A scalable direct formulation for multi-class boosting

Sakrapee Paisitkriangkrai, Chunhua Shen and Anton van den Hengel

***Abstract*—We present a scalable and effective classification model for multi-class boosting classification. In [1], a direct formulation for multi-class boosting was introduced. Unlike many existing multi-class boosting algorithms, which rely on output coding matrices, the approach in [1] directly maximizes the multi-class margin. The major problem of [1] is its extremely high computational complexity, which hampers its application on real-world problems. In this work, we propose a scalable and simple stage-wise boosting method, which also directly maximizes the multi-class margin. Our approach has several advantages: (1) it is simple and computationally efficient to train. The approach can speed up the training time by more than two orders of magnitude without sacrificing the classification accuracy. (2) Like traditional AdaBoost, it is parameter free and empirically demonstrates excellent generalization performance. In contrast, one has to tune the regularization parameter for the multi-class boosting of [1]. Experimental results on challenging multi-class machine learning and vision tasks demonstrate that the proposed approach substantially improves the convergence rate and accuracy of the final visual detector at no additional computational cost. Our new formulation opens up a new way of incorporating prior knowledge into multi-class problems, *e.g.*, feature sharing, hierarchical features, *etc*.**

***Index Terms*—Boosting, multi-class classification, column generation, convex optimization**

I. Introduction

Multi-class classification is one of the fundamental problems in machine learning and computer vision, as many real-world problems involve predictions which require an instance to be assigned to one of number of classes. Well known problems include handwritten character recognition [2], object recognition [3], and scene classification [4]. Compared to the well studied binary form of the classification problem, multi-class problems are considered more difficult to solve, especially as a number of classes increases.

In recent years a substantial body of work related to multi-class boosting has arisen in the literature. Many of these works attempt to achieve multi-class boosting by reformulating the task into a series of binary boosting problems. Often this is done through the use of output coding matrices. The common 1-vs-all and 1-vs-1 schemes are a particular example of this approach, and one where the coding matrices are predefined. The drawback of coding-based approaches is that they do not rapidly converge to low training errors on difficult data sets and many weak classifiers need to be learned (as is shown below in our experiments). As a result these algorithms fail to deliver the levels of performance required to process large data sets, or to achieve real-time data processing.

The aim of this paper is to develop a more direct boosting algorithm applicable to multi-class problems that will achieve the effectiveness and efficiency of previously proposed methods for binary classification. To achieve our goal we exploit the efficiency of the coordinate descent algorithm, *e.g.*, AdaBoost [5], along with more effective and direct formulations of multi-class boosting known as MultiBoost [1]. Our proposed approach is simpler than coding-based multi-class boosting since we do not need to learn output coding matrices. The approach is also fast to train, parameter free and has a faster convergence rate than coding-based techniques. Furthermore, the approach shares a similar property to ℓ_1-constrained maximum margin classifiers, in that it converges asymptotically to the ℓ_1-constrained solution.

Our approach is based on a novel stage-wise multi-class form of boosting which bypasses error correcting codes by directly learning base classifiers and weak classifiers' coefficients. The final decision function is a weighted average of multiple weak classifiers. The work we present in this paper intersects with several successful practical works, such as multi-class support vector machines [6], AdaBoost [5] and column generation based boosting [7], [8].

Our main contributions are as follows:

- Our approach is the first greedy stage-wise multi-class boosting algorithm which does not rely on codewords and which directly optimizes the boosting objective function. In addition, our approach converges asymptotically to the ℓ_1-constrained solution;
- We show that our minimization problem shares a connection with those derived from coordinate descent methods. In addition, our approach is less prone to over-fitting as techniques, such as shrinkage, can be easily adopted;
- We theoretically justify the use of random projections by proving the margin separability in the proposed boosting. The proof provides some insights in terms of the margin preservation and the minimal number of new projected dimensions to guarantee margin separability.
- Empirical results demonstrate that the approach exhibits the same classification performance as the state-of-the-art multi-class boosting-based classifier [1], but is significantly faster to train, and *orders of magnitude more scalable*. We have made the source code of the proposed boosting methods accessible at http://code.google.com/p/boosting/downloads/.

II. Related work

There exist a variety of multi-class boosting algorithms in the literature. Many of them solve multi-class learning tasks by reducing multi-class problems to multiple binary classification problems. We briefly review some well known boosting algorithms here in order to illustrate the novelty of the proposed approach.

Coding-based boosting was one of the earliest multi-class boosting algorithms proposed (see, for example, AdaBoost.MH [9], AdaBoost.MO [9], AdaBoost.ECC [10], AdaBoost.SIP [11] and JointBoost [12]). Coding-based approaches perform multi-class classification by combining the outputs of a set of binary classifiers. This includes popular methods such as 1-vs-all and 1-vs-1, for example. Typically, a coding matrix, $M \in \{-1, 0, +1\}^{k \times t}$, is constructed (where k is the number of classes and t is the length of a codeword). The algorithm learns a binary classifier $h_j(\cdot)$, corresponding to a single column[1] of M, in a stage-wise manner. A test instance is classified as belonging to the class associated with the codeword closest in Hamming distance to the sequence of predictions generated by $h_1(\cdot), \cdots, h_t(\cdot)$. The final decision function for a test datum $\boldsymbol{x}$ is $F(\boldsymbol{x}) = \underset{r}{\operatorname{argmax}} \sum_{j=1}^{t} M(r, j) \boldsymbol{w}_j h_j(\boldsymbol{x})$. Clearly, the performance of the algorithm is largely influenced by the quality of the coding

The authors are with The Australian Center for Visual Technologies, The University of Adelaide, SA 5005, Australia (e-mail: {paul.paisitkriangkrai, chunhua.shen, javen.shi, anton.vandenhengel}@adelaide.edu.au). Correspondence should be addressed to C. Shen.

[1]Each column of M defines a binary partition of k classes over data.

matrices. Finding optimum coding matrices, which mean identifying classes which should be grouped together, is often non-trivial. Several algorithms, *e.g.*, max-cut and random-half, have been proposed to build optimal binary partitions for coding matrices [13]. Max-cut finds the binary partitions that maximize the error-correcting ability of coding matrix while random-half randomly splits the classes into two groups. Li [14] points out that random-half usually performs better than max-cut because the binary problems formed by max-cut are usually too hard for base classifiers to learn. Nonetheless, both max-cut and random-half do not achieve the best performance as they do not consider the ability of base classifiers in the optimization of coding matrices. In contrast, our proposed approach bypasses the learning of output coding by learning base classifiers and weak classifiers' coefficients directly.

Another related approach, which trains a similar decision function, is the multi-class boosting of Duchi and Singer known as GradBoost [15]. The main difference between GradBoost and the method we propose here is that GradBoost does not directly optimize the boosting objective function. GradBoost bounds the original non-smooth ℓ_1 optimization problem by a quadratic function. It is not clear how well the surrogate approximates the original objective function. In contrast, our approach solves the original loss function, which is the approach of AdaBoost and LogitBoost. Shen and Hao have introduced a direct formulation of multi-class boosting in the sense that it directly maximizes the multi-class margin. By deriving a meaningful Lagrange dual problem, column generation is used to design a fully corrective boosting method [1]. The main issue of [1] is its extremely heavy computation burden, which hampers its application on real data sets. Unlike their work, our approach learns a classification model in a stage-wise manner. In our work, only the coefficients of latest weak classifiers need to be updated. As a result, our approach is significantly more computationally efficient and robust to the regularization parameter value chosen (as long as the value is set to be small). Compared to [1], at each boosting iteration, our approach only needs to solve for k variables instead of $k \cdot t$ variables, where k is the number of classes and t is the number of current boosting iterations. This significant reduction in the size of the problem to be solved at each iteration is responsible for the *orders of magnitude reduction* in training time required.

