Multiclass Boosting with Repartitioning

Ling Li LING@CALTECH.EDU
Learning Systems Group, California Institute of Technology, Pasadena, CA 91125, USA

Abstract

A multiclass classification problem can be reduced to a collection of binary problems with the aid of a coding matrix. The quality of the final solution, which is an ensemble of base classifiers learned on the binary problems, is affected by both the performance of the base learner and the error-correcting ability of the coding matrix. A coding matrix with strong error-correcting ability may not be overall optimal if the binary problems are too hard for the base learner. Thus a trade-off between error-correcting and base learning should be sought. In this paper, we propose a new multiclass boosting algorithm that modifies the coding matrix according to the learning ability of the base learner. We show experimentally that our algorithm is very efficient in optimizing the multiclass margin cost, and outperforms existing multiclass algorithms such as AdaBoost.ECC and one-vs-one. The improvement is especially significant when the base learner is not very powerful.

1. Introduction

Many efforts of the machine learning research have been focused on binary classification problems. For a multiclass classification problem with more than two different class labels, it is possible to reformulate it as a collection of binary problems. The most popular approaches are one-vs-all where each class is compared against all others, and one-vs-one where all pairs of classes are compared (Allwein et al., 2000).

Dietterich and Bakiri (1995) and Allwein et al. (2000) unified and generalized most such approaches with error-correcting codes. In their framework, an error-correcting coding matrix is first given, with each row associated with a class from the multiclass problem. Binary classifiers (also called base classifiers) are then learned, one for each column of the matrix, on training examples that are relabeled according to the column. Given an unseen input, the vector formed by the outputs of the base classifiers is compared with every row of the coding matrix, and the class associated with the "closest" row is predicted as the class of the input.

The coding matrix is usually chosen for strong error-correcting ability (Dietterich & Bakiri, 1995). However, strong error-correcting ability alone does not guarantee good learning performance—one important assumption for normal error-correcting codes that errors are uncorrelated may not hold for the base classifiers (Guruswami & Sahai, 1999). Thus the choice of the coding matrix has to balance the needs of strong error-correction and uncorrelated classifier errors, and is usually problem-dependent (Allwein et al., 2000).

Multiclass boosting algorithms based on error-correcting codes (Schapire, 1997; Guruswami & Sahai, 1999) tackle the error correlation among the base classifiers by deliberately reweighting the training examples. They usually start off with an empty coding matrix and all classes indistinguishable from others, and then iteratively append columns to the matrix and train base classifiers so that the confusion between classes can be gradually reduced. The examples are reweighted in a fashion similar to the weighting scheme in the binary AdaBoost (Freund & Schapire, 1996), aiming at uncorrelated errors. In order to reduce the confusion between classes as fast as possible, in each iteration, a max-cut problem can be solved so that the "optimal" matrix column is obtained.

It is however common that researchers usually do not pursue the "optimal" coding matrix when applying the multiclass boosting algorithms. Instead, some choose the matrix columns at random (Schapire, 1997; Guruswami & Sahai, 1999).[1] Although the fact of max-cut being NP-complete prevents an efficient solution, this is not exactly the reason for researchers not using it; after all, many multiclass classification prob-

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

[1]We actually did not find out how Guruswami and Sahai (1999) chose the columns.

lems have less than ten classes, and even some simple heuristic methods can do better in reducing the confusion than a random method. It is mostly because that, combined with the boosting algorithm, a max-cut or heuristic method does not improve over a random one (Schapire, 1997).

In this paper, we discuss why max-cut does not work well with existing multiclass boosting algorithms, and propose a general remedy which leads to a new boosting algorithm. We first discuss in Section 2 how AdaBoost.ECC, a typical multiclass boosting algorithm, can be explained as gradient descent on a margin cost function (Sun et al., 2005). The trade-off between the error-correcting ability and the base learning performance is then explained. We propose in Section 3 the new algorithm to achieve a better trade-off by modifying the coding matrix according to the learning ability of the base learner. In Section 4, our algorithm is tested on real-world data sets with four base learners of various degrees of complexity, and the results are quite promising. Finally we conclude in Section 5.

2. AdaBoost.ECC and Multiclass Cost

Consider a K-class classification problem where the class labels are 1, 2, …, K. The training set contains N examples, $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, where $\mathbf{x}_n$ is the input and $y_n \in \{1, 2, \ldots, K\}$. To reduce the multiclass problem to a collection of T binary problems, we use a coding matrix $\mathbf{M} \in \{-1, 0, +1\}^{K \times T}$ (Allwein et al., 2000). A base classifier f_t is learned on the relabeled examples $\{(\mathbf{x}_n, \mathbf{M}(y_n, t)) : \mathbf{M}(y_n, t) \neq 0\}$ based on the t-th column of $\mathbf{M}$, and classes that are relabeled as 0 are omitted. The columns of $\mathbf{M}$ are also called partitions (or partial partitions if there are 0's) since they define the way the original examples are split.

Given an input $\mathbf{x}$, the ensemble output $\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_T(\mathbf{x}))$ is computed, and the Hamming decoding[2] (Allwein et al., 2000) is used to predict the label of $\mathbf{x}$. In the most general settings, there is a coefficient α_t for every base classifier f_t. The Hamming distance between $\mathbf{F}(\mathbf{x})$ and the k-th row $\mathbf{M}(k)$ is

$$\Delta\left(\mathbf{M}(k), \mathbf{F}(\mathbf{x})\right) = \sum_{t=1}^{T} \alpha_t \frac{1 - \mathbf{M}(k,t) f_t(\mathbf{x})}{2}.$$

Label y is predicted if $\mathbf{M}(y)$ has the smallest Hamming distance to $\mathbf{F}(\mathbf{x})$.

