Multiclass Learning, Boosting, and Error-Correcting Codes

Venkatesan Guruswami*
MIT Laboratory for Computer Science
Cambridge, MA.

Amit Sahai*
MIT Laboratory for Computer Science
Cambridge, MA.

Abstract

We focus on methods to solve multiclass learning problems by using only simple and efficient binary learners. We investigate the approach of Dietterich and Bakiri [2] based on error-correcting codes (which we call ECC). We distill *error correlation* as one of the key parameters influencing the performance of the ECC approach, and prove upper and lower bounds on the training error of the final hypothesis in terms of the error-correlation between the various binary hypotheses.

Boosting is a powerful and well-studied learning technique that appears to annul error correlation disadvantages by cleverly weighting training examples and hypotheses. An interesting algorithm called ADABOOST.OC [12] combines boosting with the ECC approach and gives an algorithm that has the performance advantages of boosting and at the same time relies only on simple binary weak learners. We propose a variant of this algorithm, which we call ADABOOST.ECC, that, by using a different weighting of the votes of the weak hypotheses, is able to improve on the performance of ADABOOST.OC, both theoretically and experimentally, and in addition is arguably a *more direct* reduction of multiclass learning to binary learning problems than previous multiclass boosting algorithms.

1 Introduction

We focus on methods to solve multiclass learning tasks by reducing them to learning binary classification problems. In multiclass learning, the concept we seek to learn takes values from a discrete set of $k > 2$ "classes." This notion captures many real-world learning problems. For example, in digit recognition, the underlying concept would classify a hand-printed digit to one of $k = 10$ classes. One difficulty that arises in learning a multiclass concept is that most "general purpose" learning algorithms known, such as neural networks, are best suited (or work only) for the case when the concept being learned takes on *binary* values. The theory of learning has also tended to focus mostly on binary learning. This motivates the question of how one can use and combine several binary learners to perform multiclass learning, and it is this question which we address in this work.

Perhaps the most straightforward reduction from multiclass learning to binary learning is to use a separate binary learner to learn each individual class. Thus we obtain a collection of hypotheses each of which tries to predict whether an instance belongs to one particular class. The final hypothesis classifies an instance to belong to the class whose associated hypothesis labels it positive, if a *unique* such class exists; otherwise ties are broken arbitrarily. Following [2], we call this the *one-per-class* approach.

Another approach, along the lines of the one used by Sejnowski and Rosenberg [14] in their widely known NETtalk system, is to associate with each class a unique binary string of length n. Then n binary hypotheses are learned, one for each of the n bit positions. During training for an example from class i, the desired outputs of these n hypotheses are specified by the binary string associated with class i. A test example is then classified to belong to the class whose associated n bit string is closest in Hamming distance to the sequence of predictions generated by the n hypotheses. This led Dietterich and Bakiri [2] to the beautiful idea of picking the strings associated with the classes to belong to an error-correcting code so that misclassifications by a few of the binary hypotheses can be corrected. We call this approach the ECC approach.

Dietterich and Bakiri [2] consider picking codes with strong error-correction properties and experimentally demonstrate the rather good performance of this approach on some standard multiclass learning data sets. In this work, we attempt to prove a theoretical statement about the performance of the ECC approach. We prove that the worst-case training error of the hypothesis produced by the ECC approach is significantly better than that of the one-per-class approach. One reason why the powerful theorems from coding theory cannot be directly applied to prove stronger bounds on the performance of the ECC approach is that, unlike in coding theory where one usually assumes that the errors in the different

*Email: {venkat,amits}@lcs.mit.edu. Supported by a DOD NDSEG doctoral fellowship and partially by DARPA grant DABT63-96-C-0018.

bit positions occur independently, in our case the errors made by the various hypothesis can be (and typically will be) considerably correlated. We distill "error-correlation" as one of the key parameters influencing the performance of the ECC approach. We prove that given strong error-correction properties of the underlying code and a small error-correlation between the hypotheses, the ECC approach is guaranteed to produce a hypothesis with small error. To complement this result, we prove that no matter how good the underlying code is, if the error-correlation is high, then the ECC approach can give a final hypothesis with very poor performance.