Notation Let $(\boldsymbol{x}_i, y_i)_{i=1}^m$ be the set of training data, where $\boldsymbol{x}_i \in \mathbf{R}^d$ represents an instance, and $y_i \in \{1, 2, \cdots, k\}$ the corresponding class label (where m is the number of training samples and k is the number of classes). Let $h(\cdot)$ denote a binary weak classifier which projects an instance $\boldsymbol{x}$ into $\{-1, +1\}$. By assuming that we have a total of n possible weak classifiers, the output of all weak learners can be represented as $H \in \mathbf{R}^{m \times n}$, where H_{ij} is the label predicted by weak classifier $h_j(\cdot)$ on the training data $\boldsymbol{x}_i$. Each row $H_{i:}$ of the matrix H represents the output of all weak classifiers when applied to a single training instance $\boldsymbol{x}_i$. We build a classifier of the form

$$F(\boldsymbol{x}) = \operatorname*{argmax}_r \textstyle\sum_{j=1}^t W_{jr} h_j(\boldsymbol{x}) \tag{1}$$

where $W \in \mathbf{R}^{t \times k}$. Each column of W, $\boldsymbol{w}_r$, contains coefficients of the linear classifier for class r and each row of W, $W_{j:}$, consists of the coefficients for the weak classifier $h_j(\cdot)$ for all class labels. The predicted label is the index of the row of W attaining the highest sum.

III. Our approach

In order to classify an example $(\boldsymbol{x}_i, y_i)$ correctly, $H_{i:}\boldsymbol{w}_{y_i}$ must be greater than $H_{i:}\boldsymbol{w}_r$, for any $r \neq y_i$. In this paper, we define a set of margins associated with a training example as,

$$\rho_r(\boldsymbol{x}_i, y_i) = \rho_{i,r} = H_{i:}\boldsymbol{w}_{y_i} - H_{i:}\boldsymbol{w}_r, r = 1, \cdots, k. \tag{2}$$

The training example $\boldsymbol{x}_i$ is correctly classified only when $\rho_{i,r} \geq 0$. In boosting, we train a linear combination of basis functions (weak classifiers) which minimizes a given loss function over predefined training samples. This is achieved by searching for the dimension which gives the steepest descent in the loss and assigning its coefficient accordingly at each iteration. Commonly applied loss functions are exponential loss of AdaBoost [5] and binomial log-likelihood loss of LogitBoost [16].

$$\begin{aligned} &\text{Exponential} && L_{\exp}(\boldsymbol{x}_i, y_i) = \exp(-\rho_{i,r}) \\ &\text{Logistic} && L_{\log}(\boldsymbol{x}_i, y_i) = \log(1 + \exp(-\rho_{i,r})) \end{aligned}$$

The two losses behave similarly for positive margin but differently for negative margin. $L_{\log}$ has been reported to be more robust against outliers and misspecified data compared to $L_{\exp}$. In the rest of this section, we present a coordinate descent based multi-class boosting as an approximate ℓ_1-regularized fitting. We then illustrate the similarity between both approaches. Finally, we discuss various strategies that can be adopted to prevent over-fitting.

A. Stage-wise multi-class boosting

In this section, we design an efficient learning algorithm which maximizes the margin of our training examples, $\rho_r(\boldsymbol{x}_i, y_i), r = 1, \cdots, k$. The general ℓ_1-regularized optimization problem we want to solve is

$$\min_W \quad \textstyle\sum_{i=1}^m \sum_{r=1}^k L(\boldsymbol{x}_i, y_i) + \nu \|W\|_1, \quad \text{s.t.: } W \geq 0. \tag{3}$$

where L can be any convex loss functions and parameter ν controls the trade off between model complexity and small error penalty. Although (3) is ℓ_1-norm regularized, it is possible to design our algorithm with other ℓ_p-norm regularized. We first derive the dual of both exponential loss and logistic loss, and propose our new stage-wise multi-class boosting.

Exponential loss The learning problem for an exponential loss can be written as,

$$\begin{aligned} \min_W \quad & \textstyle\sum_{i=1}^m \sum_{r=1}^k \exp\left[-\left(H_{i:}\boldsymbol{w}_{y_i} - H_{i:}\boldsymbol{w}_r\right)\right] + \nu \|W\|_1, \\ \text{s.t.:} \quad & W \geq 0; \end{aligned} \tag{4}$$

We introduce auxiliary variables, $\rho_{i,r}$, and rewrite the primal problem as,

$$\begin{aligned} \min_{W,\boldsymbol{\rho}} \quad & \log\Big(\sum\nolimits_{i,r} \exp\left(-\rho_{i,r}\right)\Big) + \nu \|W\|_1, \\ \text{s.t.:} \quad & \rho_{i,r} = H_{i:}\boldsymbol{w}_{y_i} - H_{i:}\boldsymbol{w}_r, \forall i, \forall r, W \geq 0; \end{aligned} \tag{5}$$

where (i, r) represents the joint index through all of the data and all of the classes. Here we work on the logarithmic version of the original cost function. Since $\log(\cdot)$ is strictly monotonically increasing, this does not change the original optimization problem. Based on the fact that the conjugate of the log-sum-exp function is the negative entropy function, the Lagrange dual problem can be derived as

$$\begin{aligned} \min_U \quad & \textstyle\sum_{i,r} U_{ir} \log(U_{ir}), \\ \text{s.t.:} \quad & \textstyle\sum_i \left[\delta_{r,y_i}\left(\sum_{l=1}^k U_{il}\right) - U_{ir}\right] H_{i:} \leq \nu \mathbf{1}^\top, \\ & \textstyle\sum_{i,r} U_{ir} = 1, U \geq 0. \end{aligned} \tag{6}$$

where $\delta_{s,t}$ denotes the indication operator such that $\delta_{s,t} = 1$ if $s = t$ and $\delta_{s,t} = 0$, otherwise. The objective function of the dual encourages the dual variables, U, to be uniform.

Logistic loss The learning problem can be expressed as,

$$\begin{aligned} \min_{W,\boldsymbol{\rho}} \quad & \textstyle\sum_{i,r} \log\big(1 + \exp\left(-\rho_{i,r}\right)\big) + \nu \|W\|_1, \\ \text{s.t.:} \quad & \rho_{i,r} = H_{i:}\boldsymbol{w}_{y_i} - H_{i:}\boldsymbol{w}_r, \forall i, \forall r, W \geq 0; \end{aligned} \tag{7}$$

Using the fact that the conjugate of the logistic loss function $\log\big(1+\exp(-x)\big)$ is $(-u)\log(-u)+(1+u)\log(1+u)$, the Lagrange dual can be written as [2],

$$\begin{aligned}\min_{U} \quad & \textstyle\sum_{i,r}\big[U_{ir}\log(U_{ir})+(1-U_{ir})\log(1-U_{ir})\big], \qquad (8)\\ \text{s.t.:} \quad & \textstyle\sum_i\left[\delta_{r,y_i}\left(\sum_{l=1}^{k}U_{il}\right)-U_{ir}\right]H_{i:}\le\nu\mathbf{1}^\top,\forall r;\\ & \textstyle\sum_{i,r}U_{ir}=1, U\ge 0.\end{aligned}$$

Although both dual problems have identical constraints, we will show later that their solutions (selected weak classifiers and coefficients) are slightly different.

