Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition Letters 28 (2007) 631–643
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Unifying multi-class AdaBoost algorithms with binary base learners under the margin framework

Yijun Sun [a,*], Sinisa Todorovic [b], Jian Li [c]

[a] Interdisciplinary Center for Biotechnology Research, P.O. Box 103622, University of Florida, Gainesville, FL 32610, USA
[b] Beckman Institute, University of Illinois at Urbana-Champaign, 405 N. Mathews Ave, Urbana, IL 61801, USA
[c] Department of Electrical and Computer Engineering, P.O. Box 116130, University of Florida, Gainesville, FL 32611, USA

Received 30 January 2006; received in revised form 21 June 2006
Available online 14 December 2006

Communicated by K. Tumer

Abstract

Multi-class AdaBoost algorithms AdaBooost.MO, -ECC and -OC have received a great attention in the literature, but their relationships have not been fully examined to date. In this paper, we present a novel interpretation of the three algorithms, by showing that MO and ECC perform stage-wise functional gradient descent on a cost function defined over margin values, and that OC is a shrinkage version of ECC. This allows us to strictly explain the properties of ECC and OC, empirically observed in prior work. Also, the outlined interpretation leads us to introduce shrinkage as regularization in MO and ECC, and thus to derive two new algorithms: SMO and SECC. Experiments on diverse databases are performed. The results demonstrate the effectiveness of the proposed algorithms and validate our theoretical findings.

Keywords: AdaBoost; Margin theory; Multi-class classification problem

1. Introduction

AdaBoost is a method for improving the accuracy of a learning algorithm (Freund and Schapire, 1997; Schapire and Singer, 1999). It repeatedly applies a base learner to the re-sampled version of training data to produce a collection of hypothesis functions, which are then combined via a weighted linear vote to form the final ensemble classifier. Under a mild assumption that the base learner consistently outperforms random guessing, AdaBoost can produce a classifier with arbitrary accurate training performance.

AdaBoost, originally designed for binary problems, can be extended to solve for multi-class problems. In one of such approaches, a multi-class problem is first decomposed into a set of binary ones, and then a binary classifier is used as the base learner. This approach is important, since the majority of well-studied classification algorithms are designed only for binary problems. It is also the main focus of this paper. Several algorithms have been proposed within the outlined framework, including AdaBoost.MO (Schapire and Singer, 1999), -OC (Schapire, 1997) and -ECC (Guruswami and Sahai, 1999). In these algorithms, a code matrix is specified such that each row of the code matrix (i.e., code word) represents a class. The code matrix in MO is specified before learning; therefore, the underlying dependence between the fixed code matrix and so-constructed binary classifiers is not explicitly accounted for, as discussed in (Allwein et al., 2000). ECC and OC seem to alleviate this problem by alternatively generating columns of a code matrix and binary hypothesis functions in a pre-defined number of iteration steps. Thereby, the underlying dependence between the code matrix and binary classifiers is exploited in a stage-wise manner.

* Corresponding author. Tel.: +1 352 273 8065; fax: +1 352 392 0044.
E-mail address: sun@dsp.ufl.edu (Y. Sun).

0167-8655/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2006.11.001

MO, ECC, and OC, as the multi-class extensions of AdaBoost, naturally inherit some of the well-known properties of AdaBoost, including a good generalization capability. Extensive theoretical and empirical studies have been reported in the literature aimed at understanding the generalization capability of the two-class AdaBoost (Dietterich, 2000; Grove and Schuurmans, 1998; Quinlan, 1996). One leading explanation is the margin theory (Schapire et al., 1998), stating that AdaBoost can effectively increase the margin, which in turn is conducive to a good generalization over unseen test data. It has been proved that AdaBoost performs a stage-wise functional gradient descent procedure on the cost function of sample margins (Mason et al., 2000; Breiman, 1999; Friedman et al., 2000). However, to the best of our knowledge neither similar margin-based analysis, nor empirical comparison of the multi-class AdaBoost algorithms has to date been reported.

Further, we observe that the behavior of ECC and OC for various experimental settings, as well as the relationship between the two algorithms are not fully examined in the literature. For example, Guruswami and Sahai (1999) claim that ECC outperforms OC, which, as we show in this paper, is not true for many settings. Also, it has been empirically observed that the training error of ECC converges faster to zero than that of OC (Guruswami and Sahai, 1999), but no mathematical explanation of this phenomenon, as of yet, has been proposed.

In this paper, we investigate the aforementioned missing links in the theoretical developments and empirical studies of MO, ECC, and OC. We show that MO and ECC perform stage-wise functional gradient descent on a cost function, defined in the domain of margin values. We further prove that OC is actually a shrinkage version of ECC. This theoretical analysis allows us to derive the following results. First, several properties of ECC and OC are formulated and proved, including (i) the relationship between their training convergence rates, and (ii) their performances in noisy regimes. Second, we show how to avoid the redundant calculation of pseudo-loss in OC, and thus to simplify the algorithm. Third, two new regularized algorithms are derived, referred to as SMO and SECC, where shrinkage as regularization is explicitly exploited in MO and ECC, respectively.

We also report experiments on the algorithms' behavior in the presence of mislabeled training data. Mislabeling noise is critical for many applications, where preparing a good training data set is a challenging task. Indeed, an erroneous human supervision in hard-to-classify cases may lead to training sets with a significant number of mislabeled data. The influence of mislabeled training data on the classification error for two-class AdaBoost was also investigated in (Dietterich, 2000). However, to our knowledge, no such study has been reported for multi-class AdaBoost yet. The experimental results support our theoretical findings. In a very likely event, when for example 10% of training patterns are mislabeled, OC outperforms ECC. Moreover, in the presence of mislabeling noise, SECC converges fastest to the smallest test error, as compared to the other algorithms.

The remainder of the paper is organized as follows. In Sections 2 and 3, we briefly review the output-coding method for solving the multi-class problem and two-class AdaBoost. Then, in Sections 4 and 5, we show that MO and ECC perform functional gradient-descent. Further, in Section 5, we prove that OC is a shrinkage version of ECC. The experimental validation of our theoretical findings is presented in Section 6. We conclude the paper with our final remarks in Section 7.

2. Output coding method

In this section, we briefly review the output coding method for solving multi-class classification problems (Allwein et al., 2000; Dietterich and Bakiri, 1995). Let $\mathscr{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \in \mathscr{X} \times \mathscr{Y}$ denote a training dataset, where $\mathscr{X}$ is the pattern space and $\mathscr{Y} = \{1, \ldots, C\}$ is the label space. To decompose the multi-class problem into several binary ones, a code matrix $\mathbf{M} \in \{\pm 1\}^{C \times L}$ is introduced, where L is the length of a code word. Here, $M(c)$ denotes the cth row, that is, a code word for class c, and $M(c,l)$ 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 L training steps, the output coding method produces a final classifier $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_L(\mathbf{x})]^{\mathrm{T}}$, where $f_l(\mathbf{x}) : \mathbf{x} \to \mathbb{R}$. 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 $\mathbf{f}(\mathbf{x})$, with respect to a specified decoding strategy. In this paper, we use the loss-based decoding strategy (Allwein et al., 2000), given by $y^* = \arg\min_{y \in \mathscr{Y}} \sum_{l=1}^{L} \exp(-M(y,l) f_l(\mathbf{x}))$.

3. AdaBoost and LP-based margin optimization

In this section, we briefly review the AdaBoost algorithm, and its relationship with margin optimization. Given a set of hypothesis functions $\mathscr{H} = \{h(\mathbf{x}) : \mathbf{x} \to \{\pm 1\}\}$, AdaBoost finds an ensemble function in the form of $F(\mathbf{x}) = \sum_t \alpha_t h_t(\mathbf{x})$ or $f(\mathbf{x}) = F(\mathbf{x}) / \sum_t \alpha_t$, where $\forall t$, $\alpha_t \geqslant 0$, by minimizing the cost function $G = \sum_{n=1}^{N} \exp(-y_n F(\mathbf{x}_n))$. The sample margin is defined as $\rho(\mathbf{x}_n) \stackrel{\Delta}{=} y_n f(\mathbf{x}_n)$, and the classifier margin, or simply margin, as $\rho \stackrel{\Delta}{=} \min_{1 \leqslant n \leqslant N} \rho(\mathbf{x}_n)$. Interested readers can refer to Freund and Schapire (1997) and Sun et al. (in press) for more detailed discussion on AdaBoost.

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 when $t \to \infty$, AdaBoost solves a linear programming (LP) problem:

$$\begin{aligned} \max \quad & \rho, \\ \text{s.t.} \quad & \rho(\mathbf{x}_n) \geqslant \rho, \quad n = 1, \ldots, N. \end{aligned} \tag{1}$$

where the margin is directly maximized. In the recent paper Rudin et al. (2004), however, the equivalence of the two algorithms has been proven to not always hold. Nevertheless, these two algorithms are closely connected in the sense that both algorithms try to maximize the margin. Throughout this paper, we will make use of this connection to define a cost function over the domain of margin values, upon which the multi-class AdaBoost.MO, ECC and OC algorithms perform stage-wise gradient descent.

