{aligned} err_{T+1} \leq & \frac{\lambda + 1}{m} \sum_{i=1}^{m} \lambda \left(\sum_{y \neq y_i} \mu_T(y|x_i) \right)^2 \\ & + \frac{1 + \lambda}{2m} Z_{T+1} \sum_{i=1}^{m} \sum_{y \neq y_i} D_{T+1}(i, y)(1 + f(y)f(y_i)) \\ & + \frac{1 + \lambda}{m} Z_{T+1} U_{T+1} \sum_{i=1}^{m} \exp(-2\alpha h(x_i) f(y_i)) D_{T+1}(i). \end{aligned} \tag{10}$$

Notice that the last term in above expression can be viewed as some kind of classification error. This is because $\exp(-2\alpha h(x_i) f(y_i))$ gives a small value (e.g., $e^{-2\alpha}$) when the prediction $h(x_i)$ is identical to the assigned class membership $f(y_i)$, and a large value (e.g., $e^{2\alpha}$) when $h(x_i) \neq f(y_i)$. Thus, we can interpret $D_{T+1}(i)$ and $D_{T+1}(i, y)$ as the weight for the i-th example in the $T + 1$ iteration. More specifically, $D_{T+1}(i, y)$ can be interpreted as the weight assigned to an instance x_i when it is misclassified to class y in the $T + 1$ iteration. Note that according to the definition in (8), weight $D_{T+1}(i, y)$ depends not only on the

confidence $\mu_T(y|x_i)$, but also on the confidence $\mu_T(y_i|x_i)$. Hence, if a instance has a very small chance to be classified correctly, its weight $D_{T+1}(i, y)$ could be small even if it is very likely to be misclassified into class y. It is this smoothing mechanism that will effectively prevent the boosting algorithm from overfitting training examples. $D_{T+1}(i)$, according to the definition in (9), is computed as the linear combination of $D_{T+1}(i, y)$, and therefore can be interpreted as the overall weight assigned to the instance x_i for its misclassification. Note that the overall weight $D_{T+1}(i)$ will not only depend on how likely each instance is misclassified but also depend on the partitioning of multiple classes by the coding function $f(y)$.

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 (10) is:

$$\alpha = \frac{1}{4} \ln\left(\frac{1-\epsilon_{T+1}}{\epsilon_{T+1}}\right) \tag{11}$$

where

$$\epsilon_{T+1} = \sum_{i=1}^{m} D_{T+1}(i)[f(y_i) \neq h(x_i)]. \tag{12}$$

Note that when $\lambda = 0$, our weight distribution $D_{T+1}(i, y)$ in (8) becomes the same expression as the one in Fig. 1. This is because when $\lambda = 0$, we have

$$\mu_T(y|x_i) = \frac{\exp(\Phi_T(y, x_i))}{\exp(\Phi_T(y_i, x_i))} = \exp\big((\vec{f}_T(y) - \vec{f}_T(y_i)) \cdot \vec{g}_T(x_i)\big).$$

Then,

$$\begin{aligned} D_{T+1}(i, y) &= \frac{1}{Z_{T+1}} \mu_T(y_i|x_i)\mu_T(y|x_i) \\ &\propto D_T(i, y) \exp\big(2\alpha_T([f_T(y) = h_T(x_i)] + [f_T(y_i) \neq h_T(x_i)])\big). \end{aligned}$$

The above relation can be derived directly using the definition of $\mu_T(y|x)$ and $D_T(i, y)$. Thus, the proposed boosting algorithm can be viewed as the generalized version of AdaBoost.OC algorithm. In Fig. 2, we summarize the steps for the proposed boosting algorithm. Note that in the last step of each iteration, we compute the values of $\mu_T(y|x_i)$ by an iterative updating equation:

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

The above updating equation can be directly derived from the original definition of $\mu_t(y|x_i)$ in (5) by using the fact

$$\Phi_{t+1}(y, x_i) = \Phi_t(y, x_i) + \alpha f_t(y)h_t(x_i).$$

Note also that the final output hypothesis $H(x)$ in Fig. 2 is the same as the one defined in (1) since

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

For $t = 1, \ldots, T$

1. Compute weight distribution $D_t(i, y) = \mu_t(y_i|x_i)\mu_t(y|x_i)/Z_t$ where Z_t is a normalization factor (chosen so that $D_{t+1}(i, y)$ is sum to 1)
2. Generate coding function $f_t(y) : \mathcal{Y} \to \{-1, 1\}$
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 $h_t(x) : \mathcal{X} \to \{-1, 1\}$ 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

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

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

Fig. 2 Description of the MSmoothBoost algorithm

$$\begin{aligned} H(x) &= \arg\max_{y \in \mathcal{Y}} \vec{f}_T(y) \cdot \vec{g}_T(x) \\ &= \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \alpha_t h_t(x) f_t(y) \\ &= \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \alpha_t (2[h_t(x) = f_t(y)] - 1) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \alpha_t [h_t(x) = f_t(y)]. \end{aligned}$$

We refer to the proposed algorithm as *MSmoothBoost*.