Another powerful and extensively studied method for learning is boosting [11, 3, 5, 9, 6, 7, 1, 12, 13]. This approach, which has been pioneered by Freund and Schapire [11, 3, 5, 6, 12], seems to handle error correlation disadvantages well by clever reweighting of the example space. Boosting gives a general method to produce a highly accurate combined hypothesis from simple constituent hypotheses (called "weak hypotheses") which only perform slightly better than random guessing. The combined hypothesis is an appropriately weighted vote of the weak hypotheses, and has a strong provable performance guarantee (in terms of its error). While several boosting algorithms [5, 13] lose a lot in terms of their run-time efficiency when applied to multiclass learning problems, an interesting variant called ADABOOST.OC due to Schapire [12], combines boosting with the ECC approach, to give an algorithm that has the performance advantages of boosting while relying only on binary weak learners. We propose a variant of this algorithm, which we call ADABOOST.ECC , that uses a better weighting of the weak hypotheses and in fact chooses the weights purely as a function of the error of the binary weak hypotheses, and therefore represents a more "direct" reduction of multiclass learning to binary learning. This algorithm is the main contribution of the paper, and it obtains an improved error convergence rate theoretically, and also significantly outperforms ADABOOST.OC in our experiments. In fact, in one of the experiments our algorithm achieves an improvement of over 20% in the test error. It also improves over the reported error of Real ADABOOST.MH [13], which has been observed to perform exceedingly well on several multiclass learning problems, for a similar computation time.

ORGANIZATION. We begin by reviewing some basic properties of error-correcting codes that are necessary for understanding the ECC approach. Then, in Section 3, we present our results on the ECC approach. In Section 4, we discuss the various multiclass boosting algorithms available in the literature. We then consider one of them, ADABOOST.OC, in detail, and then present our boosting algorithm ADABOOST.ECC, together with an analysis of its performance. Finally, in Section 5 we describe the experiments we conducted using our learning algorithms and report the observed results.

2 Basics of Error-Correcting Codes

In this section, we define error-correcting codes (ECC's) and give their most elementary properties. We will also be using a particular family of codes called Hadamard-matrix codes. These are codes obtained from standard Hadamard matrices. For notational convenience, when referring to binary values, we prefer to use -1 and 1 instead of 0 and 1 respectively.

Definition 1 *Given two n-bit binary vectors x and y, the* Hamming distance *between x and y is the number of bit positions on which x and y differ.*

Definition 2 *An (n, K, d)* error-correcting code *$\mathcal{C}$ is a set of K binary vectors of dimension n, called* codewords*, such that the Hamming distance between every pair of distinct codewords is at least d.*

The simplest error-correction property of an (n, K, d) error-correcting code $\mathcal{C}$ is that, since every codeword has distance at least d from every other codeword, the closed Hamming balls of radius $\lfloor \frac{d-1}{2} \rfloor$ around each codeword are disjoint. Hence, if a binary vector v differs from some codeword $x \in \mathcal{C}$ in at most $\lfloor \frac{d-1}{2} \rfloor$ positions, then x is the unique closest codeword in $\mathcal{C}$ to v. Hence we say that the code $\mathcal{C}$ can correct at least $\lfloor \frac{d-1}{2} \rfloor$ errors. Note also that any subset of size K' of an (n, K, d) code, called a *subcode*, is an (n, K', d) code.

Hadamard-matrix codes: We give an explicit family $\{H_n\}$ of $(2^n, 2^n, 2^{n-1})$ codes called the Hadamard-matrix codes. These codes have the property that if one writes all the codewords as the rows of a 2^n by 2^n matrix, then in fact the matrix is symmetric, so in particular the columns also have distance 2^{n-1}. In fact, if one takes an $(2^n, 2^n, 2^{n-1})$ Hadamard-matrix code, and adds all the negations of the codewords (that is, for each codeword, replaces each 1 with a -1 and vice-versa), then one retains the distance property, *i.e.* one obtains a $(2^n, 2^{n+1}, 2^{n-1})$ code.

These codes are constructed iteratively. The base case of the construction is to create a $(2, 2, 1)$-code. This is given by the rows of the matrix:

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

All the properties of Hadamard-matrix codes given above can be readily verified for this code.

Then H_{n+1}, the matrix for the $(2^{n+1}, 2^{n+1}, 2^n)$ Hadamard-matrix code, is created recursively by the matrix:

$$H_{n+1} = \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix}$$

Here, the distance grows by a factor of two because for any two given codewords x and y, either the first 2^n positions of x and y are the same or they are not. If they are the same, then the last 2^n positions of y are negations of the last 2^n positions of x, and thus the distance is 2^n. If the first 2^n positions are not the same then by the distance properties of H_n, both the first 2^n positions and the last 2^n positions are at distance 2^{n-1} from each other, and so the distance bound is established. A similar argument shows that the distance property is preserved when the negations of all codewords are added.