4. AdaBoost.MO

In this section, we study AdaBoost.MO. In Fig. 1, we present the pseudo-code of the algorithm as proposed in (Schapire and Singer, 1999). Given a code matrix $\mathbf{M} \in \{\pm 1\}^{C\times L}$, in each iteration t, $t = 1,\ldots,T$, a distribution $\mathbf{D}_t$ is generated over pairs of training examples and columns of the matrix $\mathbf{M}$. A set of base learners $\{h_t^{(l)}(\mathbf{x})\}_{l=1}^{L}$ is then trained with respect to the distribution $\mathbf{D}_t$, based on the binary partition defined by each column of $\mathbf{M}$. The error ϵ_t of $\{h_t^{(l)}(\mathbf{x})\}_{l=1}^{L}$ is computed as Step (4) in Fig. 1 and the combination coefficient α_t is computed in Step (5) as $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$. Then, the distribution is updated as $D_{t+1}(n,l) = D_t(n,l)\exp(-\alpha_t M(y_n,l)h_t^{(l)}(\mathbf{x}_n))/Z_t$, where Z_t is a normalization constant such that $\mathbf{D}_{t+1}$ is a distribution, i.e., $\sum_{n=1}^{N}\sum_{l=1}^{L} D_{t+1}(n,l) = 1$. The process continues until the maximum iteration number T is reached. After T rounds, MO outputs a final classifier $\mathbf{F}(\mathbf{x}) = [F^{(1)}(\mathbf{x}),\ldots,F^{(L)}(\mathbf{x})]^{\mathrm{T}}$ or $\mathbf{f}(\mathbf{x}) = \mathbf{F}(\mathbf{x})/\sum_{t=1}^{T}\alpha_t$, where $F^{(l)}(\mathbf{x}) = \sum_{t=1}^{T}\alpha_t h_t^{(l)}(\mathbf{x})$. When presented with an unseen sample $\mathbf{x}$, MO predicts label y^*, whose row $M(y^*)$ is the "closest" to $\mathbf{F}(\mathbf{x})$, with respect to a specified decoding strategy.

Now, we seek to define the cost function upon which MO performs the optimization. Given a fixed code matrix, one natural idea is to find $\mathbf{F}(\mathbf{x})$ such that the minimum value of the sample margin of each bit position is maximized:

$$\begin{aligned}&\max \quad \rho\\ &\text{s.t.} \quad \rho^{(l)}(\mathbf{x}_n) = M(y_n,l)\frac{\sum_t \alpha_t h_t^{(l)}(\mathbf{x}_n)}{\sum_t \alpha_t} \geqslant \rho, \qquad (2)\\ &\qquad n = 1,\ldots,N,\ \ l = 1,\ldots,L.\end{aligned}$$

AdaBoost.MO

(1) **Initialization**: given $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \in \mathcal{X}\times\mathcal{Y}$, initialize $D_1(n,l) = 1/NL$, $n = 1{:}N$, $l = 1{:}L$; set the maximum number of iteration steps T.
(2) **for** $t = 1:T$
(3) Train base learner with respect to distribution $\mathbf{D}_t$ and compute $\{h_t^{(l)}(\mathbf{x})\}_{l=1}^{L}$;
(4) Compute: $\epsilon_t = \sum_{n=1}^{N}\sum_{l=1}^{L} D_t(n,l)\mathbf{I}(M(y_n,l) \neq h_t^{(l)}(\mathbf{x}_n))$;
(5) Compute: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;
(6) Update weights $\mathbf{D}_{t+1}$ as $D_{t+1}(n,l) = D_t(n,l)\exp(-\alpha_t M(y_n,l)h_t^{(l)}(\mathbf{x}_n))/Z_t$;
(7) **end**
(8) **Output**: $\mathbf{F}(\mathbf{x}) = [F^{(1)}(\mathbf{x}),\cdots,F^{(L)}(\mathbf{x})]^{\mathrm{T}}$.

Fig. 1. Pseudo-code of AdaBoost.MO, as proposed in (Schapire and Singer, 1999).

In the light of the connection between AdaBoost and LP, discussed in Section 3, the cost function that MO optimizes can be specified as

$$\begin{aligned}G &= \sum_{n=1}^{N}\sum_{l=1}^{L}\exp\left(-\rho^{(l)}(\mathbf{x}_n)\sum_t \alpha_t\right)\\ &= \sum_{n=1}^{N}\sum_{l=1}^{L}\exp(-F^{(l)}(\mathbf{x}_n)M(y_n,l)), \qquad (3)\end{aligned}$$

which is indeed proved in the following theorem.

Theorem 1. *AdaBoost.MO performs a stage-wise functional gradient descent procedure on the cost function given by Eq. (3).*

Proof. The proof is provided in the Appendix. □

5. AdaBoost.ECC and AdaBoost.OC

In this section, we investigate ECC (Schapire, 1997) and OC (Guruswami and Sahai, 1999), whose pseudo-codes are given in Figs. 2 and 3, respectively. Our goal is to unify both algorithms under the framework of margin theory, and to establish the relationship between the two algorithms.

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

$$\rho(\mathbf{x}_n) \stackrel{\Delta}{=} \min_{\{c\in C, c\neq y_n\}} \Delta(M(c),\mathbf{f}(\mathbf{x}_n)) - \Delta(M(y_n),\mathbf{f}(\mathbf{x}_n)), \qquad (4)$$

where $\Delta(\cdot)$ is a distance measure. Maximization of $\rho(\mathbf{x}_n)$ in Eq. (4) 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. By specifying $\Delta(M(c),\mathbf{f}(\mathbf{x})) \stackrel{\Delta}{=} \|M(c) - \mathbf{f}^{\mathrm{T}}(\mathbf{x})\|^2$, from Eq. (4) we derive

$$\rho(\mathbf{x}_n) = 2M(y_n)\mathbf{f}(\mathbf{x}_n) - \max_{\{c\in C, c\neq y_n\}}\{2M(c)\mathbf{f}(\mathbf{x}_n)\}. \qquad (5)$$

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

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 $\mathbf{M}$: $M_t \in \{-1,+1\}^{C\times 1}$;
(4) Compute: $U_t = \sum_{n=1}^{N}\sum_{c=1}^{C} D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;
(5) Compute: $d_t(n) = \frac{1}{U_t}\sum_{c=1}^{C} D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;
(6) Train the base learner with respect to distribution $\mathbf{d}_t$, and compute $h_t(\mathbf{x})$;
(7) Compute: $\varepsilon_t = \sum_{n=1}^{N}\mathbf{I}(M(y_n,t) \neq h(\mathbf{x}_n))d_t(n)$;
(8) Compute: $\alpha_t = \frac{1}{4}\ln[(1-\varepsilon_t)/\varepsilon_t]$;
(9) Update weights $\mathbf{D}_{t+1}$ as $D_{t+1}(n,c) = \frac{1}{Z_t}D_t(n,c)\exp(-\alpha_t(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n))$;
(10) **end**
(11) **Output**: $\mathbf{F}(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}),\cdots,\alpha_T h_T(\mathbf{x})]^{\mathrm{T}}$.

Fig. 2. Pseudo-code of AdaBoost.ECC, as proposed in (Guruswami and Sahai, 1999).

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 $\mathbf{M}$: $M_t \in \{-1,+1\}^{C\times 1}$;
(4) Compute: $U_t = \sum_{n=1}^N \sum_{c=1}^C D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;
(5) Compute: $d_t(n) = \frac{1}{U_t}\sum_{c=1}^C D_t(n,c)\mathbf{I}(M(y_n,t) \neq M(c,t))$;
(6) Train the base learner 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:
$\tilde{\varepsilon}_t = \frac{1}{2}\sum_{n=1}^N \sum_{c=1}^C D_t(n,c)\left(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n))\right)$;
(9) Compute: $\tilde{\alpha}_t = \frac{1}{4}\ln[(1-\tilde{\varepsilon}_t)/\tilde{\varepsilon}_t]$;
(10) Update weights $\mathbf{D}_{t+1}$ as
$D_{t+1}(n,c) = \frac{1}{Z_t} D_t(n,c)\exp\left(2\tilde{\alpha}_t(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)))\right)$;
(11) **end**
(12) **Output**: $\mathbf{F}(\mathbf{x}) = [\tilde{\alpha}_1 h_1(\mathbf{x}), \cdots, \tilde{\alpha}_T h_T(\mathbf{x})]^{\mathrm{T}}$.