To correctly classify an example $(\mathbf{x}, y)$, we want $\Delta\left(\mathbf{M}(y), \mathbf{F}(\mathbf{x})\right)$ to be smaller than $\Delta\left(\mathbf{M}(k), \mathbf{F}(\mathbf{x})\right)$ for any $k \neq y$. Naturally, we may define the margin of the example $(\mathbf{x}, y)$ for class k as the difference between these two distances,

$$\rho_k\left(\mathbf{x}, y\right) = \Delta\left(\mathbf{M}(k), \mathbf{F}(\mathbf{x})\right) - \Delta\left(\mathbf{M}(y), \mathbf{F}(\mathbf{x})\right). \quad (1)$$

A learning algorithm should pick a coding matrix $\mathbf{M}$, T base classifiers f_t's, and their coefficients α_t's, such that the margins of the training examples are as large as possible.

Algorithm 1. AdaBoost.ECC (Guruswami & Sahai, 1999)

Input: A training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$; number of epochs T
Initialize $\tilde{D}_1(n,k) = 1$; $\mathbf{F} = (0, 0, \ldots, 0)$, i.e., $f_t = 0$
for $t = 1$ to T **do**
  Choose the t-th column $\mathbf{M}(\cdot, t) \in \{-1, +1\}^K$
  $U_t = \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) [\![\mathbf{M}(k,t) \neq \mathbf{M}(y_n,t)]\!]$
  $D_t(n) = U_t^{-1} \cdot \sum_{k=1}^{K} \tilde{D}_t(n,k) [\![\mathbf{M}(k,t) \neq \mathbf{M}(y_n,t)]\!]$
  Train f_t on $\{(\mathbf{x}_n, \mathbf{M}(y_n, t))\}$ with distribution D_t
  $\varepsilon_t = \sum_{n=1}^{N} D_t(n) [\![f_t(\mathbf{x}_n) \neq \mathbf{M}(y_n,t)]\!]$
  $\alpha_t = \frac{1}{2} \ln(\varepsilon_t^{-1} - 1)$
  $\tilde{D}_{t+1}(n,k) = \tilde{D}_t(n,k) \cdot e^{-\frac{\alpha_t}{2}[\mathbf{M}(y_n,t) - \mathbf{M}(k,t)] f_t(\mathbf{x}_n)}$
end for
return the coding matrix $\mathbf{M}$, the ensemble $\mathbf{F}$ and α_t

AdaBoost.ECC (Guruswami & Sahai, 1999) is one such algorithm with a boosting style (Algorithm 1).[3] It starts from an empty coding matrix, and iteratively generates columns and base classifiers. Just as AdaBoost (Freund & Schapire, 1996) optimizes some cost as gradient descent in the function space (Mason et al., 2000), AdaBoost.ECC optimizes an exponential cost function based on the margins (Sun et al., 2005)[4]

$$C(\mathbf{F}) = \sum_{n=1}^{N} \sum_{k \neq y_n} e^{-\rho_k(\mathbf{x}_n, y_n)}. \quad (2)$$

We will briefly show how AdaBoost.ECC optimizes this cost in the t-th iteration. Using the definitions in Algorithm 1, we notice that by induction, $\mathbf{F} = (f_1, \ldots, f_t, 0, \ldots)$ and $\tilde{D}_{t+1}(n,k) = e^{-\rho_k(\mathbf{x}_n, y_n)}$. So $C(\mathbf{F}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_{t+1}(n,k) - N$ for $\tilde{D}_{t+1}(n, y_n)$ is always 1. The negative gradient at $\alpha_t = 0$ is thus

$$-\left.\frac{\partial C\left(\mathbf{F}\right)}{\partial \alpha_t}\right|_{\alpha_t=0} = -\sum_{n=1}^{N} \sum_{k=1}^{K} \left.\frac{\partial \tilde{D}_{t+1}(n,k)}{\partial \alpha_t}\right|_{\alpha_t=0}$$
$$= \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) \left[\frac{\mathbf{M}(y_n,t) - \mathbf{M}(k,t)}{2}\right] f_t(\mathbf{x}_n). \quad (3)$$

[2]We consider base classifiers with outputs in $\{-1, +1\}$ (see experiment settings in Section 4). Thus a loss-based decoding is equivalent to the Hamming decoding.

[3]We only discuss the symmetric AdaBoost.ECC in this paper; nevertheless, our improvement can also be against the asymmetric AdaBoost.ECC.

[4]Although Sun et al. (2005) used different definitions for the ensemble output and the distance measure, their cost function is equivalent to ours.

The last equality is due to step 9 in Algorithm 1. Since $\mathbf{M}(k,t)$ in AdaBoost.ECC can only be -1 or $+1$, the negative gradient can further be reduced to

$$U_t \sum_{n=1}^{N} D_t(n)\mathbf{M}(y_n,t)f_t(\mathbf{x}_n) = U_t(1-2\varepsilon_t). \qquad (4)$$

AdaBoost.ECC tries to maximize this negative gradient and then picks α_t to exactly minimize the cost along the negative gradient.

Two steps in Algorithm 1 directly affect the maximization of the negative gradient (4). One is step 3 where the t-th column is picked. The t-th column decides the value of U_t, which indicates the error-correcting ability of the column. Roughly speaking, the larger U_t is, the stronger the error-correcting ability is, the faster the cost is reduced, and the smaller the training error bound is (Guruswami & Sahai, 1999, Theorem 2). The other is step 6, where the base classifier is learned. It is also obvious that both the cost and the training error bound can be smaller if the base learner can achieve a smaller ε_t. It seems that in order for a better cost optimization, we should both maximize U_t and minimize ε_t.

