Unifying the Error-Correcting and Output-Code AdaBoost within the Margin Framework

Yijun Sun[1,2] SUN@DSP.UFL.EDU
Sinisa Todorovic[2] SINISHA@UFL.EDU
Jian Li[2] LI@DSP.UFL.EDU
Dapeng Wu[2] WU@ECE.UFL.EDU

[1]Interdisciplinary Center for Biotechnology Research, University of Florida, Gainesville, FL 32611, USA
[2]Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL 32611, USA

Abstract

In this paper, we present a new interpretation of AdaBoost.ECC and AdaBoost.OC. We show that AdaBoost.ECC performs stage-wise functional gradient descent on a cost function, defined in the domain of margin values, and that AdaBoost.OC is a shrinkage version of AdaBoost.ECC. These findings strictly explain some properties of the two algorithms. The gradient-minimization formulation of AdaBoost.ECC allows us to derive a new algorithm, referred to as AdaBoost.SECC, by explicitly exploiting shrinkage as regularization in AdaBoost.ECC. Experiments on diverse databases confirm our theoretical findings. Empirical results show that AdaBoost.SECC performs significantly better than AdaBoost.ECC and AdaBoost.OC.

1. Introduction

A review of the literature indicates that the majority of available pattern classification algorithms are designed only for binary classification problems. Some of them can be easily generalized to solve multiclass problems (e.g., C4.5 [Quinlan, 1993]), while for others the extension is not straightforward. Therefore, it is important to investigate how to use well studied binary classification algorithms for solving multiclass problems. One of the possible approaches is to first decompose a multiclass into several binary problems by using a code matrix, then, to apply binary classifiers to these binary problems, and finally to combine the binary outcomes toward the final classification. The outlined approach can be systematized into three sub-categories [Crammer & Singer, 2000]: (1) given a set of binary classifiers, find a code matrix that yields small empirical loss; (2) given a code matrix, find a set of binary classifiers that result in small empirical loss; (3) find both a set of binary classifiers and a code matrix simultaneously that produce small empirical loss. Since herein we assume that binary classifiers are not known in advance, we omit considerations of the first category. A majority of the existing algorithms belongs to the second category in which the underlying dependence between the constructed binary classifiers and the fixed code matrix is not explicitly accounted for. This problem is discussed in [Allwein et al., 2000], where five different output codes are compared for a variety of datasets, with indecisive answers as to which output code is the best. The results in [Allwein et al., 2000] suggest that finding the optimal code matrix and binary classifiers simultaneously is the best strategy.

The third category, however, has been shown to be NP-hard [Crammer & Singer, 2000]. To alleviate this problem, a number of sub-optimal algorithms have been proposed in the literature, of which we are particularly interested in those formulated within the AdaBoost framework – more specifically, output-code AdaBoost (AdaBoost.OC) [Schapire, 1997], and error-correcting code AdaBoost (AdaBoost.ECC) [Guruswami & Sahai, 1999]. Here, the columns of the code matrix and binary hypothesis functions are generated alternatively, in a specified number of iteration steps. Thereby, the underlying dependence between the code matrix and binary classifiers is exploited in a stage-wise manner.

AdaBoost.OC and AdaBoost.ECC have been successfully applied to a number of standard multiclass problems. For both algorithms, the upper theoretical bounds of the training error have been derived. Yet, a mathematically rigorous formulation of how AdaBoost.OC and AdaBoost.ECC decrease the classification error has not to date been proposed. In addition, the relationship between these two algorithms,

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

as well as the algorithms' behavior in the case of noise-corrupted data are not fully examined in the literature. In fact, current understanding of the two algorithms, as reported in the literature, might even mislead practitioners to choose a wrong algorithm between the two for a given application. For example, in [Guruswami & Sahai, 1999], AdaBoost.ECC is said to outperform AdaBoost.OC, which, as we show in this paper, is not true for many settings. The aforementioned missing links in the theoretical development of AdaBoost.OC and AdaBoost.ECC motivated us to conduct the research reported in this paper.

We present a new interpretation of the two algorithms based on the analogy of boosting to steepest-descent minimization [Mason et al., 2000, Breiman, 1999, Friedman et al., 2000]. More precisely, we show that AdaBoost.ECC performs stage-wise functional gradient descent on a cost function, defined in the domain of margin values. We further prove that AdaBoost.OC is a shrinkage version of AdaBoost.ECC. This theoretical analysis allows us to derive the following results. First, we formulate and strictly prove several properties of AdaBoost.ECC and AdaBoost.OC, including the relationship between their convergence training-error rates, and their performances in noisy regimes. Second, we show how to simplify the computation of AdaBoost.OC by avoiding the redundant calculation of pseudo-loss. Third, we derive the shrinkage version of AdaBoost.ECC, which we refer to as AdaBoost.SECC. This novel algorithm naturally arises from the gradient-descent formulation of AdaBoost.ECC, where a shrinkage parameter can be used as a regularization parameter, similar to introducing the learning rate in neural networks.

We also study the algorithms' behavior in the presence of mislabeled training data. Mislabeling noise is a critical problem for many applications, where preparing a good training dataset is a challenging task. Indeed, human interpreters are often faced with hard-to-classify cases, which may cause erroneous human supervision. As a result, the training set may contain a significant number of mislabeled data. These considerations were also examined for two-class AdaBoost in [Dietterich, 2000].