Fig. 3. Pseudo-code of AdaBoost.OC, as proposed in (Schapire, 1997).

$$\begin{aligned} &\max \quad \rho, \\ &\text{s.t.} \quad M(y_n)\mathbf{f}(\mathbf{x}_n) - \max_{\{c\in C, c\neq y_n\}} \{M(c)\mathbf{f}(\mathbf{x}_n)\} \geqslant \rho, \quad n = 1,\ldots,N. \end{aligned} \tag{6}$$

We are particularly interested in finding $\mathbf{f}(\mathbf{x})$ of the form $\mathbf{f}(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}),\ldots,\alpha_T h_T(\mathbf{x})]^{\mathrm{T}}$, where $\sum_{t=1}^T \alpha_t = 1, \alpha_t \geqslant 0$. From Eq. (6), we derive

$$\begin{aligned} &\max \quad \rho \\ &\text{s.t.} \quad \sum_{t=1}^T \alpha_t (M(y_n,t) - M(c,t))h_t(\mathbf{x}_n) \geqslant \rho \\ &\qquad n = 1,\ldots,N,\ c = 1,\ldots,C,\ c \neq y_n, \\ &\qquad \sum_{t=1}^T \alpha_t = 1,\ \boldsymbol{\alpha} \geqslant 0. \end{aligned} \tag{7}$$

From the discussion on LP and AdaBoost in Section 3, it appears reasonable to define a new cost function, optimized by a multi-class AdaBoost algorithm:

$$\begin{aligned} G &\triangleq \sum_{n=1}^N \sum_{\{c=1,c\neq y_n\}}^C \exp(-(M(y_n) - M(c))\mathbf{F}(\mathbf{x}_n)) \\ &= \sum_{n=1}^N \sum_{\{c=1,c\neq y_n\}}^C \exp\left(-\sum_{t=1}^T \alpha_t (M(y_n,t) - M(c,t))h_t(\mathbf{x}_n)\right). \end{aligned} \tag{8}$$

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

Theorem 2. *AdaBoost.ECC performs a stage-wise functional gradient descent procedure on the cost function given by Eq. (8).*

Proof. Fig. 2 presents the pseudo-code of ECC (Guruswami and 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 minimization of G in Eq. (8), we prove the theorem. For the moment, we assume that the code matrix $\mathbf{M}$ is given, the generation of which is discussed later. After $(t-1)$ iteration steps, ECC produces $\mathbf{F}_{t-1}(\mathbf{x}_n) = [\alpha_1 h_1(\mathbf{x}_n),\ldots, \alpha_{t-1}h_{t-1}(\mathbf{x}_n), 0,\ldots,0]^{\mathrm{T}}$. In the tth iteration step, the goal of the algorithm is to compute the tth entry of $\mathbf{F}_t(\mathbf{x}_n) = [\alpha_1 h_1(\mathbf{x}_n),\ldots,\alpha_{t-1}h_{t-1}(\mathbf{x}_n), \alpha_t h_t(\mathbf{x}_n), 0,\ldots,0]^{\mathrm{T}}$.

From Eq. (8), the negative functional derivative of G with respect to $\mathbf{F}_{t-1}(\mathbf{x})$, at $\mathbf{x} = \mathbf{x}_n$, is computed as $-\nabla_{\mathbf{F}_{t-1}} G|_{\mathbf{x}=\mathbf{x}_n} = \sum_{\{c=1,c\neq y_n\}}^C (M(y_n) - M(c))G_{t-1}(n)$, where $G_{t-1}(n,c) \triangleq \exp(-(M(y_n) - M(c))\mathbf{F}_{t-1}(\mathbf{x}_n))$. To find the optimal hypothesis function h_t in the next training step, we resort to Friedman's optimization approach (Friedman, 2001). It is straightforward to show that h_t should be selected from $\mathscr{H}$, by maximizing its correlation with the tth component of $-\nabla_{\mathbf{F}_{t-1}} G$ as $h_t = \arg\max_{h\in\mathscr{H}} \sum_{n=1}^N \sum_{\{c=1,c\neq y_n\}}^C h(\mathbf{x}_n)(M(y_n,t) - M(c,t))G_{t-1}(n,c)$. To facilitate the computation of h_t, we introduce the following terms:

$$\begin{aligned} V_t(n) &\triangleq \sum_{\{c=1,c\neq y_n\}}^C |M(y_n,t) - M(c,t)|G_{t-1}(n,c), \\ V_t &\triangleq \sum_{n=1}^N V_t(n), \\ d_t(n) &\triangleq V_t(n)/V_t. \end{aligned} \tag{9}$$

Note that V_t differs from U_t, defined in Step (4) in Fig. 2, by a constant $2N(C-1)\prod_{i=1}^{t-1} Z_i$, where Z_i is a normalization constant defined in Step (9) in Fig. 2. 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

$$h_t = \arg\max_{h\in\mathscr{H}} \sum_{n=1}^N V_t(n)\operatorname{sign}(M(y_n,t))h(\mathbf{x}_n), \tag{10}$$

$$= \arg\max_{h\in\mathscr{H}} V_t \sum_{n=1}^N d_t(n)M(y_n,t)h(\mathbf{x}_n), \tag{11}$$

$$= \arg\max_{h\in\mathscr{H}} U_t \sum_{n=1}^N d_t(n)M(y_n,t)h(\mathbf{x}_n), \tag{12}$$

$$= \arg\min_{h\in\mathscr{H}} \sum_{n=1}^N \mathbf{I}(M(y_n,t) \neq h(\mathbf{x}_n))d_t(n), \tag{13}$$

$$= \arg\min_{h\in\mathscr{H}} \varepsilon, \tag{14}$$

where ε is the weighted error of h. Once h_t is found, α_t can be computed by a line search as

$$\begin{aligned} \alpha_t &= \arg\min_{\alpha\geqslant 0} G_t \\ &= \arg\min_{\alpha\geqslant 0} \sum_{n=1}^N \sum_{\substack{c=1 \\ c\neq y_n}}^C G_{t-1}(n,c)\exp[-\alpha(M(y_n,t) \\ &\quad - M(c,t))h_t(\mathbf{x}_n)]. \end{aligned} \tag{15}$$

In our case, where the hypotheses h_t are specified as binary classifiers, α_t can be solved analytically. Taking the derivative of G_t with respect to α_t gives

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

where we used the definitions from Eq. (9), and the fact that $(M(y_n,t) - M(c,t))$ takes values in the set $\{0, 2, -2\}$, and that $(M(y_n,t) - M(c,t))$ has the same sign as $M(y_n,t)$. From Eqs. (13) and (14), and $\partial G_t/\partial \alpha_t = 0$, we obtain $\alpha_t = \frac{1}{4}\ln[(1-\varepsilon_t)/\varepsilon_t]$, which is equal to the expression given in Step (8), Fig. 2.

Finally we check the update rule for data distribution $\mathbf{d}_t$. By unravelling $D_t(n,c)$ in Step (9) of the pseudo-code of ECC (see Fig. 2), we derive $D_t(n,c) = G_{t-1}(n,c)/N(C-1)\prod_{i=1}^{t-1} Z_i$, for $c \neq y_n$, and $D_t(n,c) = 0$, for $c = y_n$. By plugging this result into Steps (4) and (5) of the pseudo-code of ECC, it is straightforward to show that the expressions for $\mathbf{d}^{(t)}$ in Step (5) and Eq. (9) are the same. This completes the proof.

Now, we discuss how to generate the columns of the code matrix, denoted as $\mathbf{M}_{\cdot t}$. Simultaneous optimization of both $\mathbf{M}_{\cdot t}$ and h_t is known be an NP-hard problem (Crammer and Singer, 2000). In both OC and ECC, this problem is alleviated by conducting a two-stage optimization. That is, $\mathbf{M}_{\cdot t}$ is first generated by maximizing U_t, given in Step (4) in Fig. 2, and then h_t is trained based on the binary partition defined by $\mathbf{M}_{\cdot t}$. In (Schapire, 1997; Guruswami and 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. 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 2 provides for yet another interpretation of the outlined procedure. Ideally, in the tth iteration step we want to find $\mathbf{M}_{\cdot t}$ and

Table 1
The number of samples in each database

Database	Training	Cross-validation	Test
Cars	865 (50%)	286 (15%)	577 (35%)
Images	210 (9%)	210 (9%)	1890 (82%)
Letters	8039 (40%)	3976 (20%)	7985 (40%)
PenDigits	5621 (75%)	1873 (25%)	3498
USPS	6931 (95%)	360 (5%)	2007