A max-cut method has been proposed to obtain the "optimal" partition that maximizes U_t (Schapire, 1997). However, it appears that researchers prefer a somewhat random method for picking the partitions, e.g., rand-half that randomly picks half of the classes for label -1 and the other half for $+1$ (Schapire, 1997; Sun et al., 2005). This is actually with a reason: in long run, using the "optimal" partitions from max-cut is usually worse than using the random partitions, in both training and testing.

Let's look at a toy problem where points in a rectangle are assorted into seven tangram pieces (Figure 1). To compare the two column-picking methods, rand-half and max-cut, we ran AdaBoost.ECC on 500 random examples. Our base classifiers are perceptrons, which separate points with a straight line. It turned out that rand-half was more efficient in reducing the cost (Figure 2). And as a matter of fact, the test error in this experiment was also smaller with rand-half.

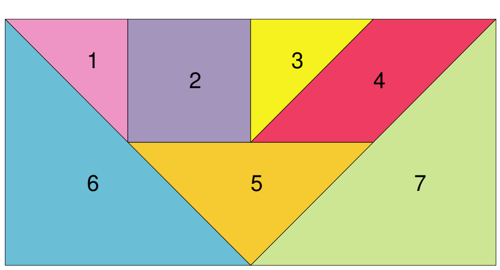


Figure 1. The tangram with seven pieces

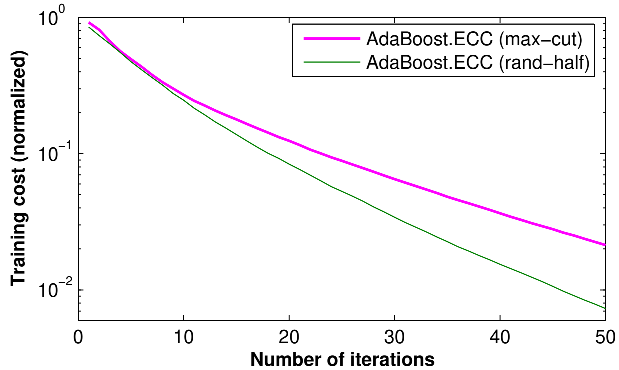
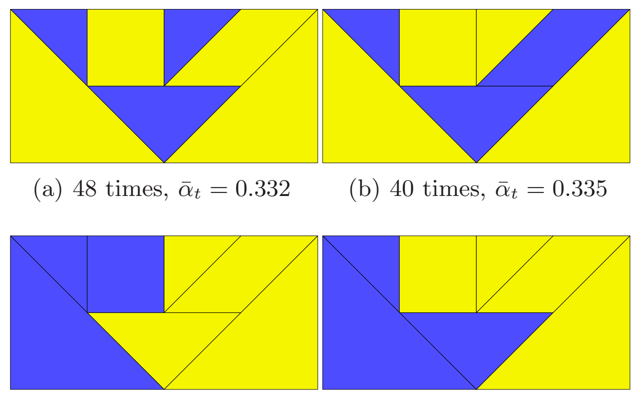


Figure 2. AdaBoost.ECC cost in the tangram experiment (normalized by $N(K-1)$)



(a) 48 times, $\bar{\alpha}_t = 0.332$ (b) 40 times, $\bar{\alpha}_t = 0.335$

(c) 9 times, $\bar{\alpha}_t = 0.813$ (d) 11 times, $\bar{\alpha}_t = 0.619$

Figure 3. Dominating partitions in the tangram experiment: (a,b) with max-cut; (c,d) with rand-half

Why did max-cut, which maximized U_t in every iteration, have a worse performance in optimizing the cost? One probable reason is that the binary problems from max-cut are usually much "harder" for the base learner. To see this in the tangram experiment, we counted how many times a partition was picked during the AdaBoost.ECC runs, and summed up for this partition the coefficients α_t, which were decided from the weighted error ε_t of the base classifiers trained on the partition. The sum indicates how much the partition influences the ensemble, and the average coefficient (denoted as $\bar{\alpha}$) implies how hard the binary problems are to the base learner. Figure 3 gives the two dominating partitions with the largest coefficient sums out of the 200 AdaBoost.ECC iterations. Obviously AdaBoost.ECC with max-cut focused on harder binary problems, while AdaBoost.ECC with rand-half was happy with easier problems. Since harder problems deteriorated the learning of base classifiers, the overall cost reduction was worse for max-cut. Note that this situation might be more prominent for later iterations since the boosting nature of AdaBoost.ECC keeps increasing the hardness of the binary problems.

It is thus important to find a good trade-off between maximizing U_t and minimizing ε_t. In next section, we will discuss a remedy based on repartitioning.

3. AdaBoost.ECC with Repartitioning

We have seen from the tangram experiment that different partitioning methods may generate binary problems of various hardness levels. How hard a problem is depends on how the relabeled examples distribute in the feature space and how well the base learner can handle such a distribution. For example, with perceptrons as the base classifiers, discriminating tangram classes 1 and 3 from 2 and 4 (Figure 3(a)) is much harder than discriminating 1 and 2 from 3 and 4 (Figure 3(c)). Thus in order to achieve a good trade-off between maximizing U_t and minimizing ε_t, we should also consider the discriminating ability of the base learner when picking the partitions.

How do we know whether a partition can be well handled by the base learner? We usually do not know unless the base learner has been tried on the partition. The learned classifier has its own preference on how the examples should be relabeled, and thus hints on what partitions better suit the base learner. We can then repartition the examples based on such information so as to reduce the cost even more.