3.1 Bounds for Training Errors

In this subsection, we will bound the training error for the MSmoothBoost algorithm with T iterations.

Theorem 1 *Let $h_1, \ldots, h_T$ and $f_1, \ldots, f_T$ be the classifiers and the coding functions that are generated by running the MSmoothBoost algorithm (in Fig. 2). Let $\epsilon_1, \ldots, \epsilon_T$ be the weighted classification errors of $h_1, \ldots, h_T$. Then, the following inequality holds*:

$$\frac{1}{m} \sum_{i=1}^{m} I\left(\max_{y \neq y_i} \Phi_T(y, x_i) - \Phi_T(y_i, x_i)\right) \leq \prod_{t=1}^{T} (1 - \nu_t \gamma_t^2) \tag{13}$$

where

$$\nu_t = \frac{\sum_{i=1}^{m} \sum_{y \neq yi} \mu_t(y_i|x_i)\mu_t(y|x_i)[f(y) \neq f(y_i)]}{\sum_{i=1}^{m} \sum_{y \neq y_i} \mu_t(y|x_i)} \tag{14}$$

and $\gamma_t = 1 - 2\epsilon_t$.

Similar to the error bound theorem for AdaBoost (Schapire 1999), the above theorem indicates that the proposed MSmoothBoost algorithm is able to reduce the empirical training error exponentially if the underlying base classifier is better than random. The variable γ_t is introduced for the classification accuracy of each iteration. In particular, the smaller the classification error of each iteration is, the larger the γ_t is, and therefore the smaller the training error will be. The variable ν_t used in this theorem reveals the effect of smoothing parameter λ on training errors. To see the relationship between λ and ν, we examine the definition of ν in (14). Notice that one of the main difference between the denominator and the numerator in (14) is the term $\mu_t(y_i|x_i)$. Since both $\mu_t(y_i|x_i) \in (0, 1]$ and $\mu(y|x_i) > 0$, we have $0 < \nu_t \leq 1$. Furthermore, since $\mu_t(y_i|x_i)$ is inverse to λ, we would expect that in general the ν is inverse to λ, namely the smaller the λ is, the larger the ν will be. Combining this analysis with the fact that the larger the ν_t is the smaller the training error is, we conclude that a small smoothing parameter λ will usually lead to a small training error. The detailed proof of the above theorem can be found in Appendix 1.

Since the introduction of smoothing parameter λ does not improve the empirical training error, a natural question arises, i.e., in what aspect will λ help improve the quality of the boosting algorithm. The following theorem answers this question by showing that the introduction of smoothing parameter λ is effective in reducing the marginal training error. Before we state this theorem, we first introduce the concept of normalized $\Phi_T(y, x)$, which is defined as

$$\hat{\Phi}_T(y, x) = \frac{\Phi_T(y, x)}{\sum_{t=1}^{T} \alpha_t}.$$

Theorem governing the marginal training error for $\hat{\Phi}(y, x)$ is then stated as follows:

Theorem 2 *Let $h_1, \ldots, h_T$ and $f_1, \ldots, f_T$ be the classifiers and the coding functions that are generated by running the MSmoothBoost algorithm (in Fig. 2). Let $\epsilon_1, \ldots, \epsilon_T$ be the weighted classification errors of $h_1, \ldots, h_T$. Then, the following inequality holds for any $\theta \in [0, 1]$:*

$$\frac{1}{m}\sum_{i=1}^{m} I\Big(\max_{y \neq y_i} \hat{\Phi}_T(y, x_i) - \hat{\Phi}_T(y_i, x_i) + \theta\Big) \leq \frac{1}{(1+\lambda)^2}\left(\lambda(2+\lambda) + \prod_{t=1}^{T}\left(\frac{1+\gamma_t}{1-\gamma_t}\right)^{4\theta}\right)\prod_{t=1}^{T}(1-\nu_t\gamma_t^2) + \frac{\lambda}{1+\lambda}. \tag{15}$$

The above theorem gives an explicit expression for how the smoothing parameter λ will impact the marginal training error. To see this clearly, we divide the upper bound in (15) into two parts:

- the empirical part of the error bound, i.e., $\lambda(1+2\lambda)/(1+\lambda)^2 \prod_{t=1}^{T}(1-\nu_t\gamma_t^2) + \lambda/(1+\lambda)$, and
- the marginal part of the error bound, i.e., $\frac{1}{(1+\lambda)^2}\prod_{t=1}^{T}(1-\nu_t\gamma_t^2)[(1+\gamma_t)/(1+\gamma_t)]^{4\theta}$.