3 Using Error-Correcting Codes

The use of error-correcting codes for reducing multiclass learning problems to binary learning problems was introduced by Dietterich and Bakiri [2]. This approach essentially proceeds by choosing some (n, k, d)-error-correcting code C, where k is the number of classes, and assigning to each class a codeword from the error-correcting code. We imagine creating a k by n binary matrix where the i'th row is the codeword corresponding to the i'th class. We then build a *binary* learning problem for each column – in the problem for column j of the matrix, we label an instance as positive or negative depending on the value of the j'th bit of the codeword corresponding to its original class. Now using some binary learner to learn each column, we obtain a hypothesis h_j for each column of the matrix. To classify a new instance $\mathbf{x}$, we evaluate $h_1(\mathbf{x}), ...h_n(\mathbf{x})$ to produce an n-bit binary vector v. Finally, we classify $\mathbf{x}$ as belonging to the class whose codeword is closest to v in Hamming distance. Because the codewords come from an error-correcting code, we know that even if some of the individual hypotheses h_j were wrong, we will still classify $\mathbf{x}$ in the right class. Indeed, we know that if at most $\lfloor \frac{d-1}{2} \rfloor$ of the h_j were wrong, the final classification will be correct.

Dietterich and Bakiri [2] gave some heuristics for choosing good codes for this approach[1] and ran several experiments showing that the ECC approach can be quite successful.

3.1 Comparing the ECC approach and one-per-class

We now give theoretical evidence that the ECC approach is superior to the one-per-class approach in terms of the performance of the underlying binary learner. In practice, however, since the binary concepts learned in the one-per-class approach are more "natural" than the ones produced by the ECC method, many binary learners exhibit superior performance when used in the one-per-class approach. In the experimental section (see Appendix B), we describe an attempt to combine the two approaches which seems to consistently outperform each approach alone.

We first examine the worst-case training performance of the one-per-class approach in terms of the performance of the binary learner. In the following analyses, we suppose that the binary learner is used to output n hypotheses $h_1, \ldots, h_n$, each with (fractional) training error $e_1, \ldots, e_n$, respectively. (For the one-per-class approach, n will equal the number of classes k.) It is immediate that the error of the final hypothesis of one-per-class is upper bounded by $\sum_{i=1}^{k} e_i$. However, in the worst case the error can in fact be this high.

Lemma 1 *The worst-case training error of one-per-class can be as high as* $\min\{\sum_{i=1}^{k} e_i, 1\}$*; and for randomized one-per-class it can be as high as* $\min\{\frac{k-1}{k}\sum_{i=1}^{k} e_i, 1\}$.

Proof: The worst case is where each hypothesis makes all its errors as "false negatives," *i.e.* it classifies positive examples as negative. If each hypothesis does so on disjoint sets of examples (so the errors are highly uncorrelated), then for a $\min\{\sum_{i=1}^{k} e_i, 1\}$ fraction of the training examples, none of the hypotheses will return a positive value. In the case of a deterministic algorithm, in the worst case the wrong choice would be made every time, and thus the bound would follow. In the randomized version (where a random choice occurs in the case of ties), one would expect a $\frac{k-1}{k}$ fraction of the mistakes from the deterministic case to be retained. □

[1] The issues involved in code design will be discussed further later.

The situation significantly improves in the ECC approach:

Lemma 2 *The worst-case training error of the ECC approach using an (n, k, d) code can be no higher than $2n/d$ times the* average *error $\frac{1}{n}\sum_{i=1}^{n} e_i$ of the binary hypotheses. In particular, for codes with minimum distance d at least $n/2$, this is no more than 4 times the average error.*

Proof: Suppose that there are N training examples. Then the total number of mistakes made by the hypotheses is $N\big(\sum_{i=1}^{n} e_i\big)$. But because the error-correcting code can correct up to $\lfloor (d-1)/2 \rfloor$ errors, all misclassified examples must have at least $d/2$ hypothesis errors. Thus, the total number of classification errors is bounded by $\frac{2N}{d}\sum_{i=1}^{n} e_i$. Thus the training error (as a fraction) is bounded by $\frac{2n}{d} \cdot \left(\frac{\sum e_i}{n}\right)$. □

Since codes with minimum distance $n/2$ are easy to construct as long as the number of classes k is small enough, this shows that the situation for the ECC approach is much better than in the one-per-class approach. Indeed we have established a connection between the error of a multiclass learner and the error of a binary learner, which is in itself something non-trivial. But can we expect the error-correction properties of the code to take us further? Can we hope to do better than the error of the binary learners? Unfortunately, good error correcting properties of the underlying code alone alone cannot accomplish this goal:

Lemma 3 *The worst-case training error of the ECC approach using an $(n, k, n/2)$ code, where n is a power of two, can be as high as 2 times the best error of the binary hypotheses.*

Proof: Let $n = 2^m$, and let N be the number of training examples. To construct the example which leads to the error claimed in the Lemma, we consider the $(n, n, n/2)$ Hadamard-matrix code (or any subcode). We note two relevant properties of the Hadamard-matrix code:

Claim 1 *For any codeword x, inverting the last $n/2$ positions transforms it into another codeword y. Hence, inverting the last $n/4 + 1$ positions of x creates a vector closer to y than to x.*

Proof: Immediate from the construction of the Hadamard–matrix code. □(*Claim 1*)

Claim 2 *For any codeword x, there is some subset (which depends on x) of its first $n/2 + 1$ positions that, when inverted, creates a vector that is closer to some codeword different from x than it is to x.*

Proof: To see this claim, let y be any codeword such that its $(n/2+1)$'th position is the inverse of $x_{(n/2+1)}$, and such that its first $n/2$ positions are not the same as those of x. It is immediate from the construction that such a codeword exists. Now, since the first $n/2$ positions of x and y come from a Hadamard-matrix code, it must be that this subset of positions differ on exactly $n/4$ positions. Furthermore, the same can be said about the last $n/2$ positions. Hence, if we consider a vector v that is equal to y on its first $n/2+1$ positions, but equal to x elsewhere, we have that v is at Hamming distance $n/4-1$ from y but $n/4+1$ from x. □(*Claim 2*)

Let $e_1, e_2, \ldots, e_n$ be the errors of the n binary hypotheses $h_1, h_2, \ldots, h_n$ associated with the columns $1, 2, \ldots, n$ respectively of the Hadamard-matrix. Let $f_1 = \min\{e_{3n/4}, e_{3n/4+1}, \ldots, e_n\}$. Consider the first f_1 fraction of training examples. Suppose the $n/4+1$ hypotheses $h_{3n/4}, \ldots, h_n$ all make errors on these examples. By Claim 1, all these examples would be misclassified. Further, let $f_2 = \min\{e_1, e_2, \ldots e_{n/2+1}\}$. On the next f_2 fraction of training examples, for each example, by Claim 2, we can find a subset of its first $n/2+1$ positions such that if the corresponding hypotheses made errors on that example, it would be misclassified. Thus, we have a situation where a $f_1 + f_2 \geq 2\min_i e_i$ fraction of the training examples are misclassified, and this completes the proof. □ (*Lemma 3*)

Hence, we see that error-correction properties alone will not suffice in establishing a bound that is better than the error of the binary hypotheses produced by the binary learner. It is interesting to note that a bad case for the ECC method is a situation where the errors are highly correlated; whereas the worst case for the one-per-class method is the situation where errors are extremely uncorrelated. This might suggest that some combination of these schemes might work well in practice. We have some experimental results supporting this view; these are discussed in Appendix B.

In the ECC method proposed by Dietterich and Bakiri [2], an instance is classified by finding the closest codeword in Hamming distance to the vector produced by the binary hypotheses. However this ignores the fact that the errors (and hence the reliability) of the various hypotheses may vary; we propose to use this information by employing variants of "soft-decision" decoding in the classification stage. In particular, we propose two approaches, which we call symmetric and asymmetric Maximum Likelihood Decoding. These approaches are discussed in Appendix A.

3.2 Error Correlation

In the world of communication, error-correcting codes can be used to transform a highly noisy channel into one with extremely low noise. An analogy of this in the world of learning would imply that even if the binary hypotheses have high error rates, using an error-correcting code should dramatically reduce the final error. Why then can we construct the counterexample of Lemma 3? The answer lies in the correlations between the errors of the hypotheses. In communication theory, it is typically assumed that transmission errors will occur independently, or at worst with some local dependence (e.g. *bursty noise*); only under such assumptions can the powerful theorems in coding theory be established. In the learning scenario, correlation between errors in the ECC method depend not only on the underlying data set and the binary learner, but also on the *code* itself! This is a situation not dealt with in classical communication theory. There is strong intuition that it is indeed the correlation of errors that could cause problems. Based on this intuition, Dietterich and Bakiri addressed this issue by focusing on codes with good column distance properties. Their intuition was that if the columns which define the individual binary learning problems are quite different, then the correlation between hypothesis errors should be reduced. Of course, this is only a heuristic, since the example given in Lemma 3 uses the Hadamard-matrix code which has excellent column distance properties. In our work, we choose instead to attack the problem of error correlation directly. First, we attempt to quantify the intuition that error correlation is important with the following Lemma:

Lemma 4 *Let Δ be an upper bound on $\Pr_x[h_i(x)$ is wrong and $h_j(x)$ is wrong$]$ for all $i, j,\ 1 \leq i < j \leq n$. Then the worst-case training error of the ECC approach using an (n, k, d) code can be no higher than $4\frac{n(n-1)}{d(d-2)}\Delta$. In particular, for codes with minimum distance d at least $n/2+2$, this is no more than 16Δ.*

Proof: Let N be the number of training examples, and let ε be the training error of the final ECC hypothesis. We count the number of pairs of individual binary hypothesis errors made over all examples. For each of the εN examples that were misclassified, at least $\binom{d/2}{2}$ pairs of errors must have occurred. On the other hand, by definition of Δ, the total number of pairs of errors made by the hypotheses over all examples is at most $\binom{n}{2}\Delta N$. We therefore have

$$\varepsilon N \binom{d/2}{2} \leq \binom{n}{2} \Delta N, \text{ and hence,}$$

$$\varepsilon \leq \frac{\binom{n}{2}}{\binom{d/2}{2}} \Delta \leq 4\frac{n(n-1)}{d(d-2)}\Delta \qquad \square$$

This Lemma can, of course, be generalized to "higher moments" of error correlation. Note that it formalizes and quantifies our intuition that low error correlation together with good distance properties of the underlying code implies that the ECC approach will work well.

4 Boosting

An Active Approach. Above, we showed that achieving small error-correlation in the ECC approach would imply small overall error. Dietterich and Bakiri tried to attack this problem by tailoring their error-correcting codes to discourage error-correlation; but as we have seen, this is only a heuristic approach. Instead, we propose to *actively* work to find binary hypotheses with controlled error-correlation. We observe that the powerful method of boosting, by cleverly weighting training examples and hypotheses, seems to overcome the problem of error correlation in combining many hypotheses. Thus, in this section, we investigate how to combine techniques from boosting and error correcting codes to get the advantages of both. We re-interpret an existing algorithm of this type, called ADABOOST.OC [12], as one that uses boosting techniques to find the binary hypotheses, but then uses weighted closest-codeword decoding to classify instances. With our perspective, we refine the method by which the binary hypotheses are found and weighted. We obtain a new, somewhat simpler, algorithm which we call ADABOOST.ECC, that achieves a stronger theoretical performance guarantee. We also present some experimental evidence that our algorithm performs better in practice, as well.

4.1 Boosting Overview

Boosting is a powerful and well-studied method of finding a (provably) highly accurate hypothesis (classification rule) by combining many "weak" hypotheses generated by a base learning algorithm, each of which is only moderately accurate. Boosting algorithms operate in several rounds. During each round they reweight examples in the training set and rerun the base learning algorithm on these reweighted examples. Effectively, boosting forces the weak learning algorithm to concentrate on the "hardest examples" (the ones misclassified so far). One can achieve a sufficiently high accuracy on the training examples by running a large number of rounds of boosting. The final hypothesis is, typically, a weighted vote of the weak hypotheses. By keeping each of the weak hypotheses to be a simple rule, one can then control the complexity of the final hypothesis, and thereby, using VC-theory [15], expect a low error on the test examples as well.

The first boosting algorithms were discovered by Schapire [11] and later improved by Freund [3]. These algorithms are pretty complicated and not suitable for easy implementation. A major breakthrough came in the form of Freund and Schapire's ADABOOST algorithm [5] which is extremely efficient and also very easy to implement. It has received extensive empirical and theoretical study and has been found to work very well on several practical binary classification problems [10, 9, 6, 1, 7, 13]. There have also been boosting algorithms for multiclass learning which we discuss next.

4.2 Boosting for Multiclass problems

The binary ADABOOST algorithm requires that the accuracy of each classification rule produced by the weak learner be (non-negligibly) greater than $1/2$. For binary classification problems, this requirement is about as minimal as can be hoped for, since random guessing will itself achieve an accuracy of $1/2$. For multiclass problems, however, in which $k > 2$ labels are possible, an accuracy of $1/2$ may be much harder to obtain than the random guessing accuracy rate of $1/k$.