Assume in the t-th iteration, a base classifier f_t has been learned. To find a new and better partition for this f_t, we try to maximize the negative gradient (3),

$$\max_{\mathbf{M}(\cdot,t)} \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) \left[\mathbf{M}(y_n,t) - \mathbf{M}(k,t)\right] f_t(\mathbf{x}_n),$$

which can be reorganized as

$$\max_{\mathbf{M}(\cdot,t)} \sum_{k=1}^{K} \boldsymbol{\mu}(k,t)\mathbf{M}(k,t),$$

with $\boldsymbol{\mu}(k,t)$ defined as $\boldsymbol{\mu}(k,t) =$

$$\sum_{n:\, y_n=k} \sum_{\ell=1}^{K} \tilde{D}_t(n,\ell) f_t(\mathbf{x}_n) - \sum_{n=1}^{N} \tilde{D}_t(n,k) f_t(\mathbf{x}_n). \quad (5)$$

Since $\mathbf{M}(k,t) \in \{-1,+1\}$, it is clear that the negative gradient is maximized when $\mathbf{M}(k,t) = \operatorname{sign}[\boldsymbol{\mu}(k,t)]$.

The repartitioning can also be justified intuitively from a single example point of view. On one side, the contribution of example $(\mathbf{x}_n, y_n)$ to $\mathbf{M}(y_n,t) = \operatorname{sign}[\boldsymbol{\mu}(y_n,t)]$ is

$$\left[\sum_{\ell=1}^{K} \tilde{D}_t(n,\ell) - \tilde{D}_t(n,y_n)\right] f_t(\mathbf{x}_n).$$

Note that with $\mathbf{F} = (f_1, \ldots, f_{t-1}, 0, \ldots)$, $\tilde{D}_t(i,k)$ is $e^{-\rho_k(\mathbf{x}_n,y_n)}$. So the summation $\sum_{\ell \neq y_n} \tilde{D}_t(n,\ell)$ actually tells, without the current f_t, how close the example is to classes other than its own class y_n. The closer it is to other classes, the larger the summation is, and thus the more likely $\mathbf{M}(y_n,t)$ would be to have the same sign as $f_t(\mathbf{x}_i)$, which would in consequence increase some of the margins of this example after f_t is included. On the other side, the contribution of the example to $\mathbf{M}(k,t)$ where $k \neq y_n$ is $-\tilde{D}_t(n,k) f_t(\mathbf{x}_n)$. With similar reasoning, this implies that if the example is close to class k, $\mathbf{M}(k,t)$ would be requested to have the opposite sign as $f_t(\mathbf{x}_n)$, which also would increase the margin ρ_k.

The repartitioning of $\mathbf{M}(\cdot,t)$ and the learning of f_t can be carried out alternatively. For example, we can start from a partition, train a base classifier on it, repartition the classes, and then train a new base classifier on the new partition. If the base learner always minimizes the weighted training error, the negative gradient would always increase until convergence. In practice, when the base learning is expensive, we may only repeat the repartitioning and learning cycle for several fixed steps.

Algorithm 2 depicts the new multiclass boosting algorithm, AdaBoost.ERP, i.e., AdaBoost.ECC with repartitioning. The changes from AdaBoost.ECC are underlined for better reading. Note that we also allow

Algorithm 2. AdaBoost.ERP (ECC with repartitioning)

Input: A training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$; number of epochs T
- Initialize $\tilde{D}_1(n,k) = 1$; $\mathbf{F} = (0,0,\ldots,0)$, i.e., $f_t = 0$
- **for** $t = 1$ to T **do**
  - Choose an initial column $\mathbf{M}(\cdot,t) \in \{\underline{-1,0,+1}\}^K$
  - **repeat** {Alternate learning and re-partitioning}
    - $U_t = \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{D}_t(n,k) \underline{[\![\mathbf{M}(k,t)\mathbf{M}(y_n,t) < 0]\!]}$
    - $D_t(n) = U_t^{-1} \cdot \sum_k \tilde{D}_t(n,k) \underline{[\![\mathbf{M}(k,t)\mathbf{M}(y_n,t) < 0]\!]}$
    - Train f_t on $\{(\mathbf{x}_n, \mathbf{M}(y_n,t))\}$ with distribution D_t
    - $\underline{\mathbf{M}(k,t) = \operatorname{sign}[\boldsymbol{\mu}(k,t)]}$ {See (5) for details}
  - **until** convergence or some specified steps
  - Update U_t and D_t with the current $\mathbf{M}(\cdot,t)$, as above
  - $\varepsilon_t = \sum_{n=1}^{N} D_t(n) [\![f_t(\mathbf{x}_n) \neq \mathbf{M}(y_n,t)]\!]$
  - $\alpha_t = \frac{1}{2}\ln(\varepsilon_t^{-1} - 1)$
  - $\tilde{D}_{t+1}(n,k) = \tilde{D}_t(n,k) \cdot e^{-\frac{\alpha_t}{2}[\mathbf{M}(y_n,t) - \mathbf{M}(k,t)] f_t(\mathbf{x}_n)}$
- **end for**
- **return** the coding matrix $\mathbf{M}$, the ensemble $\mathbf{F}$ and α_t

the initial column $\mathbf{M}(\cdot, t)$ to have 0's (step 3). Since a partial partition will be adjusted in the repartitioning step to a full one, the coefficient α_t can still be decided exactly. The benefit of having a partial partition is that only part of the examples are used for the initial base learning (step 7). This allows, for example, to first focus the learning on local structures of just a pair of classes and then extend to the full partition based on the knowledge learned from the local structures. Besides, the base learning is also faster with less examples.

The repartitioning takes $2NK$ arithmetic operations, which is usually much cheaper than the base learning.