It is not difficult to see that the empirical part of the error bound is in general proportional to the value of λ, namely the larger the λ is, the larger the empirical part will be. In contrast, the marginal part of the error bound is in general inverse to λ, i.e., the larger the λ is the smaller the marginal part will be. Hence, by choosing appropriate value for the smoothing parameter λ, we will be able to balance the tradeoff between the empirical training error and

Table 1 Properties of test datasets

Method	ecoli	wine	pendigit	iris	glass	vehicle	yeast	phys.
#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

the errors related to the margin, which will lead to more desirable generalization error. The detailed proof of this theorem can be found in Appendix 2.

4 Experiments

In this experiment, we examine the effectiveness of the proposed MSmoothBoost algorithm for multi-class learning. For the simplicity of implementation, a random code is used to generate the coding function $f(y)$. Three baseline algorithms that also combine binary classification models for multi-class learning are used in the evaluation. They are:

- the ECOC method that uses a random code,
- the AdaBoost.OC algorithm in (Schapire 1997), and
- the one-against-all approach that uses the regularized boosting algorithm (Jin et al. 2003) as its base binary classifier.

We refer to these three baseline models as *ECOC*, *AdaBoost.OC*, and *BinBoost.Reg*. We compare the MSmoothBoost algorithm to these three baseline algorithms over a number of UCI datasets and a real world application. We also evaluate the robustness of the proposed boosting algorithm over a number of corrupted datasets. Finally, an extensive set of experiments are conducted to examine the impact of parameter λ and the base classifiers on the performance of the MSmoothBoost algorithm.

4.1 Experimental Design

Eight datasets are used in our experiments, as shown in Table 1. Among them, the first seven are multi-class datasets from the UCI machine learning repository and the last one is the physiological dataset from the physiological data modeling contest of ICML 2004.[1] In order to examine the robustness of the proposed algorithms, we conducted experiments with noisy data using the seven multi-class datasets. These noisy datasets are generated by randomly selecting 20% of training data and assigning them with incorrect labels. We also use the physiological dataset in our experiments, whose basic unit is the physiological section. Each section contains multiple physiological records and is assigned to one of the five classes of activities, with code "3003", "3004", "5101", "5102", and "5199". The challenge in classifying physiological sections lies in the fact that even though each physiological section is labeled by a single class, multiple activities may be conducted within a single section. For example, a user may fall into sleep when he/she is watching TV. Thus, although the section is labeled as "watch TV", some of its records are related to the activity of sleeping. Due to the large volume of the original dataset, a small subset of the physiological dataset

[1] http://www.cs.utexas.edu/users/sherstov/pdmc/

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

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

A *bold font* is used to highlight the result when it is significantly better than its counterparts based on the t-test at the confidence level 0.95

is randomly selected and used in our experiment. Detailed information about this dataset is listed in Table 1.

A decision stump is used as the base classifier for most of the experiments. We also evaluate the performance of the proposed boosting algorithm with the decision tree as the base classifier. All the 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 with its standard deviation is used as the evaluation metric. Finally, the parameter λ in the proposed boosting algorithm is automatically determined by the cross validation with 80/20% split of the training data.

4.2 Experiment (I): Effectiveness of MSmoothBoost

We tested the MSmoothBoost algorithm on the UCI datasets. A random code is used for the proposed boosting algorithm. To see the effectiveness of MSmoothBoost 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. A bold font is used to highlight the result when it is significantly better than all of its counterparts based on the statistical t-test at the confidence level 0.95.

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, MSmoothBoost is more effective than the other two boosting methods in improving the classification accuracy for multi-class learning. For dataset "glass" and "vehicle", the MSmoothBoost algorithm performs significantly better than the other three algorithms based on the t-test.