If the underlying weak-learners are fairly powerful (like C4.5 for instance [10]) this is not a big problem since, as observed in [12], they usually get accuracy 1/2 even on the difficult distributions of examples produced by boosting. While the overall training error rate is usually much lower when more expressive and powerful weak learners are used, this implies a lot more computation time (which is a vital issue, especially for large datasets) and a higher complexity of the final hypothesis which might result in increased test error. Therefore from a practical standpoint, we are more interested in the issue of what can be achieved with very simple weak learners.

Along these lines, Freund and Schapire [5] proposed a *pseudoloss* boosting algorithm ADABOOST.M2 for multiclass learning, where the weak learner at each round chooses a set of plausible labels for each example. For example, in a digit recognition problem, the weak hypothesis may predict a certain example to be a 0, 4 or 9, rather than just a single digit. The hypothesis is then evaluated according to a related *pseudoloss* measure, and boosting proceeds as in the binary case except that we now use a distribution on example–label pairs in each round. The final hypothesis classifies an instance according to the single label that occurs most frequently in the plausible label sets chosen by the weak hypotheses, ties being broken arbitrarily.

Experiments conducted with this algorithm [6] indicate that it performs well, but has some significant drawbacks. First of all, it requires weak hypotheses optimized for the more complex pseudoloss measure, thus complicating the design of the weak learners (eg. most off-the-shelf weak learners cannot handle this measure). Secondly, this approach is fairly slow as the run-time of the weak learner is roughly $O(k)$ times slower than that of a pure error-based simple binary weak learner.

These drawbacks motivated Schapire [12] to design an alternative algorithm, called ADABOOST.OC, that combines boosting with the ECC approach we saw in Section 3. A formal description of ADABOOST.OC is provided in Figure 1.

This algorithm works as follows. As in boosting, in each round t of running the weak learner, the examples are reweighted in a manner focusing on the hardest examples by a distribution D_t on the set X of training instances. And following ECC, in round t of boosting a binary coloring $\mu_t : Y \longrightarrow \{-1, +1\}$ of the set Y of labels is picked to create a new binary classification problem (by relabeling the training examples according to μ_t). The weak learner is then used to obtain a weak hypothesis $h_t : X \longrightarrow \{-1, +1\}$. The goal of the weak learner is to minimize the error $\varepsilon_t = \mathbf{Pr}_{i \sim D_t}\left[h_t(x_i) \neq \mu_t(y_i)\right]$. A hypothesis $\tilde{h}_t$ is then defined as $\tilde{h}_t(x) = \{l \in Y : h_t(x) = \mu_t(l)\}$, its *pseudoloss* $\tilde{\varepsilon}_t$ is computed, and a weight α_t for the vote of round t is computed as a function of $\tilde{\varepsilon}_t$.

Finally, the combined hypothesis H_f is computed. This is done by, given an instance x, interpreting the binary classification $h_t(x)$ of round t as a vote for all labels in $\tilde{h}_t(x)$ and *weighting* this vote by α_t. The label l that receives the most weighted votes in this scheme is chosen as $H_f(x)$.

ADABOOST.OC combines the advantages of both the boosting and ECC paradigms: it is easy to implement as the weak learning algorithm need only be able to handle "pure" binary problems, and it comes with a strong theoretical guarantee (Theorem 1 of [12]) that if the weak learner can consistently generate weak hypotheses that are slightly better than random guessing, then with sufficient number of rounds of boosting, the error of the final hypothesis can be made arbitrarily small.

Subsequent work in [13] gives an alternative algorithm, called ADABOOST.MH, that handles multiclass problems (in fact multilabel problems where each example belongs to a certain subset of classes). This algorithm does not rely on picking colorings at each round of boosting but instead learns weak hypotheses whose domain is $X \times Y$ (as opposed to simply X as in ADABOOST.OC). For this reason, ADABOOST.MH, while giving improved performance over ADABOOST.OC in several cases, runs for roughly a factor $O(k)$ time longer (here $k = |Y|$ is the number of classes).

This motivates the attempt to improve the performance of ADABOOST.OC in practice (and if possible even in theory) *without* sacrificing what is arguably its most compelling advantage, namely its simplicity and run-time efficiency. It is this question which we address, and design a variant of

Given: $(x_1, y_1), \ldots, (x_m, y_m) : x_i \in X, y_i \in Y, |Y| = k$
Initialize: $\tilde{D}_1(i,l) = 1/m(k-1)$ if $l \neq y_i$, $\tilde{D}_1(l,l) = 0 \; \forall l \in Y$.