4. Experiments

We tested AdaBoost.ERP experimentally on ten multiclass benchmark problems (Table 1) from the UCI machine learning repository (Hettich et al., 1998) and the StatLog project (Michie et al., 1994). For problems with both training and test sets, experiments were run 100 times and the results were averaged. Otherwise, a 10-fold cross-validation was repeated 10 times for a total of 100 runs. When there was randomness in the learning algorithm and/or cross-validation was used, the standard error over 100 runs was also computed. For each run, the training part of the examples were linearly scaled to $[-1, 1]$, and then the test examples were adjusted accordingly.

We tried different ways to set the initial partitions and different schedules to repartition. In the results reported here, the initial partial partitions always contained two classes selected from all the K classes, randomly (denoted by rand-2) or to maximize the corresponding U_t (denoted by max-2). We use a string of "L" and "R" to represent the schedule of base learning and repartitioning in a boosting iteration. For example, "LRL" means that a base classifier was first learned on the two classes in the partial partition, then the partition was adjusted, and finally a new base classifier was trained on the adjusted full partition.

Table 1. Multiclass problems

dataset	#train	#test	K	#attribute
dna	2000	1186	3	180
glass	214	-	6	9
iris	150	-	3	4
letter	16000	4000	26	16
pendigits	7494	3498	10	16
satimage	4435	2000	6	36
segment	2310	-	7	18
vehicle	846	-	4	18
vowel	528	462	11	10
wine	178	-	3	13

We used four base learners of various degrees of complexity. The first one is the decision stump, also known as FindAttrTest (Schapire, 1997). The second one is the perceptron with a learning algorithm suitable for boosting (Li, 2005). The third one is a binary AdaBoost (Freund & Schapire, 1996) that aggregates up to 50 decision stumps. The last one is the soft-margin support vector machine with the perceptron kernel (SVM-perceptron) (Lin & Li, 2005).[5]

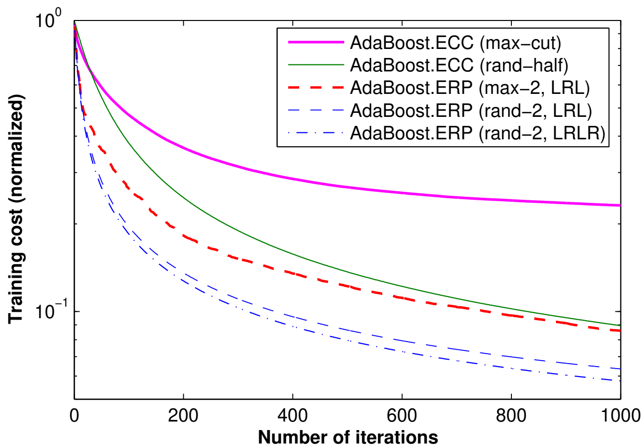
We compared our algorithm with AdaBoost.ECC with max-cut or rand-half. When the decision stump was used as the base learner, each algorithm was run for 500 iterations; for other more powerful base learners, the number of iterations was 200. However, for one exception, the letter data with 26 classes, we ran 1000 iterations with the decision stump and 500 iterations with other base learners. Note also that the exact max-cut for 26 classes is time-consuming so instead we used a simple greedy approximation for the letter data to approximately maximize U_t for AdaBoost.ECC. We also compared with one-vs-one and one-vs-all using the same base learners. For space consideration, we only list the lower test errors of these two algorithms.

Table 2 presents the test errors with the decision stump as the base learner, the lowest errors in bold. With this simple base learner, one-vs-one and one-vs-all got quite large errors since they are limited in the number of base classifiers. We can also see that most of the time AdaBoost.ECC with max-cut was worse than AdaBoost.ECC with rand-half. This verified our analysis that, when the base learner is not powerful enough, problems from max-cut would be too hard and the overall learning performance would instead be deteriorated (see also Figure 4). With the help of repartitioning, AdaBoost.ERP achieved better test errors for most of the data sets, and for some cases it was substantially better. For better illustration, we also show in Figure 4 the training cost and the test error curves for two large data sets, letter and pendigits. With the same number of base classifiers, AdaBoost.ERP almost always achieved a much lower training cost and a lower test error. More steps of the repartitioning and base learning further improved the learning, although the marginal improvement was small.

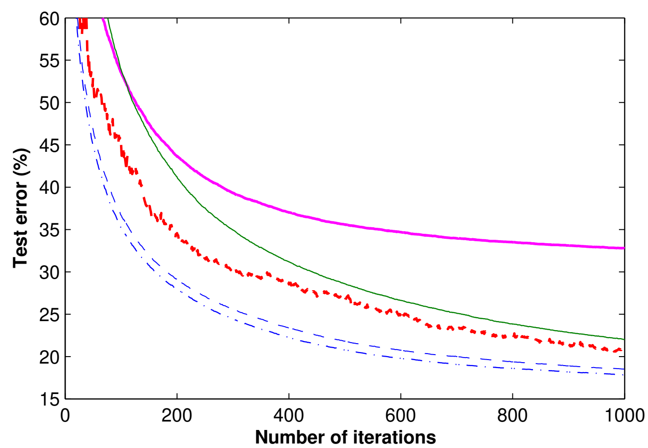
[5]For the perceptron kernel, only the regularization parameter C needs to be tuned. For problems with both training and test sets, a cross-validation with 30% of the training set kept for validation was repeated 10 times. The best $C \in \{2^{-3}, 1, 2^3, 2^6, 2^9\}$ was then used in the full training and testing. The whole process was repeated 20 time. For problems with no test sets, the best results of the 10-fold cross-validation averaged over 10 times were reported. To support the weighted data, we scale C for each example proportional to its sample weight (Vapnik, 1999).