The experimental results support our theoretical findings. In a very likely event, when for example 10% of training patterns are mislabeled, AdaBoost.OC outperforms AdaBoost.ECC. Moreover, in the presence of mislabeling noise, AdaBoost.SECC converges fastest to the smallest test error, as compared to AdaBoost.ECC and AdaBoost.OC.

2. Output Coding

First, we briefly explain the output coding method for solving multiclass classification problems [Dietterich & Bakiri, 1995, Allwein et al., 2000]. Suppose we are given a training dataset $\mathcal{D}=\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}\in\mathcal{X}\times\mathcal{Y}$ where $\mathcal{X}$ is the pattern space and $\mathcal{Y}=\{1,\cdots,C\}$ is the label space. To decompose the multiclass problem into several binary ones, a code matrix $\mathbf{M}\in\{\pm1\}^{C\times T}$ is introduced, where T is the length of a code word. Here, $M(c)$ denotes the c-th row, that is, a code word for class c, and $M(c,t)$ denotes an element of the code matrix. Each column of $\mathbf{M}$ defines a binary partition of C classes over data samples – the partition on which a binary classifier is trained. After T training steps, the output coding method produces a final classifier $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}),\cdots,f_T(\mathbf{x})]^{\mathrm{T}}$, where $f_t(\mathbf{x}) : \mathbf{x}\rightarrow\mathbb{R}$. When presented an unseen sample $\mathbf{x}$, the output coding method predicts the label y^*, such that the code word $M(y^*)$ is the "closest" to the prediction $\mathbf{f}(\mathbf{x})$, with respect to a specified decoding strategy. In this paper, we use the loss-based decoding strategy [Allwein et al., 2000], given by

$$y^* = \arg\min_{y\in\mathcal{Y}} \sum_{t=1}^{T} \exp\left(-M(y,t)f_t(\mathbf{x})\right). \quad (1)$$

3. AdaBoost.ECC and AdaBoost.OC

It has been empirically observed that AdaBoost can effectively increase the margin [Schapire et al., 1998]. For this reason, since the invention of AdaBoost, it has been conjectured that in the limit AdaBoost achieves the same solution as a Linear Programming (LP) problem in which the margin is directly optimized [Grove & Schuurmans, 1998]. In the recent paper [Rudin et al., 2004], however, the equivalence of the two algorithms has been proven not to hold always. Nevertheless, the two algorithms are connected in the sense that they try to maximize the margin. We make use of this connection, by employing the results obtained within the LP framework to define a cost function over the domain of margin values, upon which AdaBoost.ECC performs a stage-wise gradient descent.

We begin by defining the sample margin $\rho(\mathbf{x}_n)$ as

$$\rho(\mathbf{x}_n) \triangleq \min_{\{c\in\mathcal{Y}, c\neq y_n\}} \Delta(M(c),\mathbf{f}(\mathbf{x}_n)) - \Delta(M(y_n),\mathbf{f}(\mathbf{x}_n)), \quad (2)$$

where $\Delta(\cdot)$ is a distance measure. Maximization of $\rho(\mathbf{x}_n)$ in Eq. (2) can be interpreted as finding $\mathbf{f}(\mathbf{x}_n)$ close to the code word of the true label, while at the same time distant from the code word of the most confused class. For the purposes of this paper, we specify $\Delta(M(c),\mathbf{f}(\mathbf{x}))\triangleq\|M(c)-\mathbf{f}^{\mathrm{T}}(\mathbf{x})\|^2$, yielding

$$\rho(\mathbf{x}_n) = 2M(y_n)\mathbf{f}(\mathbf{x}_n) - \max_{\{c\in\mathcal{Y}, c\neq y_n\}} \{2M(c)\mathbf{f}(\mathbf{x}_n)\}. \quad (3)$$

Hence, maximization of $\rho(\mathbf{x}_n)$ can be formulated as an optimization problem:

$$\begin{array}{ll}\max & \rho\,, \\ \text{s.t.} & M(y_n)\mathbf{f}(\mathbf{x}_n) - \max\limits_{\{c\in\mathcal{Y}, c\neq y_n\}}\{M(c)\mathbf{f}(\mathbf{x}_n)\} \geq \rho, \\ & n=1,\cdots,N.\end{array} \quad (4)$$

Herein, we are particularly interested in finding $\mathbf{f}(\mathbf{x})$ of the following form:

$$\mathbf{f}(\mathbf{x}) = \frac{[\alpha_1 h_1(\mathbf{x}),\cdots,\alpha_T h_T(\mathbf{x})]^{\mathrm{T}}}{\sum_{t=1}^{T}\alpha_t} = \frac{\mathbf{F}(\mathbf{x})}{\sum_{t=1}^{T}\alpha_t}, \quad (5)$$

where the weights $\forall t,\ \alpha_t \geq 0$, and the hypothesis functions $h_t(\mathbf{x}) : x \rightarrow \{\pm 1\}$. From Eqs. (4) and (5), we derive

$$\begin{array}{ll}\max & \rho\,, \\ \text{s.t.} & \sum\limits_{t=1}^{T}\frac{\alpha_t}{\sum_{t'=1}^{T}\alpha_{t'}}(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n) \geq \rho, \\ & n=1,\cdots,N,\ \ c=1,\cdots,C,\ \ c\neq y_n,\ \ \boldsymbol{\alpha}\geq 0.\end{array} \quad (6)$$

In light of the connection between LP and AdaBoost, it appears reasonable to define a new cost function, which is optimized by a multiclass AdaBoost algorithm, as follows:

$$\begin{aligned} G &\triangleq \sum_{n=1}^{N}\sum_{\{c=1,c\neq y_n\}}^{C}\exp\left(-(M(y_n) - M(c))\mathbf{F}(\mathbf{x}_n)\right), \\ &= \sum_{n=1}^{N}\sum_{\substack{c=1,\\ c\neq y_n}}^{C}\exp(-\sum_{t=1}^{T}\alpha_t(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n)). \end{aligned} \quad (7)$$

Indeed, the following theorem shows that AdaBoost.ECC optimizes the above cost function.