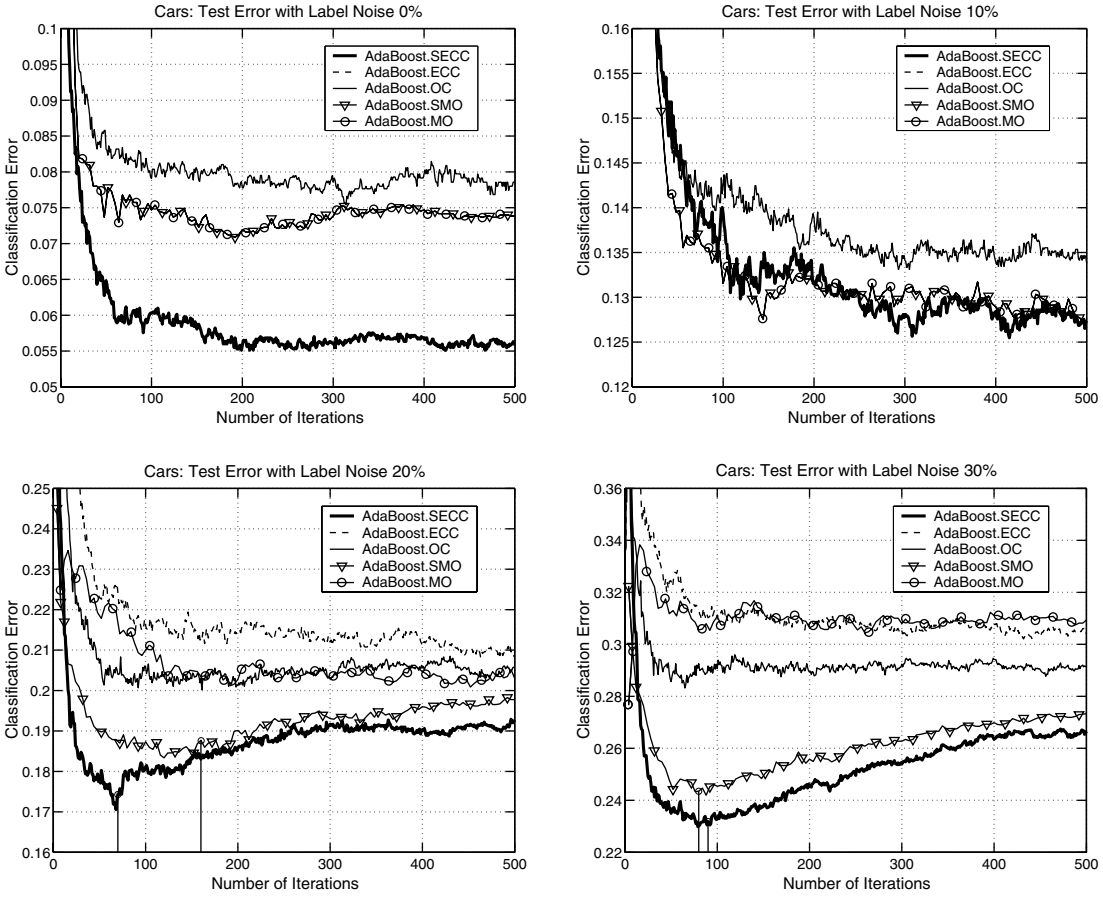

Fig. 4. Classification errors on the test data of *Cars*. In some plots, the error curves of SECC and SMO overlap those of ECC and MO (i.e., $\eta = 1$), respectively. For SECC and SMO, the optimal number of training steps T^* is found by cross-validation and indicated in the figures.

h_t simultaneously to maximize their correlation, 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. □

5.1. Relationship between AdaBoost.OC and AdaBoost.ECC

ECC is derived from OC on the algorithm level, as discussed in (Guruswami and Sahai, 1999). However, their relationship is not fully and strictly examined in the literature, which is addressed in the following theorem.

Theorem 3. *AdaBoost.OC is a shrinkage version of AdaBoost.ECC.*

Proof. We compare the expressions for computing the weights, and for updating the data-sampling distribution, to establish the relationship between OC and ECC. Let us first take a look at the pseudo-code of OC given in Fig. 3. In Step (7), a pseudo-hypothesis function is constructed as $\tilde{h}_t(\mathbf{x}) = \{c \in \mathscr{Y} : h_t(\mathbf{x}) = M(c,t)\}$. Using $\tilde{h}_t(\mathbf{x})$, a pseudo-loss, $\tilde{\varepsilon}_t$, is computed in Step (8) as $\tilde{\varepsilon}_t = \frac{1}{2}\sum_{n=1}^{N}\sum_{c=1}^{C} D_t(n,c)(\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)))$, where $D_t(n,c)$ is updated in Step (9). Note that:

$$\mathbf{I}(y_n \notin \tilde{h}_t(\mathbf{x}_n)) + \mathbf{I}(c \in \tilde{h}_t(\mathbf{x}_n)) = \frac{1}{2}(M(c,t) - M(y_n,t))h_t(\mathbf{x}_n) + 1. \quad (17)$$

It follows that

$$\tilde{\varepsilon}_t = \frac{1}{4}\underbrace{\sum_{n=1}^{N}\sum_{c=1}^{C} D_t(n,c)(M(c,t) - M(y_n,t))h_t(\mathbf{x}_n)}_{r} + \frac{1}{2} = \frac{1}{4}r + \frac{1}{2} \Rightarrow r = 4\left(\tilde{\varepsilon}_t - \frac{1}{2}\right). \quad (18)$$

Now, let us take a look at the pseudo-code of ECC given in Fig. 2. The training error, ε_t, is computed in Step (7) as

$$\varepsilon_t = \sum_{n=1}^{N} \mathbf{I}(M(y_n,t) \neq h_t(\mathbf{x}_n))d_t(n) = \frac{1}{2} - \frac{1}{2}\sum_{n=1}^{N} d_t(n)M(y_n,t)h_t(\mathbf{x}_n). \quad (19)$$

From Step (5) in Fig. 2 and the fact that $\mathbf{I}(M(y_n,t) \neq M(c,t))M(y_n,t) = (M(y_n,t) - M(c,t))/2$, we have

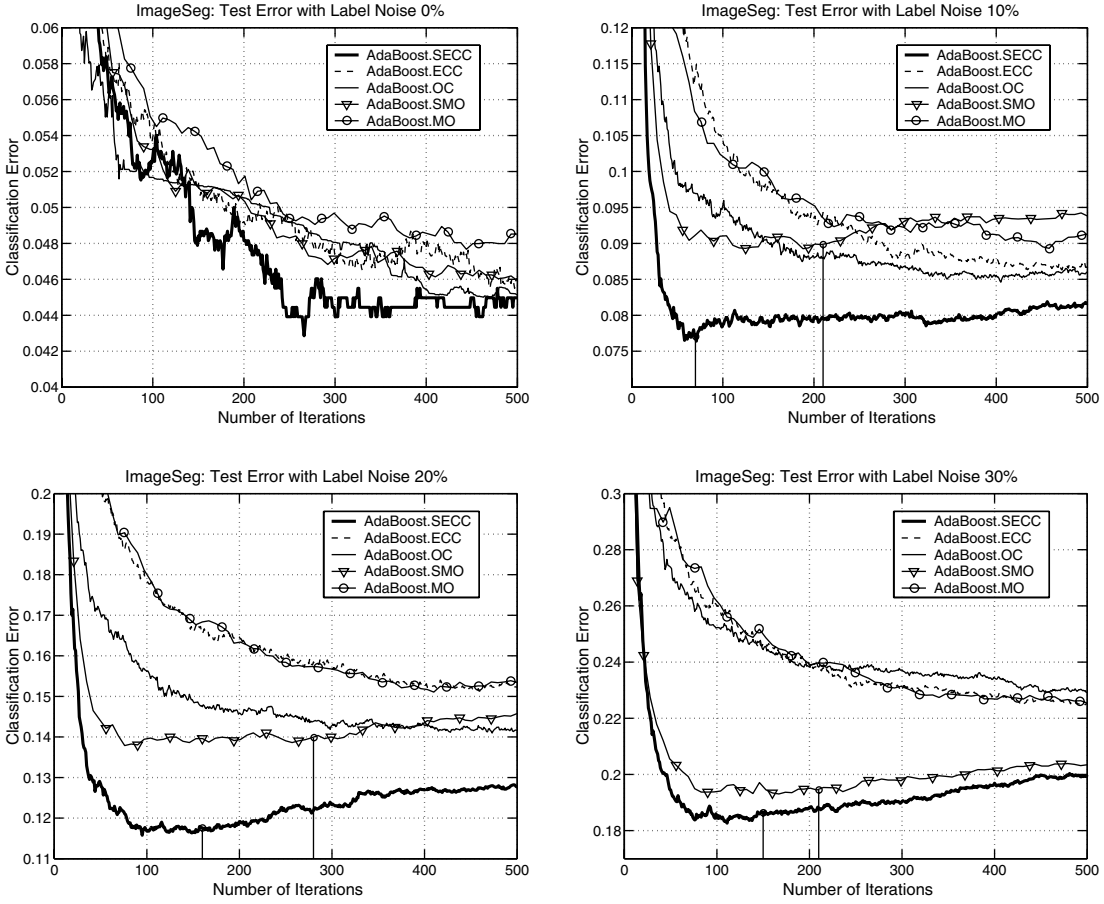

Fig. 5. Classification errors on the test data of *Images*.

$$\varepsilon_t = \frac{1}{2} - \frac{1}{4U_t}\sum_{n=1}^{N}\sum_{c=1}^{C} D_t(n,c)(M(y_n,t) - M(c,t))h_t(\mathbf{x}_n) = \frac{1}{2} + \frac{1}{4U_t}r. \quad (20)$$

By plugging Eq. (18) into Eq. (20), we get:

$$\varepsilon_t = \frac{1}{2} + \frac{1}{U_t}\left(\tilde{\varepsilon}_t - \frac{1}{2}\right). \quad (21)$$

From Eq. (21), we observe that $\varepsilon_t \leqslant \frac{1}{2}$ if and only if $\tilde{\varepsilon}_t \leqslant \frac{1}{2}$, which means that both algorithms provide the same conditions for the regular operation of AdaBoost. That is, when $\varepsilon_t \leqslant \frac{1}{2}$ both $\alpha_t \geqslant 0$ and $\tilde{\alpha}_t \geqslant 0$. Furthermore, note that $U_t \in [0,1]$, as defined in Step (4) in Fig. 2. From Eq. (21), we have that, for $\tilde{\varepsilon}_t \leqslant \frac{1}{2}$,

$$\varepsilon_t - \tilde{\varepsilon}_t = \left(1 - \frac{1}{U_t}\right)\left(\frac{1}{2} - \tilde{\varepsilon}_t\right) \leqslant 0 \Rightarrow \varepsilon_t \leqslant \tilde{\varepsilon}_t. \quad (22)$$

Finally, from Eq. (22), and Steps (8) and (9) in Figs. 2 and 3, respectively, we derive

$$\tilde{\alpha}_t = \eta_t \alpha_t, \quad (23)$$

where $\eta_t \in [0,1]$. Eq. (23) asserts that in each iteration step t, AdaBoost.OC takes a smaller step size than AdaBoost.ECC in the search for h_t over the functional space $\mathscr{H}$.