In order to see the robustness of MSmoothBoost with regard to training noises, we tested both the MSmoothBoost 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 of 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

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

Dataset	ECOC	AdaBoost.OC	BinBoost.Reg	MSmoothBoost
ecoli	18.7 ± 4.3	12.8 ± 4.9	9.3 ± 3.4	6.7 ± 3.1
wine	27.2 ± 6.1	20.3 ± 4.5	21.2 ± 3.0	$\mathbf{17.1 \pm 3.4}$
pendigit	15.6 ± 4.4	13.9 ± 3.6	12.0 ± 3.8	$\mathbf{6.4 \pm 2.5}$
iris	8.8 ± 4.9	9.7 ± 4.5	7.1 ± 3.1	8.0 ± 3.4
glass	54.7 ± 4.7	55.0 ± 6.2	49.9 ± 2.8	$\mathbf{44.5 \pm 1.4}$
vehicle	32.7 ± 4.5	35.0 ± 4.4	31.1 ± 2.7	$\mathbf{23.3 \pm 1.9}$
yeast	46.9 ± 5.8	42.5 ± 6.2	37.5 ± 3.8	37.4 ± 4.3

A *bold font* is used to highlight the result when it is significantly better than its counterparts based on the t-test at the confidence level 0.95

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 MSmoothBoost 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 MSmoothBoost employ certain mechanisms to alleviate the overfitting problem while AdaBoost.OC does not. Finally, comparing the MSmoothBoost algorithm to BinBoost.Reg, we found that MSmoothBoost is more resilient to training noise. For example, for dataset "pendigit", given 20% training noise, the classification error for MSmoothBoost is only 6.4% while the classification error for BinBoost.Reg is over 12%. In fact, it performs significantly better than the other three methods over four UCI datasets. Although BinBoost.Reg employs the regularized boosting algorithm as its binary classifier, it is less effective than MSmoothBoost because the regularized boosting algorithm is designed for binary classification, not for multi-class learning. Based on the above observation, we conclude that MSmoothBoost is effective for multi-class learning and is robust to training noise.

4.3 Experiment (II): Impact of λ on MSmoothBoost

In these experiments, we examine how the smoothing parameter will affect the classification error of MSmoothBoost. In the first experiment, we test the effect of λ on the clean UCI datasets. Table 4 summarizes the classification errors of the MSmoothBoost algorithm with λ varied from 0.0 to 1.0. Note that MSmoothBoost of $\lambda = 0$ corresponds to the AdaBoost.OC algorithm, as already discussed before.

By comparing the results of using smoothing (i.e., $\lambda > 0$) to the cases without smoothing (i.e., $\lambda = 0$), we observe that no matter what value λ is used, in general smoothing is helpful in reducing the classification error of boosting. In fact, for most datasets, the classification errors of MSmoothBoost are relatively stable when different values of λ are used. For instance, the classification error of "iris" is only changed between 4.5% and 5.7% when λ is varied from 0.1 to 1.0. The exceptional case is dataset "vehicle", whose classification error is increased significantly from 21.5% to 27.7% when λ is varied from 0.1 to 1.0. However, even for the worst cases, the MSmoothBoost algorithm is able to produce

Table 4 Classification errors (%) of MSmoothBoost using different smoothing parameter λ for the original UCI datasets. $\lambda = 0$ corresponds to the AdaBoost.OC algorithm

λ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ecoli	3.5	3.0	2.8	4.1	3.8	3.9	4.1	3.4	3.2	5.2	4.4
wine	16.0	14.7	15.5	13.2	14.0	13.4	13.3	15.2	13.9	14.8	13.7
pendigit	2.7	1.3	2.1	1.7	1.1	1.8	1.4	1.7	2.6	2.7	1.6
iris	5.2	4.5	4.7	5.2	5.3	5.0	5.7	5.7	4.7	4.7	4.7
glass	49.7	43.8	44.4	45.4	44.9	44.7	45.7	44.8	47.6	48.0	47.6
vehicle	26.0	21.5	21.5	22.4	22.5	23.4	23.5	25.0	25.9	27.7	26.2
yeast	36.4	35.0	34.6	36.7	36.3	35.5	35.3	35.9	35.1	34.9	35.8

Table 5 Classification errors (%) of MSmoothBoost using different smoothing parameter λ for the "corrupted" UCI datasets with 20% training errors. $\lambda = 0$ corresponds to the AdaBoost.OC algorithm

λ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ecoli	12.8	5.1	6.8	6.3	6.5	5.4	7.8	7.7	6.8	6.6	7.0
wine	21.2	17.5	17.3	16.4	17.0	17.2	16.2	17.1	17.3	17.5	17.6
pendigit	13.9	7.1	6.3	5.8	5.5	4.8	4.5	6.6	4.8	6.1	5.1
iris	9.7	6.2	6.7	7.0	7.5	7.5	7.3	8.0	7.7	8.3	8.2
glass	55.0	45.4	45.6	45.0	46.3	45.9	45.4	46.0	46.3	47.6	48.2
vehicle	35.0	24.0	24.6	24.8	24.9	25.9	26.2	26.1	26.6	27.0	28.2
yeast	42.5	36.6	37.8	37.5	38.3	38.8	38.0	38.3	38.5	38.1	38.6

classification errors that are comparable to the AdaBoost.OC algorithm. This is consistent with the study of the regularization approach in data classification. For instance, people have found that introducing regularization into the maximum entropy model usually reduces the classification error (Nigam et al. 1999). In the meantime, a number of studies (e.g., Zhang 2003) also indicated that although the reduction in classification error usually depends on the amount of regularization introduced into the classification model, it is usually true that any reasonable amount of regularization can improve the classification performance.