For $t = 1, 2, \ldots, T$:
Compute coloring $\mu_t : Y \longrightarrow \{-1, +1\}$.
Let $U_t = \sum_{i=1}^m \sum_{l \in Y} \tilde{D}_t(i,l) [\![\mu_t(y_i) \neq \mu_t(l)]\!]$.
Let $D_t(i) = \frac{1}{U_t} \cdot \sum_{l \in Y} \tilde{D}_t(i,l) [\![\mu_t(y_i) \neq \mu_t(l)]\!]$.
Get hypothesis $h_t : X \longrightarrow \{-1, +1\}$ from the weak learner for distribution D_t.
Let $\tilde{h}_t(x) = \{ l \in Y : h_t(x) = \mu_t(l) \}$.
Let $\tilde{\varepsilon}_t = \frac{1}{2} \sum_{i=1}^m \sum_{l \in Y} \tilde{D}_t(i,l) \cdot \left([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![l \in \tilde{h}_t(x_i)]\!] \right)$.
Let $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \tilde{\varepsilon}_t}{\tilde{\varepsilon}_t} \right)$.
Update:
$$\tilde{D}_{t+1}(i,l) = \frac{1}{\tilde{Z}_t} \cdot \tilde{D}_t(i,l) \exp \left\{ \alpha_t ([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![l \in \tilde{h}_t(x_i)]\!]) \right\}$$
where $\tilde{Z}_t$ is a normalization factor.

Output the final hypothesis:
$$H_f(x) = \operatorname{argmax}_{l \in Y} \sum_{t=1}^T \alpha_t [\![l \in \tilde{h}_t(x)]\!].$$

Figure 1: A description of ADABOOST.OC [12]

ADABOOST.OC, which we call ADABOOST.ECC, that is in fact slightly simpler than ADABOOST.OC but performs better both in practice and in theory.

4.3 Our new algorithm ADABOOST.ECC

Our algorithm (see Figure 2) proceeds along the lines of ADABOOST.OC, but the main point of difference in our algorithm is that instead of picking a weight α_t based on the pseudoloss $\tilde{\varepsilon}_t$ as in ADABOOST.OC, we pick weights α_t and β_t for the positive and negative votes of $h_t : X \longrightarrow \{-1, +1\}$ that are based just on the structure of h_t and its performance on the binary learning problem of round t; no pseudoloss measure is computed or used. (It is possible that we may pick one or both of α_t, β_t to be negative.) We then form a new hypothesis $g_t : X \longrightarrow \Re$ defined by:

$$\begin{aligned} g_t(x) &= \alpha_t \text{ if } h_t(x) = +1 \\ &= -\beta_t \text{ if } h_t(x) = -1. \end{aligned}$$

Now given an example x, we view $g_t(x)$ as providing a vote of value $g_t(x) \cdot \mu_t(l)$ for the label l, and the final hypothesis $H_f(x)$ outputs that label y for which the algebraic sum of its votes from the various g_t's is maximum. Note that this is exactly a *weighted closest codeword decoding* for the error-correcting code formed by the colorings μ_t. A formal description of the algorithm is given in Figure 2. Note that explicit computation of the *pseudoloss* as is done by ADABOOST.OC is not performed in this algorithm. We now provide a theoretical guarantee on the training error of the hypothesis produced by ADABOOST.ECC .

Theorem 1 *The training error of the hypothesis H_f under the uniform distribution on training examples is at most*

$$(k-1) \prod_{i=1}^T \tilde{Z}_t = (k-1) \prod_{t=1}^T (Z_t U_t + 1 - U_t).$$

where $Z_t = \sum_{i=1}^m D_t(i) \exp\big(- g_t(x_i) \mu_t(y_i) \big)$.

Proof: The proof is based on ideas from Theorem 11 of [5]. Suppose that the final hypothesis H_f of ADABOOST.ECC makes a mistake on instance x_i, so that $H_f(x_i) \neq y_i$. Then, by the definition of H_f,

$$\sum_{t=1}^T \left[g_t(x_i) \mu_t(H_f(x_i)) - g_t(x_i) \mu_t(y_i) \right] \geq 0 \qquad (1)$$

Now consider the binary problem of learning the all FALSE concept f_{-1} on the the domain $\{x_1, x_2, \ldots, x_m\} \times Y$ under the uniform distribution $\tilde{D}_1$ over all pairs (x_i, l) with $l \in Y - \{y_i\}$: i.e $f(x,l) = -1 \; \forall x \in X, l \in Y$. We view the update step (for $\tilde{D}_t(\cdot,\cdot)$) in ADABOOST.ECC as the update rule of the binary ADABOOST algorithm (see [5]), whose weak learner, at round t, returns the weak hypothesis $\tilde{f}_t(x_i, l) \triangleq (1/2) \times (g_t(x_i)\mu_t(l) - g_t(x_i)\mu_t(y_i))$ and we choose its weight to be 1. By the analysis of binary ADABOOST presented in [13], the probability that for a uniformly chosen pair (x_i, l) for $1 \leq i \leq m$ and $l \in Y - \{y_i\}$, we can have

$$\hat{f}(x_i, l) \triangleq \operatorname{sgn}\Big(\sum_{t=1}^T \tilde{f}_t(x_i, l) \Big) = 1$$

(which amounts to an error in learning the concept f_{-1}) is at most $\prod_{t=1}^T \tilde{Z}_t$.