Theorem 1. *AdaBoost.ECC performs a stage-wise functional gradient descent procedure on the cost function given by Eq. (7).*

Proof: In Fig. 1, we present the pseudo code of the symmetric version of AdaBoost.ECC, as proposed in [Guruswami & Sahai, 1999]. By comparing the expressions for: (i) the data-sampling distribution $\mathbf{d}_t$ (Step (5)), and (ii) the weights α_t (Step (8)), with those obtained from the minimization of G, we prove the theorem. For the time being, we assume that the code matrix $\mathbf{M}$ is given, the generation of which is discussed later.

After $(t-1)$ iteration steps, AdaBoost.ECC produces $\mathbf{F}_{t-1}(\mathbf{x}_n) = [\alpha_1 h_1(\mathbf{x}_n),\cdots,\alpha_{t-1}h_{t-1}(\mathbf{x}_n),0,\cdots,0]^{\mathrm{T}}$. In the t-th iteration step, the goal of the algorithm is to compute the t-th entry of $\mathbf{F}(\mathbf{x})$, given by $\alpha_t h_t(\mathbf{x}_n)$, and to update $\mathbf{F}(\mathbf{x})$ as $\mathbf{F}_t(\mathbf{x}_n) = [\alpha_1 h_1(\mathbf{x}_n),\cdots,\alpha_t h_t(\mathbf{x}_n),0,\cdots,0]^{\mathrm{T}}$. From Eq. (7), the negative functional derivative of G with respect to $\mathbf{F}_{t-1}(\mathbf{x})$, if $\mathbf{x}=\mathbf{x}_n$, is computed as $-\nabla_{\mathbf{F}_{t-1}(\mathbf{x})}G|_{\mathbf{x}=\mathbf{x}_n} = \sum_{c=1,c\neq y_n}^{C}(M(y_n)-M(c))^{\mathrm{T}}\mathrm{e}^{-(M(y_n)-M(c))\mathbf{F}_{t-1}(\mathbf{x}_n)}$. Similar to the derivation steps in [Friedman, 2001], it is straightforward to show that h_t should be selected from $\mathcal{H}$ by maximizing its correlation with the t-th component of $-\nabla_{\mathbf{F}_{t-1}(\mathbf{x})}G\big|_{\mathbf{x}=\mathbf{x}_n}$ as

$$h_t = \arg\max_{h\in\mathcal{H}}\{\sum_{n=1}^{N}\sum_{c=1,c\neq y_n}^{C} h(\mathbf{x}_n)(M(y_n,t)-M(c,t)) \cdot \mathrm{e}^{-(M(y_n)-M(c))\mathbf{F}_{t-1}(\mathbf{x}_n)}\}. \quad (8)$$

To facilitate the computation of Eq. (8), we introduce the following terms:

$$\begin{aligned} V_t(n) &\triangleq \sum_{\substack{c=1,\\ c\neq y_n}}^{C}|M(y_n,t)-M(c,t)|\mathrm{e}^{-(M(y_n)-M(c))\mathbf{F}_{t-1}(\mathbf{x}_n)}, \\ d_t(n) &\triangleq V_t(n)/\textstyle\sum_{n=1}^{N}V_t(n) \triangleq V_t(n)/V_t\,. \end{aligned} \quad (9)$$

Note that V_t, given by Eq. (9), differs from U_t, defined in Step (4) in Fig. 1, by a constant. Also, note that $(M(y_n,t)-M(c,t))$ either equals zero, or has the same sign as $M(y_n,t)$. It follows that

$$\begin{aligned} h_t &= \arg\max_{h\in\mathcal{H}}\textstyle\sum_{n=1}^{N}V_t(n)\mathrm{sign}\left(M(y_n,t)\right)h(\mathbf{x}_n), && (10)\\ &= \arg\max_{h\in\mathcal{H}} V_t\textstyle\sum_{n=1}^{N}d_t(n)M(y_n,t)h(\mathbf{x}_n)\,, && (11)\\ &= \arg\max_{h\in\mathcal{H}} U_t\textstyle\sum_{n=1}^{N}d_t(n)M(y_n,t)h(\mathbf{x}_n)\,, && (12)\\ &= \arg\min_{h\in\mathcal{H}}\textstyle\sum_{n=1}^{N}\mathbf{I}(M(y_n,t)\neq h(\mathbf{x}_n))d_t(n)\,, && (13)\\ &= \arg\min_{h\in\mathcal{H}}\varepsilon\,, && (14)\end{aligned}$$

where ε is the training error. Once h_t is found, α_t can be computed by minimizing the intermediate cost function G_t. The derivative of G_t with respect to α_t reads

$$\frac{\partial G_t}{\partial \alpha_t} = -\textstyle\sum_{n=1}^{N}\sum_{c=1,c\neq y_n}^{C}(M(y_n,t)-M(c,t))h_t(\mathbf{x}_n) \cdot \mathrm{e}^{-\sum_{j=1}^{t}\alpha_j(M(y_n,j)-M(c,j))h_j(\mathbf{x}_n)}. \quad (15)$$