In the second experiment, we test the impact of the smoothing parameter λ for the corrupted UCI datasets with 20% of training examples whose class labels are incorrect. Table 5 summarizes the classification errors of the MSmoothBoost algorithm for the corrupted datasets when λ is varied from 0.0 to 1.0. Similar to the previous study, we observe that the classification error of MSmoothBoost for the corrupted datasets does not experience a significant change when we vary λ from 0.1 to 1.0. Second, for all the values of λ that are tested in this experiment, the MSmoothBoost algorithm is able to outperform the AdaBoost.OC (i.e., $\lambda = 0$) across all the datasets.

Based on the two sets of experiments, we conclude that MSmoothBoost is effective in reducing the classification errors and dealing with the noisy training examples. We also conclude that the performance of the MSmoothBoost algorithm is relatively stable across different λ.

Table 6 Classification errors (%) for the original UCI datasets

Dataset	ECOC	AdaBoost.OC	MSmoothBoost
ecoli	9.7 ± 3.6	3.1 ± 3.2	3.1 ± 2.5
wine	15.4 ± 2.8	16.2 ± 2.5	16.0 ± 2.5
pendigit	4.0 ± 2.0	3.2 ± 2.0	3.4 ± 2.4
iris	6.5 ± 2.5	6.0 ± 2.8	5.5 ± 3.2
glass	40.5 ± 2.0	45.3 ± 1.7	40.8 ± 2.3
vehicle	3.6 ± 0.7	3.0 ± 0.7	3.9 ± 0.8
yeast	27.4 ± 2.9	24.7 ± 2.1	24.1 ± 2.2

Decision tree is used as the base classifier

Table 7 Classification errors (%) for the "corrupted" UCI datasets with 20% training errors

Dataset	ECOC	AdaBoost.OC	MSmoothBoost
ecoli	20.8 ± 4.1	13.5 ± 5.2	13.8 ± 4.0
wine	16.5 ± 3.8	23.2 ± 4.5	15.8 ± 3.5
pendigit	$\mathbf{9.5 \pm 4.8}$	17.6 ± 4.9	17.7 ± 5.0
iris	9.8 ± 5.0	12.3 ± 5.2	9.3 ± 4.3
glass	42.1 ± 1.7	53.0 ± 4.7	42.7 ± 1.3
vehicle	$\mathbf{5.8 \pm 1.0}$	15.3 ± 1.8	8.1 ± 0.7
yeast	34.4 ± 3.0	35.8 ± 2.9	$\mathbf{31.0 \pm 1.9}$

Decision tree is used as the base classifier. A *bold font* is used to highlight the result when it is significantly better than its counterparts based on the t-test at the confidence level 0.95

4.4 Experiment (III): The Impact of Base Classifier

In these experiment, we evaluate the MSmoothBoost algorithm by using the decision tree as the base classifier. The goal of this experiment is to examine the effect of base classifier on the performance of the proposed boosting algorithm.

In the first experiment, we test the ECOC method, the AdaBoost.OC algorithm, and the MSmoothBoost algorithm on the original UCI dataset. All the three algorithms use the decision tree as their base classifier. The classification results are summarized in Table 6. First, we observe that for most of the datasets, both boosting algorithms are unable to reduce the classification error when compared to the simple ECOC method. The only exception case is dataset "ecoli" in which the boosting algorithms are able to reduce the classification error from 9.7% to around 3%. This is in contrast to the case when the decision stump is used as the base classifier in which we usually observe a significant improvement when the boosting algorithms are applied. Second, we observe that for dataset "glass", instead of reducing classification error, the AdaBoost.OC algorithm in fact increases the classification error substantially, from around 40.5% to 45.3%. In contrast, the MSmoothBoost is able to achieve the same classification error as the ECOC method. This result indicates that even for clean datasets, the AdaBoost.OC algorithm can still overfit training examples when the underlying base classifier is a strong classifier, whereas the MSmoothBoost algorithm is able to avoid the overfitting problem by using the smoothing technique.