Learning weak classifiers From the dual, the set of constraints can be infinitely large[3], *i.e.*,

$$\textstyle\sum_i\left[\delta_{r,y_i}\left(\sum_{l=1}^{k}U_{il}\right)-U_{ir}\right]h(\boldsymbol{x}_i)\le\nu,\forall r,\forall h(\cdot)\in\mathcal{H}. \qquad (9)$$

Similar to LPBoost, we apply a technique known as column generation to identify an optimal set of constraints[4] [17]. At each iteration, we find the most violated constraint to the dual problem. The subproblem for generating weak classifiers is,

$$h^*(\cdot)=\underset{h(\cdot)\in\mathcal{H},r}{\operatorname{argmax}}\ \textstyle\sum_{i=1}^{m}\left[\delta_{r,y_i}\left(\sum_{l=1}^{k}U_{il}\right)-U_{ir}\right]h(\boldsymbol{x}_i). \qquad (10)$$

Solving this subproblem is identical to finding a weak classifier with minimal weighted error in AdaBoost (since dual variables, U, can be viewed as sample weights). In each iteration, we add one more constraint into the dual problem. The process continues until we can not find any violated constraints. Through Karush-Kunh-Tucker (KKT) optimality condition, the gradient of Lagrangian over primal variable, $\boldsymbol{\rho}$, and dual variable, U, must vanish at the optimal. The KKT condition gives the relationship between the optimal primal and dual variables as,

$$\text{Exponential}: \qquad U^*_{ir}=\frac{\exp(-\rho^*_{i,r})}{\sum_{i,r}\exp(-\rho^*_{i,r})}, \qquad (11)$$

$$\text{Logistic}: \qquad U^*_{ir}=\frac{\exp(-\rho^*_{i,r})}{\big(1+\exp(-\rho^*_{i,r})\big)}. \qquad (12)$$

Optimizing weak learners' coefficients Weak learners' coefficients can be calculated in a totally corrective manner as in [8]. However, the drawback of [8] is that the primal variable, W, is updated at every boosting iteration. Hence the training time is often slow when the number of training samples and classes are large. In this paper, we propose a more efficient approach based on a stage-wise algorithm similar to those derived in AdaBoost. The advantages of our approaches compared to [8] are (1) it is computationally efficient as we only update weak learners' coefficient at the current iteration and (2) our method is parameter free, as a result, the training time is faster since we no longer have to cross validate the regularization parameter. We will show later in our experiments that the regularization parameter only needs to be set to a very small value. By inspecting the primal problem, (5) and (7), the optimal W can be calculated analytically as follows. At iteration t, we fix the value of $W_{1,:}$, $W_{2,:}$, $\cdots$, $W_{t-1,:}$. So $W_{t,:}$ is the only variable to optimize. The primal cost function for exponential loss can then be written as,

$$\begin{aligned}\min_{W_{t,:}} \quad & \log\Big(\textstyle\sum_{i=1}^{m}\sum_{r=1}^{k}U_{ir}^{t-1}\exp\left(-\rho_{i,r}^{t}\right)\Big)+\nu\|W_{t,:}\|_1, \qquad (13)\\ \text{s.t.:} \quad & W_{t,:}\ge 0;\end{aligned}$$

where $U_{ir}^{t-1}=\exp\left(-\sum_{j=1}^{t-1}\rho_{i,r}^{j}\right)$ and $\rho_{i,r}^{j}=h_j(\boldsymbol{x}_i)W_{j,y_i}-h_j(\boldsymbol{x}_i)W_{j,r}$. Here we drop the terms that are irrelevant to $W_{t,:}$. U_{ir}^{0} is initialized to $\frac{1}{mk}$ In each iteration, we compute U_{ir}^{t} and cache the value for the next iteration. Similarly, the cost function for logistic loss is,

$$\begin{aligned}\min_{W_{t,:}} \quad & \textstyle\sum_{i=1}^{m}\sum_{r=1}^{k}\log\Big(1+U_{ir}^{t-1}\exp\left(-\rho_{i,r}^{t}\right)\Big)+\nu\|W_{t,:}\|_1, \qquad (14)\\ \text{s.t.:} \quad & W_{t,:}\ge 0;\end{aligned}$$

The above primal problems, (13) and (14), can be solved using an efficient Quasi-Newton method like L-BFGS-B, and the dual variables can be obtained using the KKT condition, (11). The details of our multi-class stage-wise boosting algorithm are given in Algorithm 1.

Algorithm 1 Stage-wise based multi-class boosting

Input:
1) A set of examples $\{\boldsymbol{x}_i, y_i\}$, $i=1\cdots m$;
2) The maximum number of weak classifiers, t;
Output: A multi-class classifier $F(\boldsymbol{x})=\underset{r}{\operatorname{argmax}}\sum_{j=1}^{t}W_{jr}h_j(\boldsymbol{x})$
Initilaize:
1) $j\leftarrow 0$;
2) Initialize sample weights, $U_{ir}=\frac{1}{mk}$;
while $j<t$ **do**
  ① Train a weak learner, $h_t(\cdot)$, by solving the subproblem (10);
  ② If the stopping criterion, $\sum_i\left[\delta_{r,y_i}\left(\sum_{l=1}^{k}U_{il}\right)-U_{ir}\right]h_j(\boldsymbol{x}_i)\le\nu+\epsilon,\forall r$, has been met, we exit the loop;
  ③ Add the best weak learner, $h_j(\cdot)$, into the current set;
  ④ Solve the primal problem: (13) for exponential loss or (14) for logistic loss;
  ⑤ Update sample weights (dual variables), (11);
  ⑥ $j\leftarrow j+1$;

Computational complexity In order to appreciate the performance gain, we briefly analyze the complexity of the new approach and MultiBoost [1]. The time consuming step in Algorithm 1 is in step 1 (weak classifier learning) and 4 (calculating coefficients). In step 1, we train a weak learner by solving the subproblem (10). For simplicity, we use decision stumps as weak learners. The fastest way to train the decision stump is to sort feature values and scan through all possible threshold values sequentially to update (10). The algorithm takes $O(m\log m)$ for sorting and $O(km)$ for scanning k classes. In each iteration, we need to train d decision stumps. Hence, this step takes $O(dm\log m+dkm)$ at each iteration. In step 4, we solve k variables at each iteration. Let us assume the computational complexity of L-BFGS is roughly $O(n^{2.5})$. The algorithm spends $O(k^{2.5})$ at each iteration. Hence, the total time complexity for t boosting iterations is $O(tdm\log m+tdkm+tk^{2.5})$. Roughly, the first term dominates when the number of samples is large and the last term dominates when the number of classes is large. For MultiBoost, the time complexity of weak classifier learning would be the same as ours. However, in step 4, they would need to solve kt variables (since the algorithm is totally corrective). The time complexity for this step[5] is $O(k^{2.5}+(2k)^{2.5}+\cdots+(tk)^{2.5})>O(k^{2.5}t^3)$. Hence, the total time complexity for Multiboost is $>O(tdm\log m+tdkm+k^{2.5}t^3)$. Clearly, the last term will dominate when the number of iterations is

[2]Note that the sign of U has been reversed.

[3]For decision stumps, the size of $\mathcal{H}$ is the the number of features times the number of samples.