Note that $(M(y_n,t)-M(c,t))$ takes values in the set $\{0,2,-2\}$. Also, recall that $(M(y_n,t)-M(c,t))$ has the same sign as $M(y_n,t)$. From Eq. (9) we derive

$$\begin{aligned} -\frac{\partial G_t}{\partial \alpha_t} &= V_t\sum_{n=1}^{N}d_t(n)M(y_n,t)h_t(\mathbf{x}_n)e^{-2\alpha_t M(y_n,t)h_t(\mathbf{x}_n)}, \\ &= V_t\left(\sum_{n=1}^{N}\mathbf{I}(M(y_n,t)=h_t(\mathbf{x}_n))d_t(n)\mathrm{e}^{-2\alpha_t} - \sum_{n=1}^{N}\mathbf{I}(M(y_n,t)\neq h_t(\mathbf{x}_n))d_t(n)\mathrm{e}^{2\alpha_t}\right). \end{aligned} \quad (16)$$

From Eqs. (13) and (14), and $\partial G_t/\partial\alpha_t = 0$ we obtain

$$\alpha_t = \tfrac{1}{4}\ln[(1-\varepsilon_t)/\varepsilon_t]\,, \quad (17)$$

AdaBoost.ECC

(1) **Initialization**: given $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N \in \mathcal{X} \times \mathcal{Y}$, initialize $D_1(n,c) = \mathbf{I}(c \neq y_n)/N(C-1),\ n = 1{:}N,\ c = 1{:}C$; set the maximum number of iteration steps T.

(2) **for** $t = 1 : T$

(3) Define the t-th column of $\mathbf{M}$: $M_{\cdot t} \in \{-1, +1\}^{C \times 1}$;

(4) $U_t = \sum_{n=1}^N \sum_{c=1}^C D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;

(5) $d_t(n) = \frac{1}{U_t} \sum_{c=1}^C D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;

(6) Train the base learner from $\mathcal{D}$ with respect to distribution $\mathbf{d}_t$, and compute $h_t(\mathbf{x})$;

(7) $\varepsilon_t = \sum_{n=1}^N \mathbf{I}(M(y_n,t) \neq h_t(\mathbf{x}_n))d_t(n)$;

(8) $\alpha_t = \frac{1}{4} \ln[(1-\varepsilon_t)/\varepsilon_t]$;

(9) Update weights $\mathbf{D}_{t+1}$ as $D_{t+1}(n,c) = \frac{1}{Z_t} D_t(n,c) \exp(-\alpha_t(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n))$, where Z_t is a normalizing constant;

(10) **end**

(11) **Output**: $\mathbf{F}(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}), \cdots, \alpha_T h_T(\mathbf{x})]$.

Figure 1. Pseudo code of AdaBoost.ECC, as proposed in [Guruswami & Sahai, 1999].

which is equal to the expression given in Step (8), Fig. 1.

Finally we check the update rule for data-sampling distribution $\mathbf{d}_t$. By unravelling $D_t(n,c)$ in Step (9) in the pseudo code of AdaBoost.ECC (see Fig. 1), we derive: $D_t(n,c) = \frac{1}{Z_t} \exp(-(M(y_n) - M(c))\mathbf{F}_{t-1}(\mathbf{x}_n))$, if $c \neq y_n$, and $D_t(n,c) = 0$, if $c = y_n$ where Z_t is a normalization constant. By plugging $D_t(n,c)$ into Steps (4) and (5) in the pseudo code of AdaBoost.ECC, it is straightforward to show that the expressions for $\mathbf{d}_t$ in Step (5) and Eq. (9) are the same. ■

Now, we discuss how to generate the columns of the code matrix, denoted as $\mathbf{M}_{\cdot t}$. Recall that simultaneous optimization of both $\mathbf{M}_{\cdot t}$ and h_t is NP-hard. Both AdaBoost.OC and AdaBoost.ECC perform in fact a two-stage optimization in which $\mathbf{M}_{\cdot t}$ is first generated by maximizing U_t, given in Step (4) in Fig. 1, and then h_t is trained based on the binary partition defined by $\mathbf{M}_{\cdot t}$. In [Schapire, 1997,Guruswami & Sahai, 1999], this procedure is justified by showing that maximizing U_t decreases the upper bound of the training error. Maximizing U_t is a special case of the "Max-Cut" problem, which is known to be NP-complete. Herewith, for computing the approximate solution of the optimal $\mathbf{M}_{\cdot t}$, in our experiments we use the same approach as that used in [Schapire, 1997].

We point out that the proof of Theorem 1 provides for yet another interpretation of the outlined procedure. Ideally, in the t-th iteration step we want to find $\mathbf{M}_{\cdot t}$ and h_t simultaneously to optimize the correlation in Eq. (8), which however is NP-hard. Therefore, we resort to the two-stage optimization. It is evident from Eq. (12) that $\mathbf{M}_{\cdot t}$ should be generated such that U_t is maximized.

3.1. Relationship between AdaBoost.OC and AdaBoost.ECC

AdaBoost.ECC is derived from AdaBoost.OC on the algorithm level, as discussed in [Guruswami & Sahai, 1999]. However, the relationship between the two algorithms is not fully examined in the literature. The following theorem provides for a mathematical explanation of their relationship.