In the second experiment, we evaluate the three multi-class classification algorithms using the decision tree as the base classifier for the corrupted UCI dataset with 20% training noise. The classification errors of the three algorithms are listed in Table 7. First, we observe that for almost all datasets except "ecoli" and "yeast", the AdaBoost.OC algorithm performs substantially worse than the ECOC method. For instance, the classification error of dataset "vehicle" is degraded from 5.8% to 15.3% when AdaBoost.OC is used. This result

Table 8 Classification errors (%) for the physiological dataset

A *bold font* is used to highlight the result that is statistically significantly better than its counterparts

Base Classifier	ECOC	AdaBoost.OC	MSmoothBoost
Decision Stump	33.5 ± 1.0	35.1 ± 1.4	32.7 ± 1.3
Decision Tree	32.7 ± 0.7	31.1 ± 1.1	$\mathbf{30.1 \pm 0.9}$

is consistent with the previous observation for the corrupted UCI datasets when the decision stump is used as the base classifier. Both results indicate the overfitting problem with the AdaBoost.OC algorithm when the training examples are noisy. On the other hand, the MSmoothBoost algorithm is able to alleviate the problem of overfitting substantially across almost all the datasets. For a number of datasets, the MSmoothBoost is able to achieve a similar performance as the ECOC method when AdaBoost.OC degrades the performance. For dataset "yeast", the MSmoothBoost algorithm is even able to reduce the classification error significantly, from 34.4% to 31.0%. The only exception cases are dataset "pendigit" and "vehicle", in which both boosting algorithm did poorly compared to the ECOC method. All these results indicate that in general the MSmoothBoost is more robust and effective than the AdaBoost.OC algorithm. These results also indicate that boosting algorithms are usually less effective when a strong base classifier is used.

4.5 Experiment (VI): The Physiological Dataset

The classification errors of the physiological dataset by the ECOC method, AdaBoost.OC, and MSmoothBoost using the random codes are listed in Table 8. Two base classifiers, namely decision stump and decision tree, are used in the experiment. First, as indicated in Table 8, AdaBoost.OC appears to overfit training examples when decision stump is used as the base classifier. It performs significantly worse than the ECOC method according to the statistical t-test at the confidence level 0.95. In contrast, MSmoothBoost appears to be more robust than AdaBoost.OC and is able to achieve slightly better performance than the ECOC method when the decision stump is used as the base classifier. Second, both MSmoothBoost and AdaBoost.OC are able to reduce the classification error of the ECOC method when decision tree is used as the base classifier. According to the t-test, MSmoothBoost performs significantly better than AdaBoost.OC.

5 Conclusion

In this paper, we propose a new boosting algorithm, named "MSmoothBoost", that is able to boost binary classifiers for multi-class learning problems. In particular, it addresses the overfitting problem of the AdaBoost.OC algorithm by introducing a smoothing mechanism into the boosting algorithm. We show theoretically that the proposed boosting algorithm is able to reduce the training error in an exponential fashion, and the smoothing parameter λ is able to reduce the marginal training errors by balancing the tradeoff between the empirical part of the error bound and the marginal part of the error bound. A set of extensive experiments have been conducted to evaluate the effectiveness and the robustness of the proposed boosting algorithm. Our empirical studies have shown that the proposed MSmoothBoost algorithm performs better than the AdaBoost.OC algorithm for both clean data and noisy data. In the future, we plan to extend this work to multi-label classification where each data point can be assigned with multiple labels.

Appendix 1 Proof of Theorem 1

Proof First, according to (3), we have

$$\frac{1}{m}\sum_{i=1}^{m} I\Big(\max_{y\neq y_i}\Phi_T(y,x_i)-\Phi_T(y_i,x_i)\Big) \leq \frac{1}{m}\sum_{i=1}^{m}\frac{(1+\lambda)\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))}{\exp(\Phi_T(y_i,x_i))+\lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))}.$$

Let the upper bound in the above equation denoted by τ_T. Note that using notation $\mu_T(y|x)$, τ_T can also be written as:

$$\tau_T = \frac{1+\lambda}{m}\sum_{i=1}^{m}\sum_{y\neq y_i}\mu_T(y|x_i). \tag{16}$$

Then, according to (6) and (7), we have τ_{T+1} upper bounded by

$$\begin{aligned}\tau_{T+1} \leq{}& \frac{\lambda+1}{m}\sum_{i=1}^{m}\lambda\Big(\sum_{y\neq y_i}\mu_T(y|x_i)\Big)^2 \\ &+\frac{1+\lambda}{2m}\sum_{i=1}^{m}\mu_T(y_i|x_i)\sum_{y\neq y_i}\mu_T(y|x_i)(1+f(y)f(y_i)) \\ &+\frac{1+\lambda}{2m}\sum_{i=1}^{m}\mu_T(y_i|x_i)\exp(-2\alpha h(x_i)f(y_i)) \\ &\times\sum_{y}\mu_T(y|x_i)(1-f(y)f(y_i)).\end{aligned} \tag{17}$$