[4]Note that constraints in the dual correspond to variables in the primal. An optimal set of constraints in the dual would correspond to a discriminative set of variables in the primal that we are interested in.

[5]$1^2+2^2+\cdots+n^2\approx O(n^3)$ and $1^3+2^3+\cdots+n^3\approx O(n^4)$

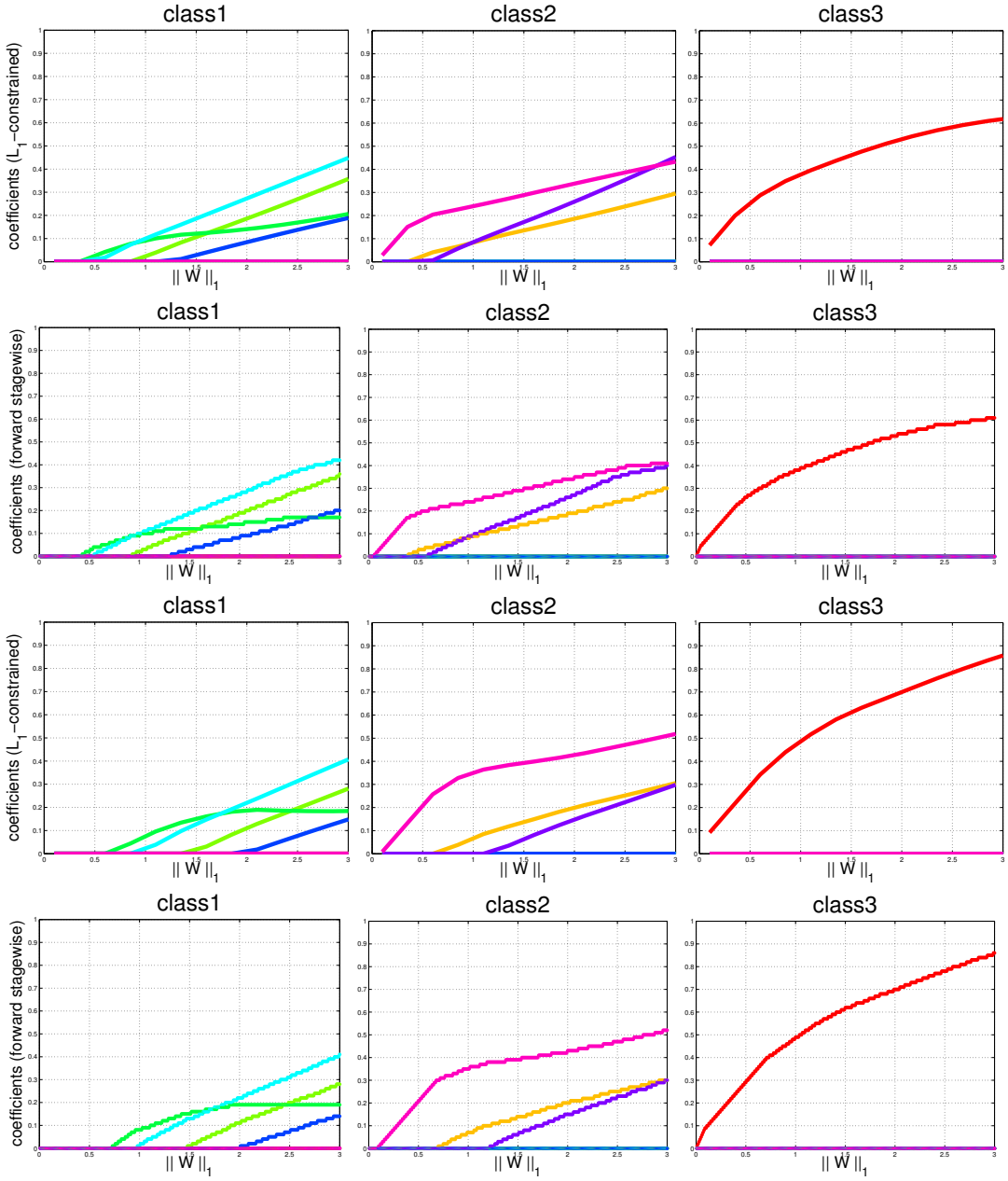

Fig. 1. Best viewed in color. *Top:* Exact coefficient paths for ℓ_1-constrained **exponential loss** *Second row:* Coefficient paths of our stage-wise boostings (**exponential loss**) *Third row:* Exact coefficient paths for ℓ_1-constrained **logistic loss** (17) *Bottom:* Coefficient paths of our stage-wise boostings (**logistic loss**). We train multi-class classifiers on USPS data sets (digit 3,6 and 9). Each figure corresponds to each digit and each curve in the figure corresponds to their coefficients value. Note the similarity between coefficients of both algorithms.

large. For example, training a multi-class classifier with 100 samples, 100 features, 10 classes for 1000 iterations using our approach would require $O(10^8)$ while MultiBoost would require $O(10^{11})$.

B. Discussion

Binary classification Here we briefly point out the connection between our multi-class formulation and binary classification algorithms such as AdaBoost. We note that AdaBoost sets the regularization parameter, ν in (4), to be very close to zero [18] and it minimizes the original exponential loss function. We can simplify our exponential loss learning problem, (4), for a binary case ($k = 2$) as,

$$\begin{aligned} \min_{W} \quad & \textstyle\sum_{i=1}^{m}\sum_{r=1}^{2} \exp\left[-\left(H_{i:}\boldsymbol{w}_{y_i} - H_{i:}\boldsymbol{w}_r\right)\right] \quad \text{s.t.: } W \geq 0; \\ = \min_{\boldsymbol{\alpha}} \quad & \textstyle\sum_{i=1}^{m} \exp\left(-z_i H_{i:}\boldsymbol{\alpha}\right), \quad \text{s.t.: } \boldsymbol{\alpha} \geq 0, \end{aligned} \tag{15}$$

where $z_i = 1$ if $y_i = 1$, $z_i = -1$ if $y_i = 2$ and $\boldsymbol{\alpha} = \boldsymbol{w}_1 - \boldsymbol{w}_2$. AdaBoost minimizes the exponential loss function via coordinate descent. At iteration t, Adaboost fixes the value of $\alpha_1, \alpha_2, \cdots, \alpha_{t-1}$ and solve α_t. (15) can simply be simplified to,

$$\min_{\alpha_t} \quad \textstyle\sum_{i=1}^{m} u_i^{t-1} \exp\left(-\rho_i^t\right), \quad \text{s.t.: } \alpha_t \geq 0; \tag{16}$$

where $u_i^{t-1} = \exp\left(-\sum_{j=1}^{t-1}\rho_i^j\right)$ and $\rho_i^j = y_i h_j(\boldsymbol{x}_i)\alpha_j$. By setting the first derivative of (16) to zero, a closed form solution of α_t is: $\alpha_t^* = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$ where $\epsilon_t = \sum_{i:y_i \neq h_t(\boldsymbol{x}_i)} u_i^{t-1}$. ϵ_t corresponds to a weighted error rate with respect to the distribution of dual variables. By replacing step 4 in Algorithm 1 with (15), our approach would yield an identical solution to AdaBoost.