Theorem 2. *AdaBoost.OC is a shrinkage version of AdaBoost.ECC.*

Proof: We compare the expressions for the weights α_t, and the data-sampling distribution $\mathbf{d}_t$, to establish the relationship between AdaBoost.OC and AdaBoost.ECC. The pseudo code of AdaBoost.OC given in Fig. 2. In Step (7), a pseudo hypothesis function, $\tilde{h}_t(\mathbf{x})$, is constructed as

$$\tilde{h}_t(\mathbf{x}) = \{c \in \mathcal{Y} : h_t(\mathbf{x}) = M(c,t)\}. \tag{18}$$

Using $\tilde{h}_t(\mathbf{x})$, a pseudo-loss, $\tilde{\varepsilon}_t$, is computed in Step (8) as

$$\tilde{\varepsilon}_t = \frac{1}{2} \sum_{n=1}^N \sum_{c=1}^C D_t(n,c) \left(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)) \right), \tag{19}$$

where $D_t(n,c)$ is updated in Step (9). Note that:

$$\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)) = \frac{(M(c,t) - M(y_n,t))h_t(\mathbf{x}_n)}{2} + 1. \tag{20}$$

Therefore, from Eq. (19), we have

$$\begin{aligned} \tilde{\varepsilon}_t &= \frac{1}{4} \underbrace{\sum_{n=1}^N \sum_{c=1}^C D_t(n,c)(M(c,t) - M(y_n,t))h_t(\mathbf{x}_n)}_{r} + \frac{1}{2} \\ &= \frac{1}{4} r + \frac{1}{2} \Rightarrow r = 4(\tilde{\varepsilon}_t - \tfrac{1}{2}) \end{aligned} \tag{21}$$

Now, let us take a look at the pseudo code of AdaBoost.ECC given in Fig. 1. The training error, ε_t, is computed in Step (7) as

$$\begin{aligned} \varepsilon_t &= \sum_{n=1}^N \mathbf{I}(M(y_n,t) \neq h_t(\mathbf{x}_n))d_t(n) , \\ &= \tfrac{1}{2} - \tfrac{1}{2} \sum_{n=1}^N d_t(n) M(y_n,t) h_t(\mathbf{x}_n) . \end{aligned} \tag{22}$$

From Step (5), in Fig. 1, we have

$$\begin{aligned} \varepsilon_t &= \frac{1}{2} - \frac{\sum_{n=1}^N \sum_{c=1}^C D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))M(y_n,t)h_t(\mathbf{x}_n)}{2U_t} = \\ &= \frac{1}{2} - \frac{\sum_{n=1}^N \sum_{c=1}^C D_t(n,c)(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n)}{4U_t} = \\ &= \frac{1}{2} + \frac{1}{4U_t} r . \end{aligned} \tag{23}$$

By plugging Eq. (21) into Eq. (23), we get:

$$\varepsilon_t = \frac{1}{2} + \frac{1}{U_t}(\tilde{\varepsilon}_t - \frac{1}{2}) . \tag{24}$$

AdaBoost.OC

(1) **Initialization**: given $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \in \mathcal{X} \times \mathcal{Y}$, initialize $D^{(1)}(n,c) = \mathbf{I}(c \neq y_n)N(C-1),\ n = 1{:}N,\ c = 1{:}C$; set the maximum number of iteration steps T;

(2) **for** $t = 1 : T$

(3) Define the t-th column of $\mathbf{M}$: $M_{\cdot t} \in \{-1, +1\}^{C \times 1}$;

(4) $U_t = \sum_{n=1}^{N} \sum_{c=1}^{C} D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;

(5) $d_t(n) = \frac{1}{U_t} \sum_{c=1}^{C} D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;

(6) Train the base learner from $\mathcal{D}$ with respect to distribution $\mathbf{d}_t$, and compute $h_t(\mathbf{x})$;

(7) Define pseudo hypothesis: $\tilde{h}_t(\mathbf{x}) = \{c \in \mathcal{Y} : h_t(\mathbf{x}) = M(c,t)\}$;

(8) Compute pseudo error: $\tilde{\varepsilon}_t = \frac{1}{2} \sum_{n=1,c=1}^{N,C} D_t(n,c)(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)))$;

(9) $\tilde{\alpha}_t = \frac{1}{4} \ln[(1 - \tilde{\varepsilon}_t)/\tilde{\varepsilon}_t]$;

(10) Update weights $\mathbf{D}_{t+1}$ as $D_{t+1}(n,c) = \frac{1}{Z_t} D_t(n,c) \exp(2\tilde{\alpha}_t(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n))))$, where Z_t is a normalizing constant;

(11) **end**

(12) **Output**: $\mathbf{F}(\mathbf{x}) = [\tilde{\alpha}_1 h_1(\mathbf{x}), \cdots, \tilde{\alpha}_T h_T(\mathbf{x})]$.

Figure 2. Pseudo code of AdaBoost.OC, as proposed in [Schapire, 1997].

From Eq. (24), we observe that $\varepsilon_t \leq \frac{1}{2}$ if and only if $\tilde{\varepsilon}_t \leq \frac{1}{2}$, which means that both algorithms provide the same conditions for the regular operation of AdaBoost. That is, when $\varepsilon_t \leq \frac{1}{2}$ both $\alpha_t \geq 0$ and $\tilde{\alpha}_t \geq 0$. Furthermore, note that $U_t \in [0, 1]$, as defined in Step (4), in Fig. 1. From Eq. (24), it follows that, for $\tilde{\varepsilon}_t \leq \frac{1}{2}$,

$$\varepsilon_t - \tilde{\varepsilon}_t = (1 - \frac{1}{U_t})(\frac{1}{2} - \tilde{\varepsilon}_t) \leq 0 \ \Rightarrow \ \tilde{\alpha}_t = \eta_t \alpha_t \ , \quad (25)$$

where $\eta_t \in [0, 1]$. Eq. (25) asserts that under the same conditions, AdaBoost.OC takes a smaller step size than AdaBoost.ECC in the direction of h_t over the functional space $\mathcal{H}$.