By rewriting $(1+f(y)f(y_i))/2 = 1-(1-f(y)f(y_i))/2$ and using the relation $\mu_T(y_i|x_i)+\lambda\sum_{y\neq y_i}\mu_T(y|x_i)=1$, we have the upper bound in the above equation simplified as:

$$\begin{aligned}\tau_{T+1} \leq{}& \frac{\lambda+1}{m}\sum_{i=1}^{m}\sum_{y\neq y_i}\mu_T(y|x_i) \\ &-\frac{1+\lambda}{2m}\sum_{i=1}^{m}\mu_T(y_i|x_i)\sum_{y\neq y_i}\mu_T(y|x_i)(1-f(y)f(y_i)) \\ &+\frac{1+\lambda}{2m}\sum_{i=1}^{m}\mu_T(y_i|x_i)\exp(-2\alpha h(x_i)f(y_i))\sum_{y}\mu_T(y|x_i)(1-f(y)f(y_i)) \\ ={}& \tau_T-\frac{1+\lambda}{2m}U_{T+1}Z_{T+1}+\frac{1+\lambda}{m}U_{T+1}Z_{T+1}\sqrt{\epsilon_{T+1}(1-\epsilon_{T+1})}.\end{aligned} \tag{18}$$

Note that ν_t defined in (14) can also be written as:

$$\nu_t = \frac{(1+\lambda)U_t Z_t}{m\tau_t}.$$

Hence, the bound in (18) can be further written as:

$$\begin{aligned}\tau_{T+1} &\leq \tau_T\left(1 - \frac{\nu_{T+1}}{2}\left(1 - \sqrt{1-4\gamma_{T+1}^2}\right)\right)\\ &\leq \tau_T(1-\nu_{T+1}\gamma_{T+1}^2).\end{aligned}$$

Applying the above inequality recursively, we have

$$\frac{1}{m}\sum_{i=1}^{m} I\left(\max_{y\neq y_i}\Phi_T(y,x_i) - \Phi_T(y_i,x_i)\right) \leq \tau_T \leq \prod_{t=1}^{T}(1-\nu_t\gamma_t^2). \qquad \square$$

Appendix 2 Proof of Theorem 2

Proof Similar to Theorem 1, the marginal training error can be bounded by the following expression:

$$\begin{aligned}&\frac{1}{m}\sum_{i=1}^{m} I\left(\max_{y\neq y_i}\hat{\Phi}_T(y,x_i) - \hat{\Phi}_T(y_i,x_i) + \theta\right)\\ &\leq \frac{1+\lambda}{m}\sum_{i=1}^{m}\frac{\sum_{y\neq y_i}\exp(\Phi_T(y,x_i) + \theta\sum_{t=1}^{T}\alpha_t)}{\exp(\Phi_T(y_i,x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i) + \theta\sum_{t=1}^{T}\alpha_t)}\\ &\leq \frac{1+\lambda}{m}\sum_{i=1}^{m}\frac{\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))}{\exp(\Phi_T(y_i,x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))}\\ &\quad + \frac{1+\lambda}{m}\sum_{i=1}^{m}\frac{\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))(\exp(\theta\sum_{t=1}^{T}\alpha_t)-1)}{\left(\begin{array}{l}\exp(\Phi_T(y_i,x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i)) +\\ \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))(\exp(\theta\sum_{t=1}^{T}\alpha_t)-1)\end{array}\right)}\end{aligned} \tag{19}$$

where the first inequality is obtained in a similar way as (3).

Using the convexity of reciprocal function $((p_1a_1 + p_2a_2)^{-1} \leq p_1/a_1 + p_2/a_2$, with $p_1 = 1/(1+\lambda)$ and $p_2 = \lambda/(1+\lambda)$), we have

$$\begin{aligned}&\left(\begin{array}{l}\exp(\Phi_T(y_i,x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i)) +\\ \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))(\exp(\theta\sum_{t=1}^{T}\alpha_t)-1)\end{array}\right)^{-1}\\ &\leq \frac{1}{(1+\lambda)^2(\exp(\Phi_T(y_i,x_i)) + \lambda\sum_{y\neq y_i}\exp(\Phi_T(y,x_i)))}\\ &\quad + \frac{\lambda}{(1+\lambda)^2\sum_{y\neq y_i}\exp(\Phi_T(y,x_i))(\exp(\theta\sum_{t=1}^{T}\alpha_t)-1)}.\end{aligned}$$