Table 2. Test errors (%) of multiclass algorithms with the decision stump as the base learner

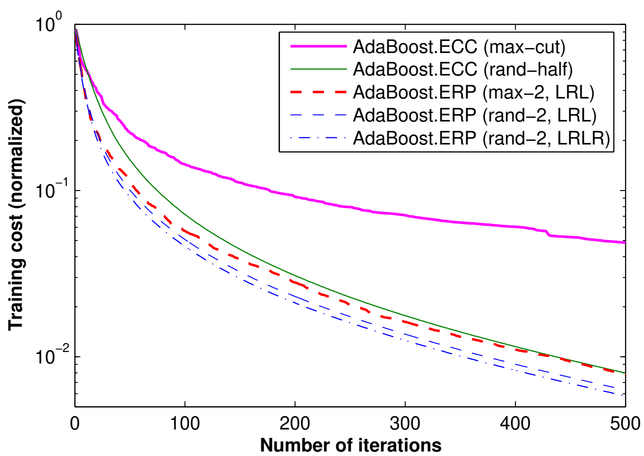
dataset	one-vs-one one-vs-all	AdaBoost.ECC max-cut	AdaBoost.ECC rand-half	AdaBoost.ERP (max-2) LRL	AdaBoost.ERP (max-2) LRLR	AdaBoost.ERP (rand-2) LRL	AdaBoost.ERP (rand-2) LRLR
dna	30.61	5.90	5.92 ± 0.02	6.41	**5.56**	5.78 ± 0.03	5.88 ± 0.03
glass	34.10 ± 1.11	27.43 ± 0.95	26.67 ± 0.92	26.05 ± 0.85	25.57 ± 0.89	**25.29 ± 0.85**	25.62 ± 0.85
iris	7.60 ± 0.55	7.67 ± 0.61	**6.60 ± 0.60**	6.73 ± 0.59	6.80 ± 0.59	7.53 ± 1.10	**6.60 ± 0.59**
letter	39.42	32.79 ± 0.19	22.00 ± 0.04	21.05	**17.73**	18.52 ± 0.03	17.84 ± 0.02
pendigits	23.67	9.06	5.94 ± 0.02	6.03	5.80	5.65 ± 0.03	**5.55 ± 0.02**
satimage	19.15	14.50	12.57 ± 0.04	**12.10**	12.45	12.59 ± 0.04	12.58 ± 0.04
segment	12.24 ± 0.21	3.28 ± 0.12	1.94 ± 0.09	2.07 ± 0.09	1.97 ± 0.09	**1.90 ± 0.09**	1.95 ± 0.09
vehicle	43.31 ± 0.48	26.93 ± 0.40	22.13 ± 0.38	23.28 ± 0.38	22.85 ± 0.39	**22.08 ± 0.39**	22.40 ± 0.41
vowel	57.14	59.74	57.98 ± 0.16	**55.63**	59.09	57.40 ± 0.15	57.65 ± 0.13
wine	15.33 ± 0.71	**2.00 ± 0.32**	3.17 ± 0.39	2.33 ± 0.36	2.72 ± 0.39	2.83 ± 0.37	2.78 ± 0.37



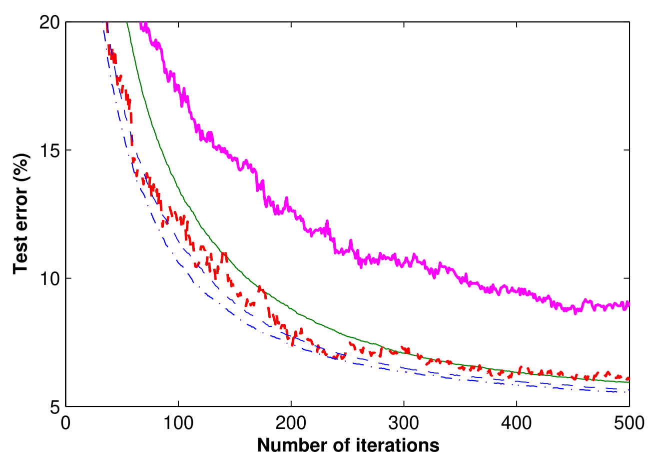
(a) cost on the letter data



(b) test error on the letter data



(c) cost on the pendigits data



(d) test error on the pendigits data

Figure 4. Multiclass boosting with the decision stump (AdaBoost.ERP (max-2, LRLR) is very close to that with rand-2)

With the perceptron as a more powerful base learner, test errors on some data sets were greatly reduced (Tables 3, with less number of iterations compared to that with the decision stump). Again repartitioning improved the learning performance on most of the data sets. Figure 5 shows the training cost and the test error curves for the letter data set. Observations are similar to those of Figure 4, but the improvement was not as dramatic as with the decision stump.

The binary AdaBoost was the only weak learner with which one-vs-one actually had comparable or even bet-

Table 3. Test errors (%) of multiclass algorithms with the perceptron as the base learner