Finally, by using Eq. (17), it is straightforward to show that the updating rules of the data-sampling distributions of the two algorithms, i.e., Step (5) in Figs. 2 and 3, are the same. In conclusion, AdaBoost.OC is a shrinkage version of AdaBoost.ECC. □

5.2. Discussion

The following remarks are immediate from Theorems 1–3:

- It is possible to reduce the computational complexity of OC, by eliminating Steps (7) and (8) in Fig. 3. Instead, $\tilde{\varepsilon}_t$ can be directly calculated from Eq. (21), given ε_t. Here, the simplification stems from the fact that ε_t is easier to compute, as in Eq. (13).
- In (Guruswami and Sahai, 1999), the authors prove that ECC has a better upper bound of the training error than OC. They also experimentally observe that the training error of ECC converges faster than that of OC. However, the fact that the training-error upper bound of one algorithm is better than that of the other cannot

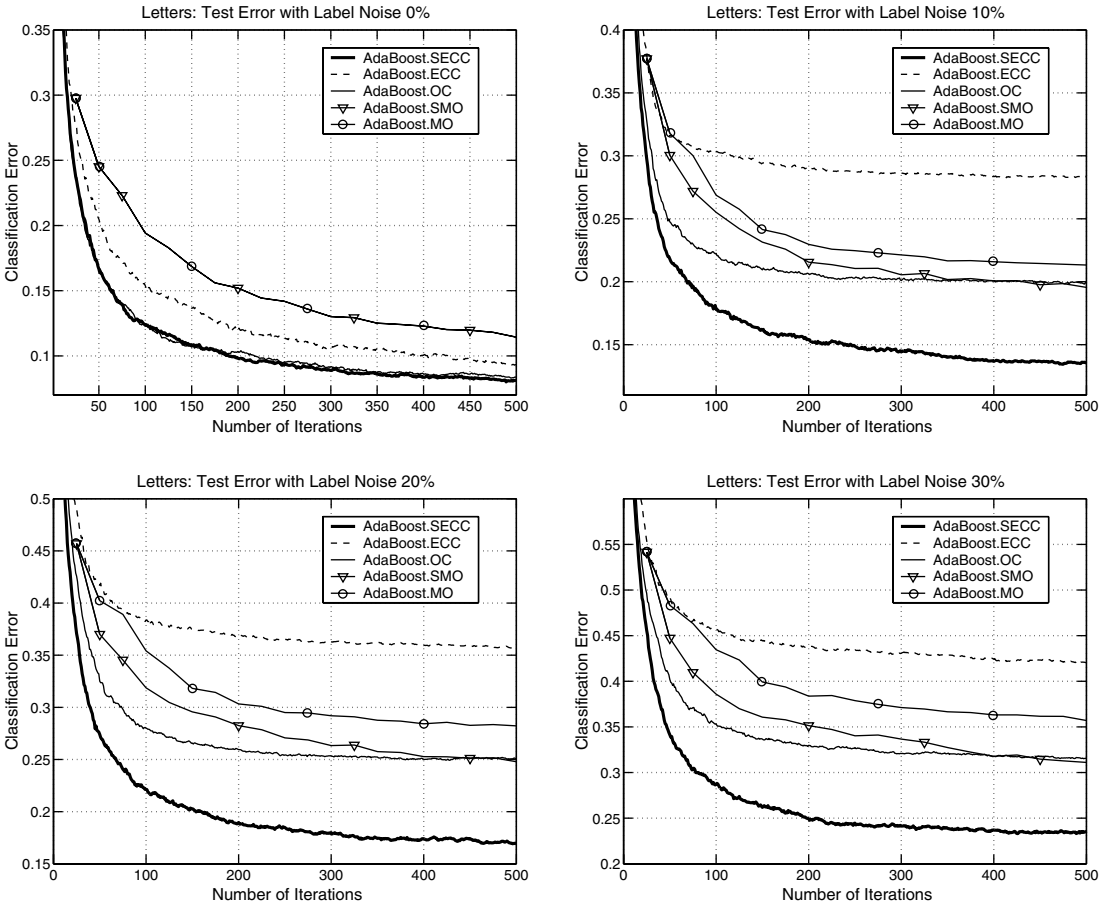

Fig. 6. Classification errors on the test data of *Letters*.

explain the empirical evidence related to the convergence rate. Theorem 3 provides for the strict proof of the observed phenomenon.

- Shrinkage can be considered a regularization method, which has been reported to significantly improve classification performance in the case of noise corrupted data (Friedman, 2001). Introducing shrinkage in the steepest decent minimization of ECC is analogous to specifying a learning rate in neural networks. Consequently, based on the analysis in Theorem 3, one can expect that in noise-corrupted-data cases, OC should perform better than ECC. This provides a guideline for selecting the appropriate algorithm between the two.
- Theorems 1 and 2 give rise to a host of novel algorithms that can be constructed by introducing the shrinkage term, $\eta \in (0,1]$, into the original multi-class algorithms. Thereby, we design SMO and SECC, respectively. The pseudo-codes of SMO and MO are identical, except for Step (5), where α_t should be computed as $\alpha_t = \eta\frac{1}{2}\ln[(1-\varepsilon_t)/\varepsilon_t]$. Similarly, the pseudo-codes of SECC and ECC are identical, except for Step (8), where α_t should be computed as $\alpha_t = \eta\frac{1}{4}\ln[(1-\varepsilon_t)/\varepsilon_t]$.
- From Theorems 1–3, it follows that MO, SMO, ECC, SECC and OC perform stage-wise functional gradient descent on a cost function expressed in terms of margin values, i.e., that they increase the classifier margin over a given number of training iterations. Therefore, the margin-based theoretical analysis of two-class AdaBoost by Schapire et al. (1998) can be directly applied to the multi-class AdaBoost to explain their good generalization capability. This property is also demonstrated experimentally in the following section.

In the following section, we present experiments with MO, SMO, ECC, SECC, and OC, which support our theoretical findings.

6. Experiments

In our experiments, 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.MO, SMO, ECC, SECC, and OC on five databases, four of which are publicly available at the UCI Benchmark Repository (Blake and Merz, 1998). These are (1) Car Evaluation Database (or short *Cars*), (2) Image Segmentation Database (*Images*), (3) Letter Rec-

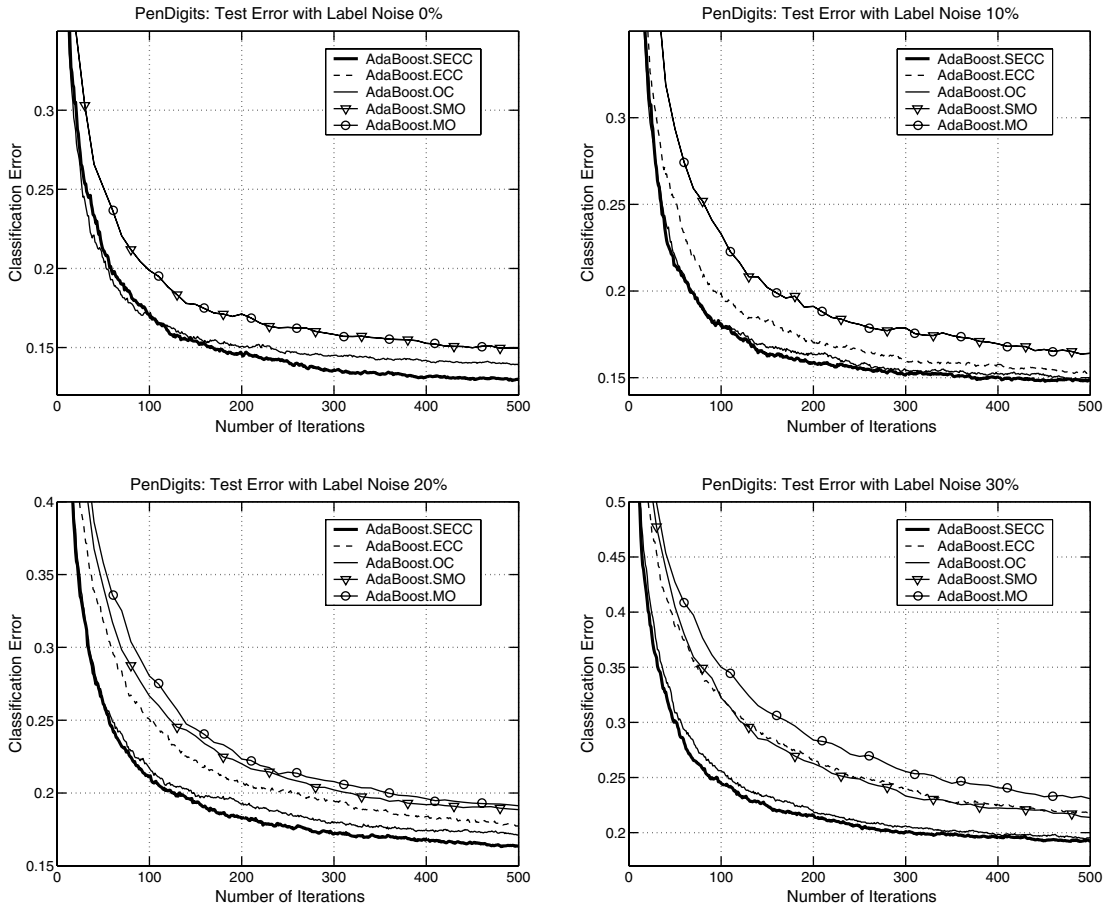

Fig. 7. Classification errors on the test data of *PenDigits*.