We finally check Step (5) in Figs. 1 and 2, that is, the updating rules for the data-sampling distributions of AdaBoost.OC and AdaBoost.ECC. By using Eq. (20), it is straightforward to show that the updating rules of the two algorithms are the same. In conclusion, AdaBoost.OC is a shrinkage version of AdaBoost.ECC. ■

3.2. Remarks

The following remarks are immediate from Theorems 1 and 2.

(1) It is possible to reduce the computational complexity of AdaBoost.OC, by eliminating Steps (7) and (8) in Fig. 2. Instead, $\tilde{\varepsilon}_t$ can be directly calculated from Eq. (24), given ε_t. Here, the simplification stems from the fact that ε_t is easier to compute, as in Eq. (13).

(2) In [Guruswami & Sahai, 1999], the authors prove that AdaBoost.ECC has a better upper bound of the training error than AdaBoost.OC. They also experimentally observe that the training error of AdaBoost.ECC converges faster than that of AdaBoost.OC. However, the fact that the training error upper bound of one algorithm is better than that of the other cannot explain the empirical evidence related to the convergence rate. Theorem 2 provides for the strict proof of the observed phenomenon.

(3) Shrinkage can be considered as a regularization method, which has been reported to significantly improve classification performance, especially in the case of noise corrupted data [Friedman, 2001]. Introducing shrinkage in the steepest decent minimization of AdaBoost.ECC is analogous to using a specified learning rate in neural networks. Consequently, based on the above analysis in Theorem 2, one can expect that, in the low noise regime, AdaBoost.ECC may have some advantages over AdaBoost.OC. However, in the noise-corrupted-data cases, one should anticipate AdaBoost.OC to perform better than AdaBoost.ECC. This provides a guideline for selecting the appropriate algorithm between the two.

(4) Shrinkage as a regularization can be pursued explicitly in AdaBoost.ECC. We refer to the resulting new algorithm as AdaBoost.SECC. The pseudo-codes of AdaBoost.SECC and AdaBoost.ECC are identical, except for Step (8), where α_t is computed as $\alpha_t = \eta \frac{1}{4} \ln[(1 - \varepsilon_t)/\varepsilon_t]$. Here, η is a shrinkage parameter that takes values in $(0, 1]$.

4. Experiments

For the three boosting algorithms, we choose C4.5 as a base learner. C4.5 is a decision-tree classifier with a long record of successful implementation in many classification systems [Quinlan, 1993]. Although, in general, C4.5 can be employed to classify multiple classes, in our experiments, we use it as a binary classifier.

We test AdaBoost.OC, AdaBoost.ECC, and AdaBoost.SECC on five databases, four of which are publicly available at the UCI Benchmark Repository [Blake & Merz, 1998]. These are: (1) Car Evaluation Database (or shortly *Cars*), (2) Image Segmentation Database (*Images*), (3) Letter Recognition Database (*Letters*), and (4) Pen-Based Recognition of Handwritten Digits (*PenDigits*). The fifth database is USPS Dataset of Handwritten Characters (*USPS*). For *Cars* and *Letters*, all data samples are grouped in a single file per each database, while for *Images*, *PenDigits* and *USPS*, the training and test datasets are available. We divide each database into training, test, and cross-validation sets as detailed in Table 1. For databases where only a single data-file is provided, we form the training, cross-validation and test files

Table 1. The number of samples in each database

database	training	cross-validation	test
Cars	865 (50%)	286 (15%)	577 (35%)
Images	210	210 (10%)	1890 (90%)
Letters	8039 (40%)	3976 (20%)	7985 (40%)
PenDigits	5621 (75%)	1873 (25%)	3498
USPS	6931 (95%)	360 (5%)	2007

Table 2. Optimal η^* values

	noise level			
database	**0%**	**10%**	**20%**	**30%**
Cars	1	1	0.05	0.05
Images	0.5	0.05	0.05	0.05
Letters	0.2	0.05	0.05	0.05
PenDigits	1	0.5	0.5	0.2
USPS	0.5	0.35	0.05	0.05

by random selection of samples from that single file, such that a certain percentage of samples per class is present in each of the three datasets. For *Cars*, and *Letters*, we randomly choose cross-validation data from the training dataset, and for *Images*, from the test dataset. In Table 1, the number of percentages in parentheses indicates the distribution of samples per each class in the corresponding dataset. For *USPS* database, to reduce the run-time of our experiments, we projected each sample using PCA onto a lower-dimensional feature space (256 features $\rightarrow$ 54 features) at the price of 10% of the representation error.

To conduct experiments with mislabeling, noise is introduced only to the training and cross-validation sets, while the test set is kept intact. The level of noise represents a percentage of randomly selected training data (or cross-validation data) whose class labels are changed. The performance of AdaBoost.OC, AdaBoost.ECC, and AdaBoost.SECC is evaluated for several different noise levels, ranging from 0% to 30%. Throughout, for AdaBoost.SECC, the optimal shrinkage parameter η^* and the optimal number of training steps T^* are found by cross-validation. We choose η^* for which the classification error on the cross-validation data at the T^*-th step is minimum. In all experiments, the maximum number of training steps is preset to $T{=}500$. The classification error both on the test and cross-validation sets is averaged over 10 runs.