ℓ_1-constrained classifier and our boosting Rosset *et al*. pointed out that by setting the coefficient value to be small, gradient-based boosting tends to follow the solution of ℓ_1-constrained maximum margin classifier, (17), as a function of γ under some mild conditions [19].

$$W^*(\gamma) = \min_{W} \quad \sum_i L(y_i, F(\boldsymbol{x}_i)), \qquad (17)$$
$$\text{s.t.:} \quad \|W\|_1 \leq \gamma, W \geq 0;$$

We conducted a similar experiment on our multi-class boosting to illustrate the similarity between our forward stage-wise boosting and the optimal solution of (17) on USPS data set. The data set consists of 256 pixels. We randomly select 20 samples from 3 classes (3, 6 and 9). For ease of visualization and interpretation, we limit the number of available decision stumps to 8. We first solve (17) using `CVX` package [6] [20]. We then train our stage-wise boosting as discussed previously. However, instead of solving (13) or (14), we set the weak learner's coefficient of the selected class in (10) to be 0.01 and the weak learner's coefficient of other classes to be 0. The learning algorithm is run for 1000 boosting iterations. The coefficient paths of each class are plotted in the second row in Fig. 1 (the first three columns correspond to exponential loss and the the last three correspond to logistic loss). We compare the coefficient paths for our boosting and ℓ_1-constrained exponential loss and logistic loss in Fig. 1. We observe that both algorithms give very similar coefficients. This experimental evidence leads us to the connection between the solution of our multi-class boosting and the solution of (17). Rosset *et al*. have also pointed this out for a binary classification problem [19]. The authors incrementally increase the coefficient of the selected weak classifiers by a very small value and demonstrate that the final coefficient paths follow the ℓ_1-regularized path. In this section, we have demonstrated that our multi-class boosting also asymptotically converges to the optimal solution of ℓ_1-regularized solution (17).

Shrinkage and bounded step-size In order to minimize over-fitting, strategies such as shrinkage [16] and bounded step-size [21] can also be adopted here. We briefly discuss each method and how they can be applied to our approach. As discussed in previous section, at iteration t, we solve,

$$W_{t,:}^* = \operatorname*{argmin}_{W_{t,:}} \sum_i L\big(y_i, F^t(\boldsymbol{x}_i) + W_{t,:}h_t(\boldsymbol{x}_i)\big).$$

The alternative approach, as suggested by [16], is to shrink all coefficients to small values. Shrinkage is simply another form of regularization. The algorithm replaces $W_{t,:}$ with $\eta W_{t,:}$ where $0 < \eta < 1$. Since, η decreases the step-size, η can be viewed as a learning rate parameter. The smaller the value of η, the higher the overall accuracy as long as there are enough iterations. Having a large enough iteration means that we can keep selecting the same weak classifier repeatedly if it remains optimal. [16] observed that shrinkage often produces a better generalization performance compared to line search algorithms. Similar to shrinkage, bounded step-size caps $W_{t,:}$ by a small value, $W_{t,:} = \min(W_{t,:}, \kappa)$ where κ is often small. The method decreases the step-size for producing a better generalization performance.

[6]Note that to solve (17), the algorithm must access all weak classifiers a priori.

IV. Experiments

Regularization parameters and shrinkage In this experiment, we evaluate the performance of our algorithms on different regularization parameters, ν in (3), and the shrinkage parameter, η. We use 5 benchmark multi-class data sets. We choose 50 random samples from each class and randomly split the data into two groups: 80% for training and the rest for evaluation. We set the maximum number of boosting iterations to 1000. Each data set is randomly split into two groups: The first experiment demonstrates that, as long as ν is sufficiently small, the final performance is not affected by the value of ν. In other words, cross validation is not required to achieve optimal results. We experiment with ν from $\{\ 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}\ \}$ on both exponential loss and logistic loss. Table I reports the final classification errors. We observe that all classifiers perform similarly. In the next experiment, we evaluate how shrinkage helps improve the generalization performance. We vary the value of η between 0.1 and 1.0 and keep the maximum number of boosting iterations to 1000. Experimental results are reported in Table II. We observe a slight increase in generalization performances in all data sets when shrinkage is applied. In the rest of our experiment, we set ν to be 10^{-9} and apply a shrinkage value of 0.5.

Comparison to MultiBoost In this experiment, we compare our algorithm to MultiBoost, a totally corrective multi-class boosting proposed in [1]. We first compare the average coefficient calculation time of our approach and MultiBoost. We use decision stumps as the weak classifier. For MultiBoost, we use the logistic loss and choose the regularization parameter from $\{\ 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\ \}$ by cross-validation. For our algorithm, we set ν to 10^{-9} and η to 0.5. All experiments are repeated 5 times using the same regularization parameter. All algorithms are implemented in MATLAB using a single processor. The weak learner training (decision stump) is written C-mex. We use MATLAB interface for L-BFGS-B [22] to solve (13) and (14). The maximum number of L-BFGS-B iterations is set to 100. The iteration stops when the projection gradient is less than 10^{-5} or the difference between the objective value of current iteration and previous iteration is less than 10^{-9}. We use the data set *letter* from the UCI repository and vary the number of classes, the number of training samples and the number of boosting iterations. The results are shown in Table III and Fig. 2. Clearly, our approach performs comparable to MultiBoost while having a fraction of the training time of MultiBoost.

In the next experiment, we evaluate our approaches on several UCI data sets. We choose 50 random samples from each class and randomly split the data into training and test sets at a ratio of 75:25. For data sets with large number of dimensions, we perform dimensionality reduction using PCA. Our PCA projected data captures 90% of the original data variance. We set the number of boosting iterations to 500 for all algorithms. On data sets with large number of classes, *e.g*. news, letter, rcv1 and sector, the training time is too long. To compare the accuracy, we implement the objective function and the gradient of the objective function for L-BFGS-B as C-mex. Table IV reports average test errors and the time it takes to compute W of different algorithms. Based on the results, our stage-wise approach performs comparable and sometimes slightly better than MultiBoost. We suspect that we have not fine-tuned the regularization parameter used in MultiBoost. In terms of training time, our algorithm is a lot faster to train than MultiBoost. We have observed significant speedup factors (at least two orders of magnitude) depending on the complexity of the optimization problem and the number of classes.

Multi-class boostings on UCI data sets Next we compare our approaches against some well known multi-class boosting algorithms: AdaBoost.MH [9], AdaBoost.ECC [10], AdaBoost.MO [9] and

TABLE I

AVERAGE TEST ERRORS WITH DIFFERENT VALUES OF ν IN (3). ALL EXPERIMENTS ARE RUN 5 TIMES WITH 1000 BOOSTING ITERATIONS. THE AVERAGE ERROR MEAN AND STANDARD DEVIATION (SHOWN IN %) ARE REPORTED. BEST AVERAGE PERFORMANCES ARE SHOWN IN BOLDFACE. THE FIRST COLUMN DISPLAYS THE DATA SET NAME AND THE NUMBER OF CLASSES. FROM THE TABLE, THERE IS NO SIGNIFICANT DIFFERENCE BETWEEN THE FINAL CLASSIFICATION PERFORMANCE OF VARIOUS ν (AS LONG AS IT IS SUFFICIENTLY SMALL)