dataset	one-vs-one	one-vs-all	AdaBoost.ECC max-cut	AdaBoost.ECC rand-half	AdaBoost.ERP (max-2) LRL	AdaBoost.ERP (max-2) LRLR	AdaBoost.ERP (rand-2) LRL	AdaBoost.ERP (rand-2) LRLR
dna		25.97 ± 0.26	**8.08 ± 0.05**	8.21 ± 0.06	8.18 ± 0.06	8.17 ± 0.06	8.09 ± 0.07	8.11 ± 0.06
glass		35.57 ± 1.16	29.48 ± 0.94	30.00 ± 1.02	**28.38 ± 0.87**	29.43 ± 1.04	30.57 ± 1.05	29.81 ± 0.94
iris		4.93 ± 0.57	5.20 ± 0.56	4.40 ± 0.53	4.67 ± 0.51	4.47 ± 0.51	**4.33 ± 0.53**	4.60 ± 0.52
letter		22.17 ± 0.07	15.88 ± 0.05	14.66 ± 0.05	13.64 ± 0.05	11.61 ± 0.04	13.65 ± 0.05	**11.59 ± 0.05**
pendigits		7.09 ± 0.09	3.71 ± 0.03	3.72 ± 0.03	3.72 ± 0.02	**3.64 ± 0.03**	3.71 ± 0.02	3.68 ± 0.03
satimage		15.14 ± 0.06	12.84 ± 0.05	12.60 ± 0.05	**12.37 ± 0.05**	12.42 ± 0.05	12.58 ± 0.05	12.57 ± 0.05
segment		7.53 ± 0.18	2.80 ± 0.12	2.74 ± 0.11	2.81 ± 0.11	2.83 ± 0.11	2.74 ± 0.10	**2.60 ± 0.11**
vehicle		31.58 ± 0.49	22.22 ± 0.45	20.47 ± 0.42	20.39 ± 0.42	20.86 ± 0.43	20.88 ± 0.44	**20.34 ± 0.43**
vowel		56.19 ± 0.29	56.26 ± 0.28	51.61 ± 0.26	50.61 ± 0.22	50.97 ± 0.26	50.40 ± 0.26	**50.31 ± 0.25**
wine		3.22 ± 0.41	**2.06 ± 0.32**	2.67 ± 0.37	2.39 ± 0.37	2.33 ± 0.37	2.39 ± 0.36	2.56 ± 0.39

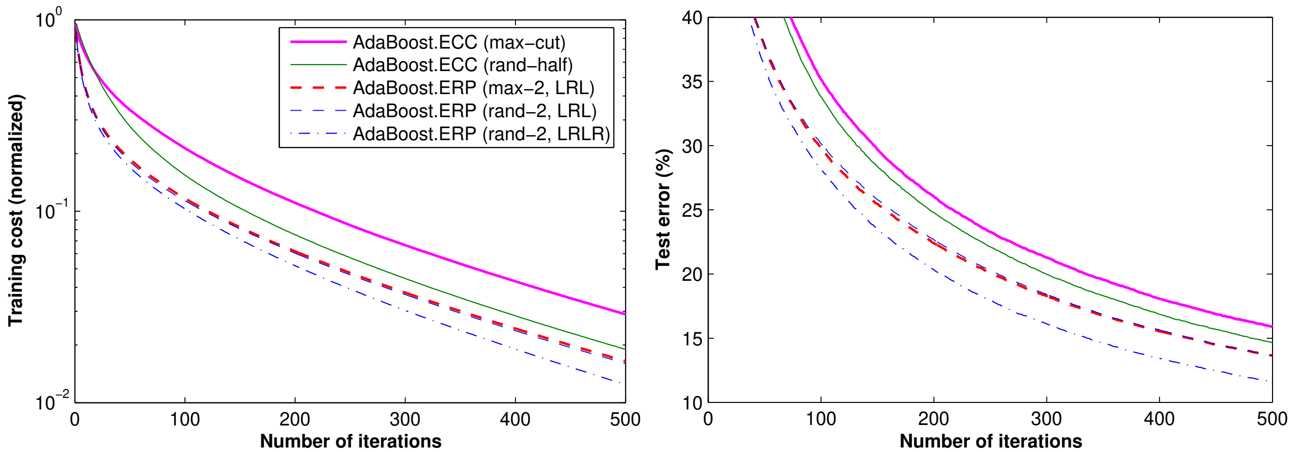


Figure 5. Multiclass boosting with the perceptron on the letter data

ter performance compared to the boosting algorithms. So we mark in Table 4 both the lowest errors among the boosting algorithms and the lowest errors among all the algorithms. Note that with this base learner, AdaBoost.ERP with only one base learning and one repartitioning ("LR") was already comparable to AdaBoost.ECC with base learning on the full training set.

SVM-perceptron brought us the overall lowest test errors for most of the data sets (Table 5). Note that we do not have all the results for dna and letter since parameter selection on an ensemble of SVMs is time-consuming. With this powerful base learner, all the multiclass algorithms performed comparably well, although AdaBoost.ERP was still better for some data sets. AdaBoost.ERP was also much faster compared to AdaBoost.ECC even though two SVMs may be learned in one iteration of AdaBoost.ERP, since the binary problems were usually easier.

5. Conclusion

We have proposed and tested AdaBoost.ERP, a new multiclass boosting algorithm with error-correcting codes and repartitioning. The repartitioning is meant to find a better coding matrix according to the learning ability of the base learner. Our experimental results have shown that, compared with AdaBoost.ECC, one-vs-one, and one-vs-all, AdaBoost.ERP achieved the lowest training cost and test error on most of the real-world data sets we used. The improvement can be especially significant when the base learner is not very powerful. AdaBoost.ERP was also faster than AdaBoost.ECC when working with SVM-perceptron.

Simple algorithms like one-vs-one have their advantages. Compared to boosting algorithms, their training time is usually much less, and the test error can be comparable or even lower when powerful base learners are used. The test time can also be substantially reduced (Platt et al., 2000). Thus it is interesting to see how boosting algorithms can be further improved in these aspects.

Acknowledgments

The author thanks Yaser Abu-Mostafa, Alex Holub, and the anonymous reviewers for helpful comments.