We point out that the algorithms behave similarly for the noise levels 20% and 30%. Also, results for the two ten-class datasets PenDigits and USPS are very similar. Hence, because of limited space, in Fig. 3, we show the test-error plots for only four databases at three noise levels, while the classification results are summarized in Table 3. In Fig. 4, we plot the training errors for *Letters* as typical examples of the algorithms' training-error convergence rates. The optimal η^* values for AdaBoost.SECC, determined through cross-validation, are presented in Table 2.

From the results we observe the following. First, convergence of the training error for AdaBoost.ECC is faster than for AdaBoost.OC and AdaBoost.SECC, which is in agreement with Theorem 2. Also, the convergence rate of the training error of AdaBoost.SECC is the slowest, which becomes very pronounced for high-level noise settings, where typically a small value of the shrinkage parameter η^* is used, as detailed in Table 2. Second, in

Table 3. Classification errors ($\%$) on the test data. The best results are marked in bold face. The optimal number of training steps T^* of AdaBoost.SECC, if different from the predefined number, are indicated in the parentheses.

database	noise	ECC	SECC	OC
Cars	0%	**5.5**	**5.5**	7.3
	10%	**12.5**	**12.5**	13.3
	20%	20.8	**17.0** (70)	20.0
	30%	30.2	**22.9** (90)	28.3
Images	0%	4.5	**4.2**	4.5
	10%	8.6	**7.6** (70)	8.4
	20%	15.1	**11.5** (160)	14.0
	30%	22.5	**18.2** (150)	22.9
Letters	0%	9.3	**8.0**	8.3
	10%	28.2	**13.4**	19.8
	20%	35.6	**16.9**	24.9
	30%	41.9	**23.3**	31.5
PenDigits	0%	**12.9**	**12.9**	13.9
	10%	15.2	**14.7**	14.8
	20%	17.7	**16.3**	17.1
	30%	21.7	**19.2**	19.4
USPS	0%	6.3	**6.2**	6.5
	10%	7.3	**6.6**	6.9
	20%	9.7	**7.5**	7.9
	30%	12.4	**8.7**	9.1

the absence of mislabeling noise (i.e., 0% of noise level), we observe that AdaBoost.ECC performs slightly better than AdaBoost.OC with respect to the test error. However, with the increase of noise level, AdaBoost.OC outperforms AdaBoost.ECC, as predicted in Section 3.2. Finally, regularization of AdaBoost.ECC, by introducing the shrinkage parameter η, improves the performance; however, it may also lead to overfitting (see *Cars* and *Images*), which can be alleviated by using the early stopping method. Overall, AdaBoost.SECC outperforms other two algorithms at all noise levels above zero. In some cases, significant improvements are observed. For example, for *Letters*, in a likely event, when 10% of training patterns are mislabeled, AdaBoost.SECC improves the classification performance of AdaBoost.ECC by about 50% (28% vs. 14%).

5. Conclusions

In this paper, we have unified AdaBoost.OC and AdaBoost.ECC through the margin concept. We have shown that AdaBoost.ECC performs stage-wise functional