ognition Database (*Letters*), and (4) Pen-Based Recognition of Handwritten Digits (*PenDigits*). The fifth database is USPS Dataset of Handwritten Characters (*USPS*) (LeCun et al., 1989). We divide each database into training, test, and validation sets. For *Cars*, *Images*, and *Letters*, all available data samples are grouped in a single file per each database. Here, training, test, and validation sets are formed 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, as detailed in Table 1. For *PenDigits* and *USPS*, the training and test datasets are already given. Here, the test datasets are kept intact, while new training and validation sets are formed from the original training data, as reported in Table 1. For *USPS* database, to reduce the run-time of our experiments, we projected each sample using principal component analysis onto a lower-dimensional feature space (256 features → 54 features) at the price of 10% of the representation error.

To conduct experiments with mislabeling, noise is introduced only to the training and validation sets, while the test set is kept intact. The level of noise represents a percentage of randomly selected training data (or validation data), whose class labels are changed. The performance of the five algorithms is evaluated for several different noise levels, ranging from 0% to 30%.

For MO and SMO, we choose "*one against all*" coding method, because of its simplicity, considering that there is no decisive answer as to which output code is the best (Allwein et al., 2000). Throughout, for SMO and SECC, the optimal shrinkage parameter η^*, and the optimal number of training steps T^* are found by cross-validation. We find T^* by sliding in 10-step increments a 20-step-wide averaging window over the test-error results on validation data. The sliding of the window is stopped at the training step $T^* = t$, when the increase in the average test error over the previously recorded value in step $(t - 10)$ is detected. Further, we choose η^* for which the classification error on the validation data at the T^*th step is minimum.

In all experiments, the maximum number of training steps is preset to $T = 500$ for ECC, SECC, and OC. As typically done in many approaches (Schapire and Singer, 1999; Schapire, 1997), we preset the number of training steps to obtain a reasonable comparison of the performance of the algorithms balanced against the required processing time. Although the algorithms may not reach the minimum possible test error in the specified number of iteration steps, this minimum error becomes irrelevant in real applications

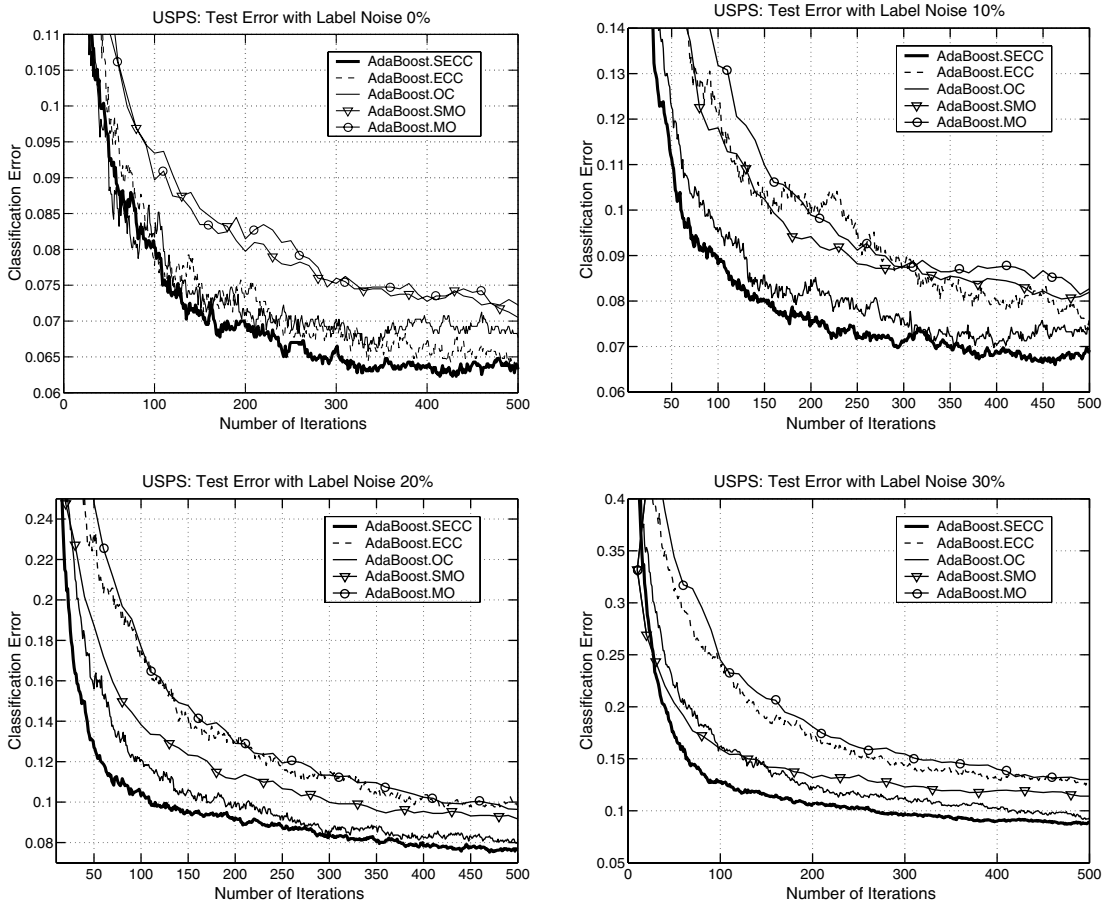

Fig. 8. Classification errors on the test data of *USPS*.

when the processing time exceeds several hours on 2.4GHZ 1GB RAM PC, which is the case when $T = 500$. To obtain comparable results (in terms of the computation) for MO and SMO, where we use "one against all" coding strategy, the maximum number of training steps is computed as $T = 500/(\#\text{ of classes})$, which is different for each database. Thus, for *Cars* $T = 125$, for *Images* $T = 72$, for *Letters* $T = 20$, for *PenDigits* $T = 50$, and for *USPS* $T = 50$. Note that the outlined numbers of iteration steps take approximately the same processing time as $T = 500$ for ECC, SECC, and OC. The classification error both on the test and validation sets is averaged over 10 runs for each database.

Figs. 4–8 show classification error on the test data. In Figs. 9 and 10 we plot the training errors for *Images* and *Letters* as two typical examples of the algorithms' training-error convergence rates. The optimal η^* values for SECC and SMO, are presented in Table 2. In Table 3, we also list the classification error on the test data at the optimal step T^*, indicated in the parentheses if different from the maximum preset number of iteration steps T.

Fig. 11 illustrates the classifier margin on *Letters* over a range of mislabeling noise levels for the five algorithms. These results experimentally validate Theorems 1–3 that the algorithms increase the margin as the number of iterations becomes larger, which is conducive to a good generalization capability.

From the results we observe the following. First, the training error of ECC converges faster to zero than that of OC and SECC, which is in agreement with Theorem 3. Also, the training convergence rate of 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 the absence of mislabeling noise, we observe that ECC performs similarly to OC with respect to the test error. However, with the increase of noise level, OC outperforms ECC, as predicted in Section 5.2. Moreover, SECC performs better than both OC and ECC at all noise levels above zero. For example, for *Letters*, in a likely event, when 10% of training patterns are mislabeled, SECC improves the classification performance of ECC by about 50% (28.2% vs. 13.4%).

Third, regularization of MO and ECC, by introducing the shrinkage parameter η, improves their performance; however, it may also lead to overfitting (see Figs. 4 and 5). It is possible to estimate the optimal number of training steps to avoid overfitting through cross-validation. Overall,