Table 4. Test errors (%) with the AdaBoost that aggregates 50 decision stumps as the base learner

dataset	one-vs-one one-vs-all	AdaBoost.ECC max-cut	AdaBoost.ECC rand-half	AdaBoost.ERP (max-2) LR	AdaBoost.ERP (max-2) LRLR	AdaBoost.ERP (rand-2) LR	AdaBoost.ERP (rand-2) LRLR
dna	**6.32**	7.93	7.30 ± 0.03	7.50	**6.75**	7.48 ± 0.05	7.17 ± 0.04
glass	26.57 ± 0.87	27.29 ± 0.96	26.52 ± 0.91	26.48 ± 0.92	26.48 ± 0.94	26.62 ± 0.95	$\mathbf{25.57 \pm 0.90}$
iris	6.00 ± 0.60	6.40 ± 0.58	7.00 ± 0.57	$\mathbf{5.67 \pm 0.59}$	6.20 ± 0.59	79.33 ± 0.75	9.13 ± 1.47
letter	**12.12**	40.12 ± 0.24	20.82 ± 0.05	27.88	**16.05**	16.89 ± 0.05	16.30 ± 0.04
pendigits	4.92	8.81	5.37 ± 0.02	$\mathbf{4.00 \pm 0.01}$	5.69	5.55 ± 0.08	5.12 ± 0.02
satimage	**12.55**	14.60	13.87 ± 0.04	14.75	13.65	$\mathbf{12.74 \pm 0.05}$	13.67 ± 0.04
segment	2.66 ± 0.11	2.44 ± 0.10	$\mathbf{1.89 \pm 0.08}$	1.94 ± 0.10	2.15 ± 0.09	2.18 ± 0.10	1.96 ± 0.09
vehicle	24.66 ± 0.41	24.60 ± 0.47	22.82 ± 0.45	26.01 ± 0.43	23.32 ± 0.44	$\mathbf{22.80 \pm 0.43}$	23.05 ± 0.44
vowel	**46.10**	56.93	56.64 ± 0.14	$\mathbf{50.33 \pm 0.03}$	57.14	51.35 ± 0.23	56.21 ± 0.14
wine	$\mathbf{2.61 \pm 0.34}$	4.72 ± 0.51	$\mathbf{2.94 \pm 0.40}$	3.50 ± 0.42	4.72 ± 0.48	3.17 ± 0.37	3.22 ± 0.42

Table 5. Test errors (%) of multiclass algorithms with the SVM-perceptron as the base learner

dataset	one-vs-one one-vs-all	AdaBoost.ECC max-cut	AdaBoost.ECC rand-half	AdaBoost.ERP (max-2) LR	AdaBoost.ERP (max-2) LRLR	AdaBoost.ERP (rand-2) LR	AdaBoost.ERP (rand-2) LRLR
glass	28.71 ± 0.96	28.52 ± 0.90	28.14 ± 1.01	29.00 ± 0.98	$\mathbf{28.05 \pm 0.88}$	28.24 ± 0.92	28.19 ± 0.87
iris	4.00 ± 0.47	3.87 ± 0.52	$\mathbf{3.73 \pm 0.49}$	$\mathbf{3.73 \pm 0.48}$	3.93 ± 0.49	3.93 ± 0.50	$\mathbf{3.73 \pm 0.49}$
pendigits	1.71 ± 0.00	1.80 ± 0.01	1.81 ± 0.03	2.34 ± 0.04	$\mathbf{1.64 \pm 0.02}$	3.71 ± 0.14	1.81 ± 0.04
satimage	7.70	7.66 ± 0.02	7.70 ± 0.07	7.71 ± 0.02	7.72 ± 0.05	7.76 ± 0.02	$\mathbf{7.63 \pm 0.06}$
segment	2.09 ± 0.09	2.21 ± 0.10	$\mathbf{2.08 \pm 0.09}$	2.09 ± 0.09	2.16 ± 0.09	2.10 ± 0.09	2.14 ± 0.09
vehicle	17.89 ± 0.37	19.08 ± 0.39	18.65 ± 0.37	17.93 ± 0.38	$\mathbf{17.67 \pm 0.35}$	17.96 ± 0.37	17.89 ± 0.37
vowel	37.45	39.49 ± 0.14	39.42 ± 0.29	$\mathbf{36.44 \pm 0.02}$	39.95 ± 0.19	38.16 ± 0.37	40.03 ± 0.25
wine	0.94 ± 0.22	1.22 ± 0.26	0.94 ± 0.25	$\mathbf{0.89 \pm 0.23}$	1.06 ± 0.23	1.17 ± 0.25	0.94 ± 0.21

This work was supported by the Caltech SISL Graduate Fellowship.

References

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

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference (ICML '96)* (pp. 148–156). Morgan Kaufmann.

Guruswami, V., & Sahai, A. (1999). Multiclass learning, boosting, and error-correcting codes. *Proceedings of the Twelfth Annual Conference on Computational Learning Theory* (pp. 145–155). ACM Press.

Hettich, S., Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases.

Li, L. (2005). *Perceptron learning with random coordinate descent* (Computer Science Technical Report CaltechCSTR:2005.006). California Institute of Technology, Pasadena, CA.

Lin, H.-T., & Li, L. (2005). Novel distance-based SVM kernels for infinite ensemble learning. *Proceedings of the 12th International Conference on Neural Information Processing* (pp. 761–766).

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (2000). Functional gradient techniques for combining hypotheses. In A. J. Smola, P. L. Bartlett, B. Schölkopf and D. Schuurmans (Eds.), *Advances in large margin classifiers*, chapter 12, 221–246. MIT Press.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (Eds.). (1994). *Machine learning, neural and statistical classification*. Ellis Horwood.

Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. *Advances in Neural Information Processing Systems 12* (pp. 547–553). MIT Press.

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems. *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)* (pp. 313–321). Morgan Kaufmann.

Sun, Y., Todorovic, S., Li, J., & Wu, D. (2005). Unifying the error-correcting and output-code AdaBoost within the margin framework. *ICML 2005: Proceedings of the 22nd International Conference on Machine Learning* (pp. 872–879). Omnipress.

Vapnik, V. N. (1999). *The nature of statistical learning theory*. Springer-Verlag. 2nd edition.