	Exponential				Logistic			
ν	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-10}	10^{-9}	10^{-8}	10^{-7}
iris (3)	8.0 (6.1)	8.0 (6.1)	8.0 (6.1)	**7.3 (6.0)**	6.0 (5.5)	**5.3 (4.5)**	7.3 (5.5)	6.7 (5.3)
glass (6)	28.8 (8.7)	28.2 (8.7)	29.4 (8.6)	**24.7 (7.4)**	26.5 (6.6)	**25.9 (5.3)**	**25.9 (5.7)**	26.5 (7.5)
usps (10)	**10.0 (3.9)**	**10.0 (3.9)**	16.0 (1.6)	13.0 (3.9)	**12.0 (2.4)**	12.2 (2.6)	**12.0 (2.4)**	12.8 (3.3)
pen (10)	**9.2 (0.8)**	9.8 (1.5)	10.2 (2.6)	10.8 (2.2)	**7.4 (3.1)**	8.2 (3.4)	9.0 (4.4)	8.6 (2.0)
news (20)	64.0 (4.5)	64.0 (4.5)	**62.3 (5.8)**	63.1 (2.8)	60.8 (4.8)	61.6 (3.7)	60.3 (3.6)	**60.1 (2.2)**

TABLE II

AVERAGE TEST ERRORS WITH DIFFERENT SHRINKAGE PARAMETERS, η. THE AVERAGE ERROR MEAN AND STANDARD DEVIATION (SHOWN IN %) ARE REPORTED. WE OBSERVE A SLIGHT INCREASE IN GENERALIZATION PERFORMANCES IN ALL DATA SETS WHEN SHRINKAGE IS APPLIED.

	Exponential				Logistic			
η	1.0	0.5	0.2	0.1	1.0	0.5	0.2	0.1
iris	8.0 (6.1)	7.3 (4.9)	6.7 (4.1)	**6.0 (3.7)**	**5.3 (4.5)**	8.0 (5.0)	7.3 (4.9)	7.3 (4.9)
glass	28.2 (8.7)	**26.4 (7.5)**	27.7 (4.5)	27.7 (5.7)	25.9 (5.3)	**24.7 (4.5)**	28.8 (5.7)	26.5 (5.1)
usps	10.0 (3.9)	9.8 (3.0)	**7.8 (4.2)**	9.6 (4.3)	12.2 (2.6)	**7.4 (3.0)**	**7.4 (4.6)**	9.2 (3.8)
pen	9.8 (1.5)	**8.2 (1.9)**	8.6 (2.1)	9.0 (1.9)	**8.2 (3.4)**	**8.2 (1.5)**	8.4 (2.1)	9.0 (1.9)
news	64.0 (4.5)	53.6 (2.8)	**52.6 (4.3)**	53.4 (3.6)	61.6 (3.7)	53.6 (4.3)	52.8 (4.4)	**52.2 (2.7)**

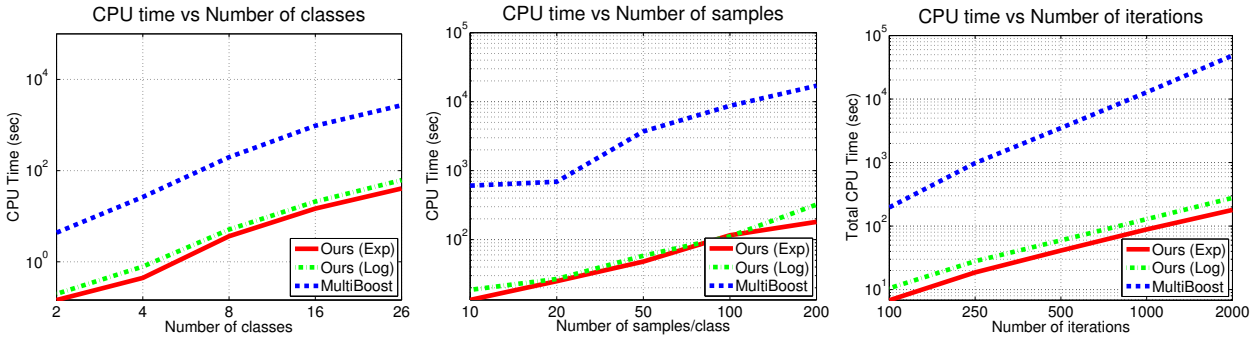

Fig. 2. Average CPU time on a log-log scale. The vertical axis corresponds to the required CPU time to calculate weak learners' coefficients (step 4 in Algorithm 1). In this figure, we vary the number of classes and boosting iterations. Our algorithm is at least two orders of magnitude faster than MultiBoost.

TABLE III

AVERAGE TEST ERRORS BETWEEN OUR APPROACH AND MULTIBOOST BY VARYING THE NUMBER OF CLASSES, TRAINING SAMPLES PER CLASS AND BOOSTING ITERATIONS. ALL ALGORITHMS PERFORM SIMILARLY

# classes	4	8	16	26
Ours (Exp)	**7.6(2.2)**	20.8(4.3)	**21.7(2.6)**	25.6(2.2)
Ours (Log)	**7.6(2.6)**	**20.6(4.9)**	21.9(2.9)	**24.7(2.4)**
MultiBoost [1]	8.4(1.7)	22.2(3.1)	23.9(2.2)	28.5(2.0)
# samples	20	50	100	200
Ours (Exp)	**27.8(5.1)**	23.6(1.1)	21.1(1.0)	18.2(1.2)
Ours (Log)	**27.8(4.1)**	**22.7(1.2)**	20.3(1.4)	17.5(1.2)
MultiBoost [1]	29.9(4.3)	25.7(1.3)	**19.9(0.9)**	**16.2(1.4)**
# iterations	250	500	1000	2000
Ours (Exp)	27.7(1.8)	25.0(2.2)	24.2(2.3)	**24.2(1.5)**
Ours (Log)	**25.9(2.3)**	**24.5(1.9)**	**23.8(1.6)**	24.3(1.3)
MultiBoost [1]	27.1(1.5)	26.9(2.0)	26.9(0.7)	26.9(0.7)

GradBoost [15]. For AdaBoost.ECC, we perform binary partition using the random-half method [14] For GradBoost, we implement ℓ_1/ℓ_2-regularized multi-class boosting and choose the regularization parameter from $\{ 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1} \}$. All experiment are repeated 10 times. The maximum number of boosting iterations is set to 1000. We plot average test errors versus number of weak classifiers in Fig. 3 and Table V. On UCI data sets, we observe that all methods perform comparably. However, our algorithm performs better than other algorithms when the number of classes is large (> 10).

MNIST handwritten digits Next we evaluate our approach on well known handwritten digit data sets. We first resize the original image to a resolution of 28×28 pixels and apply a de-skew pre-processing. We then apply a spatial pyramid and extract 3 levels of HOG features with 50% block overlap. The block size in each level is 4×4, 7×7 and 14×14 pixels, respectively. Extracted HOG features from all levels are concatenated. In total, there are $2,172$ HOG features. We train our classifiers using $60,000$ training samples and test it on the original test sets of $10,000$ samples. We train $1,000$ boosting iterations and the results are briefly summarized in Table VI. Our algorithm performs best compared to other evaluated algorithms.