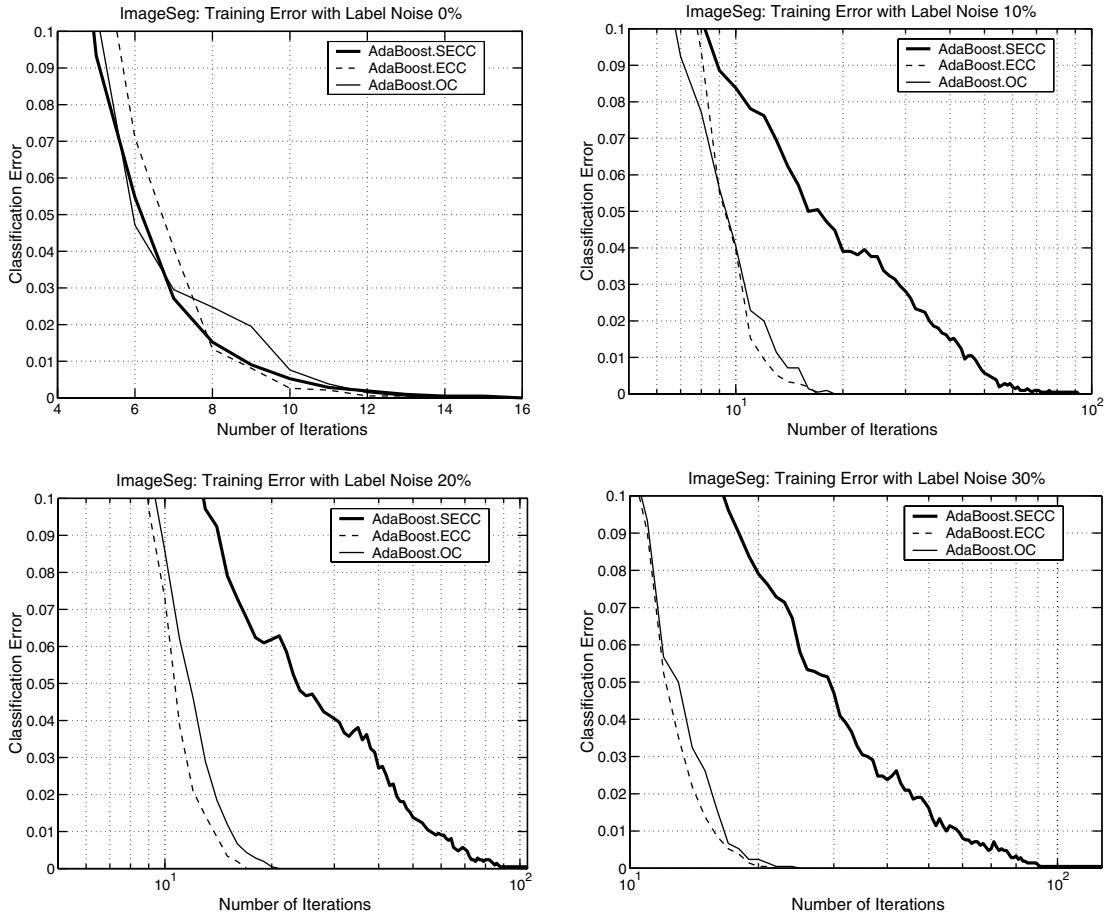

Fig. 9. Classification errors on the training data of *Images*.

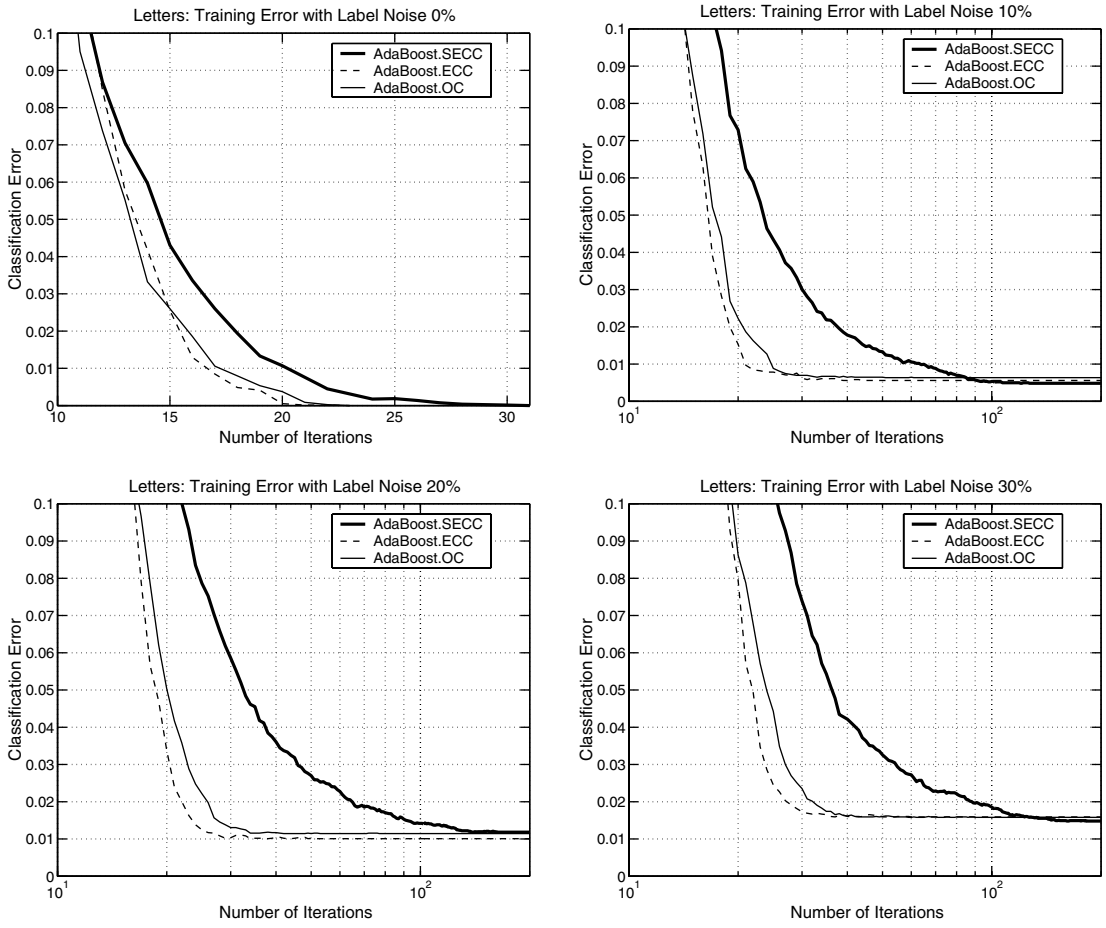

Fig. 10. Classification error on the training data of *Letters*.

Table 2
Optimal η^* values

Database	AdaBoost.SECC Noise level				AdaBoost.SMO Noise level			
	0%	10%	20%	30%	0%	10%	20%	30%
Cars	1	1	0.05	0.05	1	1	0.05	0.05
Images	0.5	0.05	0.05	0.05	0.8	0.05	0.05	0.05
Letters	0.2	0.05	0.05	0.05	1	0.5	0.35	0.2
PenDigits	1	0.5	0.5	0.2	1	1	0.8	0.8
USPS	0.5	0.35	0.05	0.05	0.65	0.65	0.65	0.05

SECC outperforms other four algorithms with significant improvements for all databases and all noise levels.

7. Conclusion

In this paper, we have unified AdaBoost.MO, ECC, and OC under the framework of margin theory. We have shown that MO and ECC perform stage-wise functional gradient descent on a cost function defined over margin values, and that OC is a shrinkage version of ECC. Based on these analyses, we have formulated and explained

Table 3
Classification errors (%) on the test data

Database	Noise (%)	ECC	SECC	MO	SMO	OC
Letters	0	9.3	**8.0**	9.5	9.5	8.3
	10	28.2	**13.4**	20.6	18.5	19.8
	20	35.6	**16.9**	27.2	23.4	24.9
	30	41.9	**23.3**	34.5	29.0	31.5
USPS	0	6.3	**6.2**	7.2	7.0	6.5
	10	7.3	**6.6**	8.1	8.0	6.9
	20	9.7	**7.5**	9.6	9.1	7.9
	30	12.4	**8.7**	13.0	11.4	9.1
Cars	0	**5.5**	**5.5**	7.1	7.1	7.3
	10	**12.5**	**12.5**	12.7	12.7	13.3
	20	20.8	**17.0** (70)	20.1	18.3 (160)	20.0
	30	30.2	**22.9** (90)	27.6	24.1 (80)	28.3
Images	0	4.5	**4.2**	4.7	4.6	4.5
	10	8.6	**7.6** (70)	8.9	8.9 (210)	8.4
	20	15.1	**11.5** (160)	15.1	13.7 (280)	14.0
	30	22.5	**18.2** (150)	22.5	19.3 (210)	22.9
PenDigits	0	**12.9**	**12.9**	14.9	14.9	13.9
	10	15.2	**14.7**	16.3	16.3	14.8
	20	17.7	**16.3**	19.1	18.8	17.1
	30	21.7	**19.2**	23.1	21.3	19.4

The best results are marked in bold face.

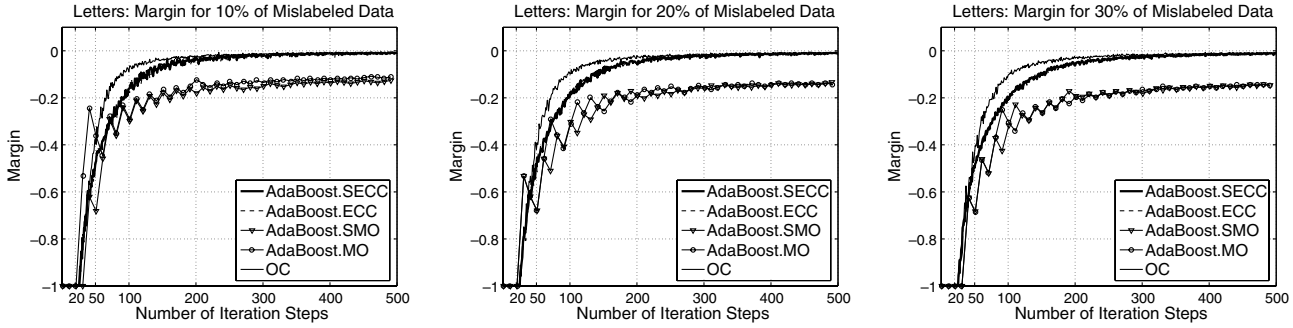

Fig. 11. Margin of the five algorithms on *Letters*.

several properties of ECC and OC, and derived the shrinkage versions of MO and ECC, referred to as SMO and SECC.