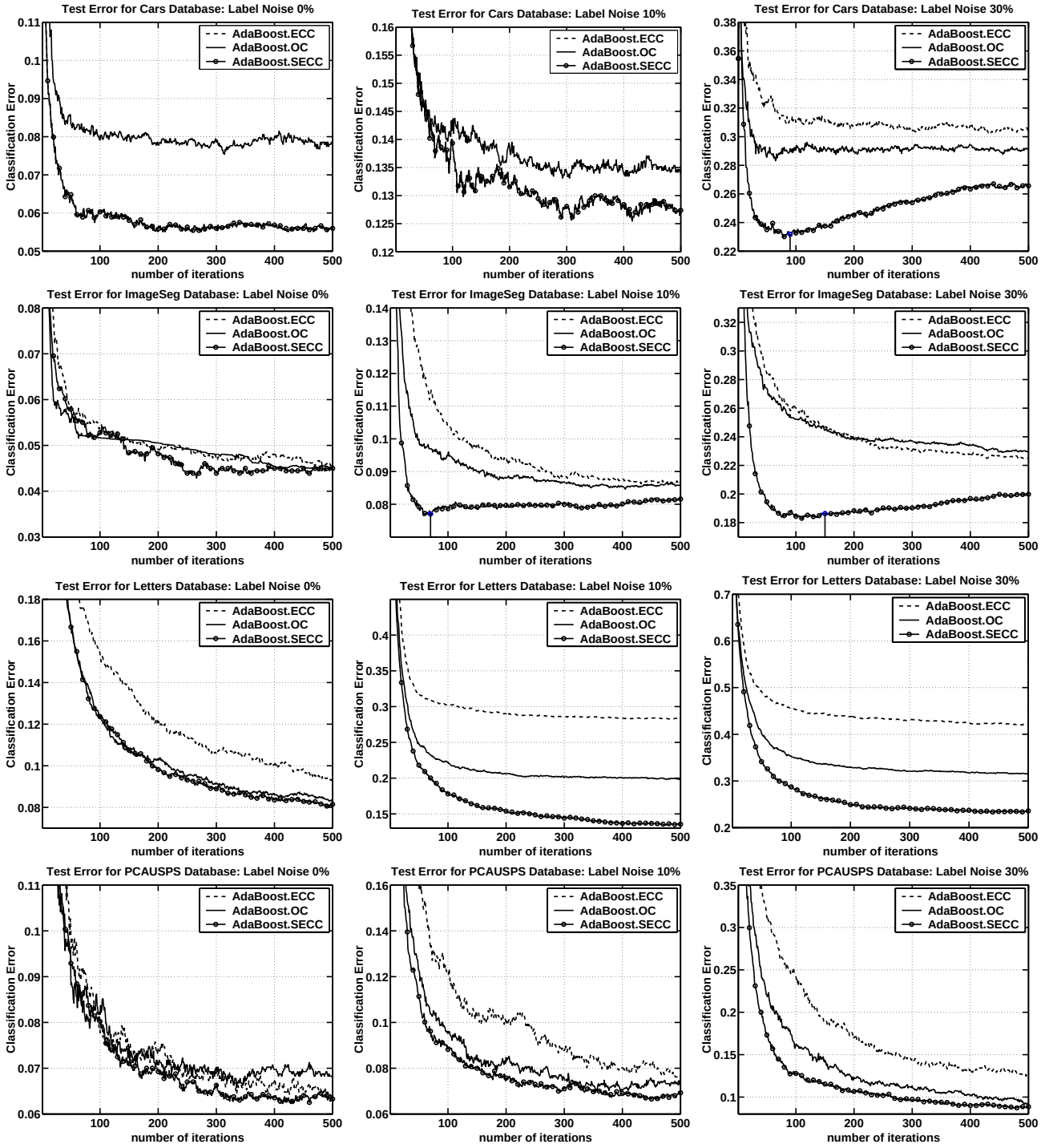

Figure 3. Classification errors on the test data. In some plots, the error curves of AdaBoost.SECC are overlapped with those of AdaBoost.ECC (i.e., $\eta = 1$, see Table 2). For AdaBoost.SECC, the optimal number of training steps T^* is found by cross-validation and indicated in the figures.

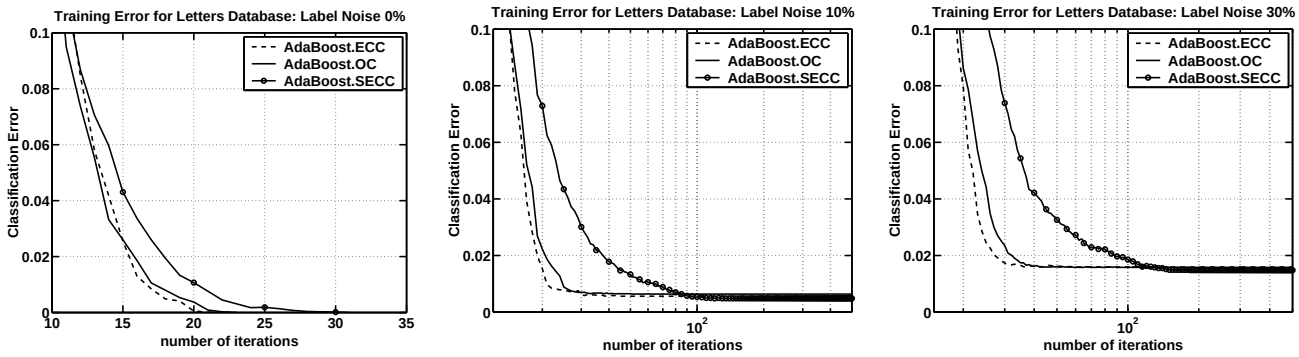

Figure 4. Letters: typical behavior of the three algorithms over all five datasets with respect to the convergence rate of the training error.

gradient descent on a cost function, defined in the domain of margin values, and that AdaBoost.OC is a shrinkage version of AdaBoost.ECC. Based on this analysis, we have formulated and explained several properties of AdaBoost.ECC and AdaBoost.OC, and derived the shrinkage version of AdaBoost.ECC, referred to as AdaBoost.SECC.

We have also reported experiments, conducted on five datasets, with training data corrupted by various levels of mislabeling noise. The empirical results confirm our theoretical findings: (1) AdaBoost.ECC is the fastest, and AdaBoost.SECC is the slowest, with respect to the convergence rate of the training error; (2) in the absence of mislabeling noise, AdaBoost.ECC yields a slightly smaller test error than AdaBoost.OC; (3) for noise levels above zero, AdaBoost.SECC performs significantly better than the second place AdaBoost.OC and the worst AdaBoost.ECC, with respect to the test error.

References

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

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, *11*, 1493–1517.

Crammer, K., & Singer, Y. (2000). On the learnability and design of output codes for multiclass problems. *Computational Learing Theory* (pp. 35–46).

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. *J. Artificial Intelligence Research*, *2*, 263–286.

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

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, *28*, 337–407.

Grove, A. J., & Schuurmans, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. *Proc. 15th Nat'l Conf. on Artificial Intelligence* (pp. 692–699). Madison, WI.

Guruswami, V., & Sahai, A. (1999). Multiclass learning, boosting, and error-correcting codes. *Proc. 12th Annual Conf. Computational Learning Theory* (pp. 145–155). Santa Cruz, California.

Mason, L., Bartlett, J., Baxter, P., & Frean, M. (2000). Functional gradient techniques for combining hypotheses. In B. Scholkopf, A. Smola, P. Bartlett and D. Schuurmans (Eds.), *Advances in Large Margin Classifiers*, 221–247. MIT Press.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rudin, C., Daubechies, I., & Schapire, R. E. (2004). The dynamics of adaboost: Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, *5*, 1557 – 1595.

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems. *Proc. 14th Int'l Conf. Machine Learning* (pp. 313–321). Nashville, TN, USA.

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, *26*, 1651–1686.