Scene recognition In the next experiment, we compare our approach on the 15-scene data set used in [4]. The set consists of 9 outdoor scenes and 6 indoor scenes. There are $4,485$ images in total. For each run, we randomly split the data into a training set and a test set based on published protocols. This is repeated 5 times and the average accuracy is reported. In each train/test split, a visual codebook is generated. Both training and test images are then transformed into histograms of code words. We use CENTRIST [24] as our feature descriptors. We build 200 visual code words using the histogram intersection kernel (HIK), which has been shown to outperform k-means and k-median [24]. We represent each image in a spatial hierarchy manner [25]. Each image consists of 31 sub-windows. An image is represented by the concatenation of histograms

TABLE IV

AVERAGE TEST ERRORS AND THE TIME TO CALCULATE W FOR DIFFERENT ALGORITHMS. ALL EXPERIMENTS ARE RUN 5 TIMES WITH 500 BOOSTING ITERATIONS. CLEARLY, ALL ALGORITHMS PERFORM COMPARABLY. HOWEVER, THE TRAINING TIME OF OUR ALGORITHMS IS MUCH LESS THAN MULTIBOOST [1]. †IT MAY BE DUE TO THE FACT THAT WE HAVE NOT CAREFULLY FINE TUNED THE CROSS VALIDATION PARAMETER USED

data	Ours (Exp)		Ours (Log)		MultiBoost [1]		Speedup factor	
(# classes/# dims)	Error	Time	Error	Time	Error	Time	Exp	Log
iris (3/4)	6.0 (1.5)	0.47s	5.3 (1.8)	0.61s	6.7 (0.0)	8.6s	18	14
glass (6/9)	26.5 (5.5)	1.2s	25.8 (6.4)	1.67s	28.8 (6.7)	7.9s	7	5
usps (10/256)	7.8 (3.0)	2.3s	7.2 (4.0)	5.0s	9.4 (2.1)	26.5s	12	5
pen (10/16)	7.6 (1.7)	2.7s	8.0 (1.9)	5.6s	10.2 (3.1)	47.8s	18	9
news (20/771)	55.3 (3.1)	13.1s	55.8 (3.7)	20.3s	62.5 (3.0)	7m54s	36	23
letter (26/16)	22.8 (3.0)	35.7s	22.5 (3.2)	51.1s	23.8 (1.9)	44m4s	74	52
rcv1 (38/1213)	28.8 (0.6)	1m49s	29.8 (2.1)	2m59s	34.7 (3.0)	47m30m	26	16
sector (105/2959)	19.3 (1.3)	56m32s	20.2 (1.7)	1h53m	38.8 (2.0)†	16h37m	17	9

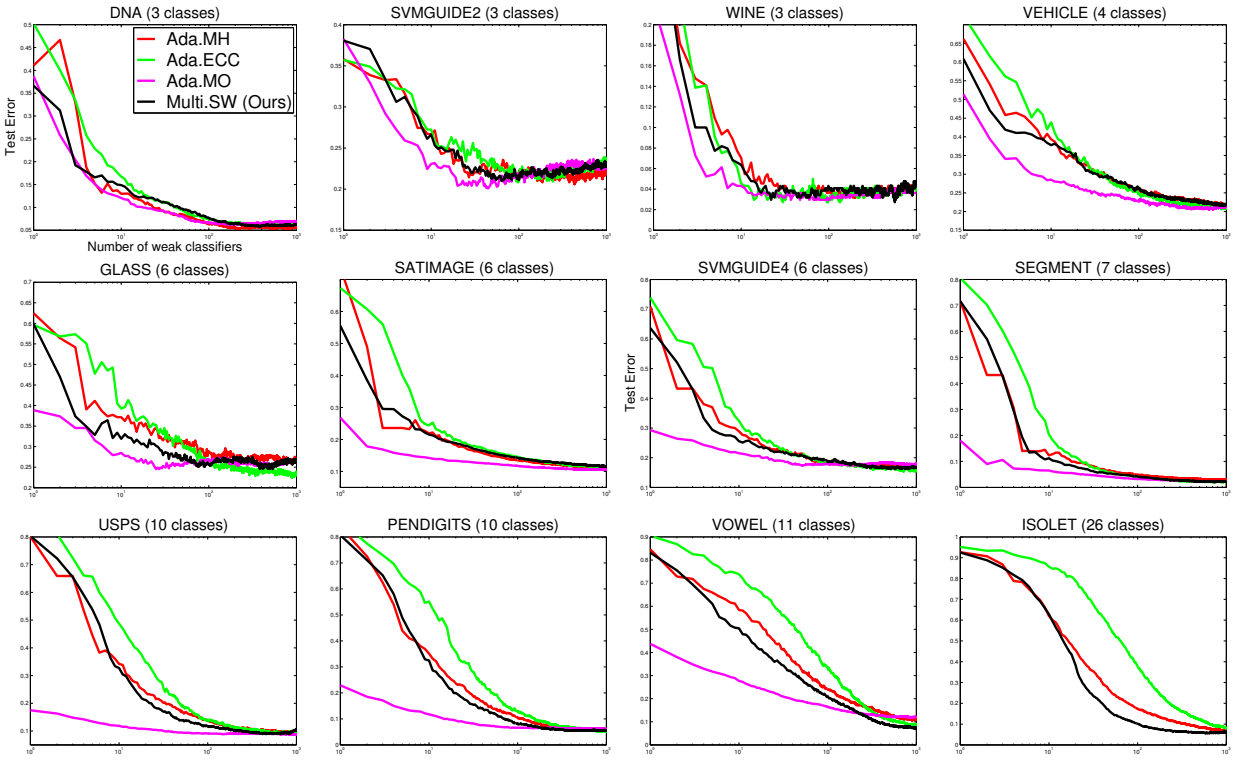

Fig. 3. Average test error vs. number of weak classifiers on multi-class UCI data sets. The vertical axis denotes the averaged test error rate and the horizontal axis denotes the number of weak classifiers. On UCI benchmark data sets, our approach performs comparable to other algorithms.

of code words from all 31 sub-windows. Hence, in total there are $6,200$ dimensional histogram. Fig. 4 shows the average classification errors. Based on our experimental results, our approaches have the best convergence rate and the lowest test error compared to other algorithms evaluated. We also apply a multi-class SVM to the above data set using the LIBSVM package [26] and report the recognition results in Table VII. SVM with $6,200$ features achieves an average accuracy of 76.30% (linear) and 81.47% (non-linear). Our results indicate that our proposed approach achieves a comparable accuracy to non-linear SVM while requiring less computation [7] (less number of features).

V. CONCLUSION AND FUTURE WORKS

In this paper, we focus primarily on the direct formulation of multi-class boosting methods. Unlike many existing multi-class boosting algorithms, which rely on error correcting code, we directly maximize the multi-class margin in a stage-wise manner. Reformulating the problem this way enables us to speed up the training time while maintaining the high classification performance. Various multi-class boosting algorithms are thoroughly evaluated on a multitude of multi-class data sets. Empirical results reveal that the new approach can speed up the classifier training time by more than two orders of magnitude without sacrificing detection accuracy. On visual object classification problems, e.g., MNIST and Scene-15, it is beneficial to apply our approach due to its fast convergence and better accuracy has been observed compared to coding-based approach.

[7]This is particularly important for many realtime applications.

REFERENCES

[1] C. Shen and Z. Hao, "A direct formulation for totally-corrective multi-class boosting," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2011.

[2] S. Knerr, L. Personnaz, and G. Dreyfus, "Handwritten digit recognition by neural networks with single-layer training," *IEEE Trans. Neural Networks.*, vol. 3, no. 6, pp. 962–968, 1992.

[3] G. Griffin, A. D. Holub, and P. Perona, "The caltech-256,," Technical Report 7694, California Institute of Technology, 2007.