We have also conducted empirical studies to compare five multi-class AdaBoost algorithms. The experimental results showed: (1) ECC is faster than OC, while SECC is the slowest with respect to the convergence rate of the training error; (2) in the absence of mislabeling noise, ECC performs similarly to OC. However, for noise levels above zero, OC outperforms ECC, which is supported by our theoretical studies; (3) regularization of MO and ECC, by introducing the shrinkage parameter, improves significantly their performance in the presence of mislabeling noise. Overall, SECC performs significantly better than other four algorithms for all databases and all noise levels.

Appendix. Proof of Theorem 1

By comparing the steps in the algorithm of MO – in particular, the optimization of the hypothesis functions in Step (3) of Fig. 1, and the computation of weights α_t in Step (5) – with the corresponding expressions obtained from the minimization of G, we prove the theorem.

In the $(t-1)$th iteration, MO produces $\mathbf{F}_{t-1}(\mathbf{x}) = [F_{t-1}^{(1)}(\mathbf{x}), \ldots, F_{t-1}^{(L)}(\mathbf{x})]^{\mathrm{T}}$. In the tth iteration, the algorithm updates $\mathbf{F}_{t-1}(\mathbf{x})$ as

$$\begin{aligned}\mathbf{F}_t(\mathbf{x}) &= \mathbf{F}_{t-1}(\mathbf{x}) + \alpha_t[h_t^{(1)}(\mathbf{x}), \ldots, h_t^{(L)}(\mathbf{x})]^{\mathrm{T}} \\ &= \mathbf{F}_{t-1}(\mathbf{x}) + \alpha_t \mathbf{h}_t. \end{aligned} \tag{24}$$

From Eq. (3), the gradient of G with respect to $\mathbf{F}_{t-1}$, at $\mathbf{x} = \mathbf{x}_n$, reads $-\nabla_{\mathbf{F}_{t-1}} G|_{\mathbf{x}=\mathbf{x}_n} = [M(y_n,1)\mathrm{e}^{-F_{t-1}^{(1)}(\mathbf{x}_n)M(y_n,1)}, \ldots, M(y_n,L)\mathrm{e}^{-F_{t-1}^{(L)}(\mathbf{x}_n)M(y_n,L)}]$. Similar to Friedman's approach (Friedman, 2001), we find $\mathbf{h}_t$ as

$$\begin{aligned}\mathbf{h}_t &= \arg\min_{\{\mathbf{h}\in\mathscr{H},\beta\}} \sum_{n=1}^{N} \| - \nabla_{\mathbf{F}_{t-1}} G|_{\mathbf{x}=\mathbf{x}_n} - \beta(\mathbf{h}(\mathbf{x}_n))^{\mathrm{T}}\|^2 \\ &= \arg\max_{\mathbf{h}\in\mathscr{H}} \sum_{l=1}^{L}\sum_{n=1}^{N} \exp(-M(y_n,l)F_{t-1}^{(l)}(\mathbf{x_n}))M(y_n,l)h^{(l)}(\mathbf{x}_n), \end{aligned} \tag{25}$$

where β is a nuisance parameter, and $\mathbf{h} \in \mathscr{H}$ denotes that $\forall l,\ h^{(l)} \in \mathscr{H}$. The right-hand side of Eq. (25) can be conveniently expressed in terms of data distribution $\mathbf{D}$, computed in Step (6) in Fig. 1. First, by unravelling Step (6), we obtain

$$D_t(n,l) = \frac{\exp(-M(y_n,l)F_{t-1}^{(l)}(\mathbf{x}_n))}{\sum_{i=1}^{L}\sum_{j=1}^{N}\exp(-M(y_j,l)F_{t-1}^{(i)}(\mathbf{x}_j))}. \tag{26}$$

Then, by plugging Eq. (26) into Eq. (25), we get

$$\begin{aligned}\mathbf{h}_t &= \arg\max_{\mathbf{h}\in\mathscr{H}} \sum_{l=1}^{L}\sum_{n=1}^{N} D_t(n,l)M(y_n,l)h^{(l)}(\mathbf{x}_n) \\ &= \arg\min_{\mathbf{h}\in\mathscr{H}} \sum_{l=1}^{L}\left(\sum_{i=1}^{N} D_t(i,l)\right)\sum_{n=1}^{N}\frac{D_t(n,l)}{\sum_{i=1}^{N} D_t(i,l)}\mathbf{I}(M(y_n,l)\neq h^{(l)}(\mathbf{x}_n)). \end{aligned} \tag{27}$$

From Eq. (27), it follows that each $h_t^{(l)}$ should be selected from $\mathscr{H}$, such that the training error of the binary problem defined by the lth column is minimized. This is equivalent to Step (3) in Fig. 1.

After $\mathbf{h}_t$ is found, α_t can be computed by minimizing the cost function in Eq. (3). From Eqs. (3) and (24), the derivative of G_t with respect to α_t is $\partial G_t/\partial\alpha_t = -\sum_{l=1}^{L}\sum_{n=1}^{N}\exp(-(F_{t-1}^{(l)}(\mathbf{x}_n) + \alpha_t h_t^{(l)}(\mathbf{x}_n))M(y_n,l))h_t^{(l)}(\mathbf{x}_n)M(y_n,l)$. From $\partial G_t/\partial\alpha_t = 0$ and Eq. (26), we have

$$\sum_{l=1}^{L}\sum_{n=1}^{N} D_t(n,l)\exp(-\alpha_t h_t^{(l)}(\mathbf{x}_n)M(y_n,l))h_t^{(l)}(\mathbf{x}_n)M(y_n,l) = 0. \tag{28}$$

It follows that

$$\begin{aligned}&\sum_{l=1}^{L}\sum_{n=1}^{N} D_t(n,l)\mathrm{e}^{-\alpha_t}\mathbf{I}(h_t^{(l)}(\mathbf{x}_n) \\ &= M(y_n,l)) - \sum_{l=1}^{L}\sum_{n=1}^{N} D_t(n,l)\mathrm{e}^{\alpha_t}\mathbf{I}(h_t^{(l)}(\mathbf{x}_n)\neq M(y_n,l)) = 0, \end{aligned}$$

which yields $\alpha_t = \frac{1}{2}\ln[(1-\varepsilon_t)/\varepsilon_t]$, where ε_t is defined in Step (4) in Fig. 1. Thus, by minimizing the cost function G_t with respect to α_t, we obtain the same expression as the one in Step (5). This completes the proof.

References

Allwein, E.L., Schapire, R.E., Singer, Y., 2000. Reducing multiclass to binary: a unifying approach for margin classifiers. J. Mach. Learn. Res. 1, 113–141.

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

Breiman, L., 1999. Prediction games and arcing algorithms. Neural Comput. 11 (7), 1493–1517.

Crammer, K., Singer, Y., 2000. On the learnability and design of output codes for multiclass problems. In: Proc. 13th Annual Conf. Computational Learning Theory. Stanford University, CA, USA, pp. 35–46.

Dietterich, T.G., 2000. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach. Learn. 40 (2), 139–157.

Dietterich, T.G., Bakiri, G., 1995. Solving multiclass learning problems via error-correcting output codes. J. Artificial Intell. Res. 2, 263–286.

Freund, Y., Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. 55 (1), 119–139.

Friedman, J., 2001. Greedy function approximation: a gradient boosting machine. Ann. Statist. 29 (5), 1189–1232.

Friedman, J., Hastie, T., Tibshirani, R., 2000. Additive logistic regression: a statistical view of boosting. Ann. Statist. 28 (2), 337–407.

Grove, A.J., Schuurmans, D., 1998. Boosting in the limit: maximizing the margin of learned ensembles. In: Proc. 15th Natl. Conf. on Artificial Intelligence, Madison, WI, USA, pp. 692–699.

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

LeCun, Y., Boser, B., Denker, J.S., Hendersen, D., Howard, R., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1 (4), 541–551.

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

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

Quinlan, J., 1996. Bagging, boosting, and C4.5. In: Proc. 13th Natl. Conf. Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conf., Portland, OR, USA, pp. 725–730.

Rudin, C., Daubechies, I., Schapire, R.E., 2004. The dynamics of AdaBoost: cyclic behavior and convergence of margins. J. Mach. Learn. Res. 5, 1557–1595.

Schapire, R.E., 1997. Using output codes to boost multiclass learning problems. In: Proc. 14th Intl. Conf. Machine Learning. Nashville, TN, USA, pp. 313–321.

Schapire, R., Singer, Y., 1999. Improved boosting algorithms using confidence-rated predictions. Mach. Learn. 37 (3), 297–336.

Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S., 1998. Boosting the margin: a new explanation for the effectiveness of voting methods. Ann. Statist. 26 (5), 1651–1686.

Sun, Y., Todorovic, S., Li, J. Reducing the overfitting of AdaBoost by controlling its data distribution skewness. Int. J. Pattern Recog. Artificial Intell., in press.