Using the above result, the bound in (19) can be simplified as:

$$\begin{aligned}
&\frac{1}{m}\sum_{i=1}^{m} I\left(\max_{y\neq y_i}\hat{\Phi}_T(y,x_i)-\hat{\Phi}_T(y_i,x_i)+\theta\right)\\
&\quad\leq \tau_T+\tau_T\frac{\exp(\theta\sum_{t=1}^{T}\alpha_t)-1}{(1+\lambda)^2}+\frac{\lambda}{1+\lambda}\\
&\quad\leq \frac{1}{(1+\lambda)^2}\left(\lambda(2+\lambda)+\prod_{t=1}^{T}\left(\frac{1+\gamma_t}{1-\gamma_t}\right)^{4\theta}\right)\prod_{t=1}^{T}(1-\nu_t\gamma_t^2)+\frac{\lambda}{1+\lambda}.
\end{aligned}$$

The last step in the above derivation uses the relationship $\alpha_t=\frac{1}{4}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ and the result in Theorem 1. □

References

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

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting and variants. *Machine Learning*, *36*, 105–139.

Berger, A. (1999). Error-correcting output coding for text classification. In *IJCAI'99: Workshop on machine learning for information retrieval*.

Bredensteiner, E., & Bennett, K. (1999). Multicategory classification by support vector machines. *Computational Optimization and Application*, *12*, 35–46.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167.

Crammer, K., & Singer, Y. (2000). On the learnability and design of output codes for multiclass problems. *Machine Learning*, *47*(2–3), 201–233.

Dietterich, T.G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, *40*(2), 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. (2001). An adaptive version of the boost by majority algorithm. *Machine Learning*, *43*, 293–318.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning* (pp. 148–156).

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

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, *28*(2), 337–374.

Ghani, R. (2000). Using error-correcting codes for text classification. In *Proceedings of the seventeenth international conference on machine learning* (pp. 303–310).

Grove, A., & Schuurmans, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. In *Proceedings of the fifteenth national conference on artificial intelligence* (pp. 692–699).

Hastie, T., & Tibshirani, R. (1998). Classification by pairwise coupling. *Annals of Statistics*, *26*(2), 451–471.

Hill, R. (1986). *A first course in coding theory*. Oxford: Oxford University Press.

Hsu, C., & Lin, C. (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425.

Jin, R., Liu, Y., Si, L., Carbonell, J., & Hauptmann, A. (2003). A new boosting algorithm using input-dependent regularizer. In *Proceedings of twentieth international conference on machine learning*.

Lebanon, G., & Lafferty, J. (2001). Boosting and maximum likelihood for exponential models. In *Advances in neural information processing systems 14*.

Lee, Y., Lin, Y., & Wahba, G. (2002). Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. In *Advances in neural information processing systems 15*.

Masulli, F., & Valentini, G. (2003). Effectiveness of error correcting output coding methods in ensemble and monolithic learning machines. *Pattern Analysis and Applications*, *6*(4), 285–300.

McCallum, A., & Nigam, K., (1998). A comparison of event models for Naive Bayes text classification. In *AAAI-98 workshop on learning for text categorization*.

Nigam, K., Lafferty, J., & McCallum, A. (1999). Using maximum entropy for text classification. In *IJCAI'99 workshop on machine learning for information filtering*.

Platt, J., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In *Advances in neural information processing systems 13*.

Ratsch, G., Onoda, T., & Muller, K. (1998). An improvement of AdaBoost to avoid overfitting. In *Advances in neural information processing systems 11*.

Ratsch, G., Onoda, T., & Muller, K. (1999). Regularizing AdaBoost. In *Advances in neural information processing systems 12*.

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

Schapire, R. (1997). Using output codes to boost multiclass learning problems. In *Proceedings of the fourteenth international conference on machine learning* (pp. 313–321).

Schapire, R. (1999). Theoretical views of boosting and applications. In *Proceedings of the tenth international conference on algorithmic learning theory* (pp. 13–25).

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

Servedio, R. (2003). Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, *4*, 633–648.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Weston, J., & Watkins, C. (1999). Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, University of London.

Zhang, T. (2003). An infinity-sample theory for multi-category large margin classification. In *Advances in neural information processing systems 16*.

Zhu, J. & Hastie, T. (2001). Kernel logistic regression and the import vector machine. In *Advances in neural information processing systems 14* (pp. 1081–1088).