TABLE V
TEST ERRORS OF DIFFERENT ALGORITHMS ON MULTI-CLASS UCI DATA SETS. ALL EXPERIMENTS ARE RUN 10 TIMES WITH 1000 BOOSTING ITERATIONS. † IT IS IMPOSSIBLE TO TRAIN $2^{25}-1$ WEAK CLASSIFIERS AT EACH BOOSTING ITERATION

Dataset	SAMME [23]	AdaBoost.ECC [10]	AdaBoost.MH [9]	AdaBoost.MO [9]	MCBoost$^{\exp}_{\mathrm{SW}}$	MCBoost$^{\log}_{\mathrm{SW}}$
dna	5.7 (0.7)	6.8 (0.9)	**5.6 (1.2)**	6.9 (1.2)	6.2 (1.0)	6.3 (1.0)
svmguide2	22.4 (4.1)	23.2 (3.7)	**21.7 (3.3)**	22.9 (4.3)	23.1 (3.5)	22.6 (3.8)
wine	5.0 (4.3)	3.9 (3.0)	4.3 (3.8)	**3.6 (3.7)**	4.5 (4.0)	4.1 (2.6)
vehicle	30.6 (2.4)	21.0 (3.6)	21.6 (3.4)	21.3 (3.0)	22.0 (3.9)	**20.7 (3.1)**
glass	31.3 (5.6)	23.0 (3.8)	27.0 (3.6)	26.2 (6.8)	27.0 (7.5)	25.7 (6.1)
satimage	18.0 (1.1)	11.5 (0.7)	11.1 (1.1)	**10.7 (1.0)**	11.6 (0.8)	11.4 (1.0)
svmguide4	25.2 (4.9)	**15.9 (2.7)**	17.5 (2.5)	17.9 (2.3)	16.5 (3.3)	16.7 (3.7)
segment	12.6 (5.5)	**2.1 (0.5)**	3.0 (0.5)	2.3 (0.5)	**2.1 (0.3)**	2.2 (0.5)
usps	14.3 (2.7)	9.2 (2.1)	9.2 (1.7)	**8.8 (2.5)**	10.2 (1.8)	9.4 (2.2)
pendigits	0.17 (2.8)	**5.2 (0.8)**	5.8 (0.9)	6.3 (1.4)	5.6 (1.5)	5.6 (1.5)
vowel	40.0 (1.9)	8.7 (2.5)	11.2 (2.3)	12.1 (3.0)	7.2 (1.5)	**7.1 (1.7)**
isolet	31.3 (1.5)	8.1 (1.0)	6.6 (1.2)	†	6.3 (0.9)	**5.6 (1.0)**

TABLE VI
TEST ERRORS RATE (%) ON THE MNIST DATA SET

	# features	Accuracy (%)
AdaBoost.MH [9]	1000	1.29
Ada.ECC [10]	1000	1.24
MCBoost$^{\exp}_{\mathrm{SW}}$	1000	76.3 (1.14)
MCBoost$^{\log}_{\mathrm{SW}}$	1000	**79.3 (1.03)**

TABLE VII
RECOGNITION RATE OF VARIOUS ALGORITHMS ON SCENE-15 DATA SETS. ALL EXPERIMENTS ARE RUN 5 TIMES. THE AVERAGE ACCURACY MEAN AND STANDARD DEVIATION (IN PERCENTAGE) ARE REPORTED

	# features	Accuracy (%)
Ada.SIP [11]	1000	75.7 (0.10)
Ada.ECC [10]	1000	76.5 (0.67)
AdaBoost.MH [9]	1000	77.6 (0.59)
Linear SVM	6200	76.3 (0.88)
MCBoost$^{\exp}_{\mathrm{SW}}$	**1000**	**79.3 (0.47)**
Non-linear SVM	**6200**	**81.4 (0.60)**

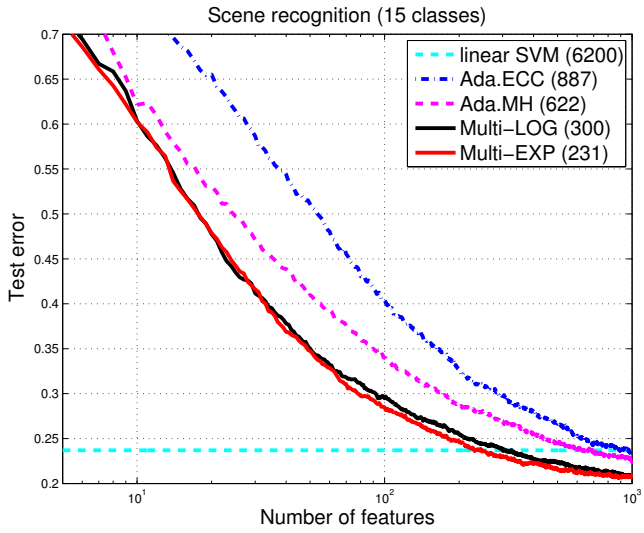

Fig. 4. Performance of different classifiers on the scene recognition data set. We also report the number of features required to achieve similar results to linear multi-class SVM. Both of our methods outperform other evaluated boosting algorithms.

[4] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2006.

[5] R. Schapire, "Theoretical views of boosting and applications," in *Proc. Int. Conf. Algorithmic Learn. Theory*, London, UK, 1999, pp. 13–25.

[6] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector mchines," *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, 2002.

[7] C. Shen and H. Li, "Boosting through optimization of margin distributions," *IEEE Trans. Neural Networks.*, vol. 21, no. 4, pp. 659 – 666, 2010.

[8] C. Shen, H. Li, and N. Barnes, "Totally corrective boosting for regularized risk minimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011.

[9] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated prediction," *Mach. Learn.*, vol. 37, no. 3, pp. 297336, 1999.

[10] V. Guruswami and A. Sahai, "Multiclass learning, boosting, and error correcting codes," in *Annual Conf. Learn. Theory*, 1999, pp. 145–155.

[11] B. Zhang, G. Ye, Y. Wang, J. Xu, and G. Herman, "Finding shareable informative patterns and optimal coding matrix for multiclass boosting," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2009.

[12] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 5, pp. 854–869, 2007.

[13] R. Schapire, "Using output codes to boost multiclass learning problems," in *Proc. Int. Conf. Mach. Learn.*, 1997.

[14] L. Li, "Multiclass boosting with repartitioning," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 569–576.

[15] J. Duchi and Y. Singer, "Boosting with structural sparsity," in *Proc. Int. Conf. Mach. Learn.*, 2009.

[16] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 2000.

[17] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Mach. Learn.*, vol. 46, no. 1-3, pp. 225–254, 2002.

[18] C. Shen and H. Li, "On the dual formulation of boosting algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 12, pp. 2216–2231, 2010.

[19] S. Rosset, J. Zhu, and T. Hastie, "Boosting as a regularized path to a maximum margin classifier," *J. Mach. Learn. Res.*, vol. 5, pp. 941–973, 2004.

[20] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," Apr. 2011.

[21] T. Zhang and B. Yu, "Boosting with early stopping: Convergence and consistency," *Ann. Statist.*, vol. 33, no. 4, pp. 1538–1579, 2005.

[22] P. Carbonetto, "A MATLAB interface for L-BFGS-B," Department of Computer Science, University of British Columbia., 2007.

[23] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Statistics & Its Interface*, vol. 2, 2009.

[24] J. Wu and J. M. Rehg, "CENTRIST: A visual descriptor for scene categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1489–1501, 2011.

[25] A. Bosch, A. Zisserman, and X. Munoz, "Scene classification using a hybrid generative/discriminative approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 4, pp. 712 – 727, 2008.

[26] C-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Sys. & Tech.*, vol. 2, no. 3, 2011.