By Equation 1, for any example x_i such that $H_f(x_i) \neq y_i$ in ADABOOST.ECC , we have $\hat{f}(x_i, H_f(x_i)) = 1$. Hence if U denotes the uniform distribution on the training examples, we have

$$\begin{aligned} \Pr_{x_i \sim U}[H_f(x_i) \neq y_i)] &\leq \Pr_{x_i \sim U}[\exists l \neq y_i : \hat{f}(x_i, l) = 1] \\ &\leq (k-1) \Pr_{(x_i,l) \sim \tilde{D}_1} [\hat{f}(x_i, l) = 1] \\ &\leq (k-1) \prod_{t=1}^T \tilde{Z}_t \end{aligned}$$

Given: $(x_1, y_1), \ldots, (x_m, y_m) : x_i \in X, y_i \in Y, |Y| = k$
Initialize: $\tilde{D}_1(i,l) = 1/m(k-1)$ if $l \neq y_i$, $\tilde{D}_1(i,l) = 0$ if $l = y_i$.

For $t = 1, 2, \ldots, T$:

Compute coloring $\mu_t : Y \longrightarrow \{-1, +1\}$.

Let $U_t = \sum_{i=1}^m \sum_{l \in Y} \tilde{D}_t(i,l)[\![\mu_t(y_i) \neq \mu_t(l)]\!]$.

Let $D_t(i) = \frac{1}{U_t} \cdot \sum_{l \in Y} \tilde{D}_t(i,l)[\![\mu_t(y_i) \neq \mu_t(l)]\!]$.

Get hypothesis $h_t : X \longrightarrow \{-1, +1\}$ from the weak learner for distribution D_t.

Compute the weight of positive and negative votes α_t and β_t respectively.

Define:

$$g_t(x) = \alpha_t \text{ if } h_t(x) = +1$$
$$= -\beta_t \text{ if } h_t(x) = -1.$$

Update:

$$\tilde{D}_{t+1}(i,l) = \frac{1}{\tilde{Z}_t} \cdot \tilde{D}_t(i,l) \exp\left\{(g_t(x_i)\mu_t(l) - g_t(x_i)\mu_t(y_i))/2\right\}$$

where $\tilde{Z}_t$ is a normalization factor.

Output the final hypothesis:

$$H_f(x) = \text{argmax}_{l \in Y} \sum_{t=1}^T g_t(x)\mu_t(l).$$

Figure 2: A description of ADABOOST.ECC

To complete the proof, it remains to express $\tilde{Z}_t$ in terms of Z_t and U_t. Now,

$$\begin{aligned}
\tilde{Z}_t &= \sum_i \sum_l \tilde{D}_t(i,l) \exp\left[\frac{g_t(x_i)\mu_t(l) - g_t(x_i)\mu_t(y_i)}{2}\right] \\
&= \sum_i \exp\left[-g_t(x_i)\mu_t(y_i)\right]\Big(\sum_{l:\mu_t(l)\neq\mu_t(y_i)} \tilde{D}_t(i,l)\Big) + \\
&\quad \sum_i \sum_{l:\mu_t(l)=\mu_t(y_i)} \tilde{D}_t(i,l) \\
&= U_t \sum_i D_t(i) \exp\left[-g_t(x_i)\mu_t(y_i)\right] + (1 - U_t) \\
&= U_t Z_t + (1 - U_t).
\end{aligned}$$

The last but one step above follows from the definitions of U_t and $D_t(\cdot)$. This completes the proof. □

Observe that the specification of ADABOOST.ECC in Figure 2 is for the *asymmetric version* (since the weight of the positive vote α_t is not necessarily equal to the weight of the negative vote β_t). One can also consider the *symmetric version* of this algorithm where $\alpha_t = \beta_t$ for each t. It turns out, using the methods from [13] on domain-partitioning weak hypotheses, that the optimum choice of α_t, β_t is given by

$$\begin{aligned}
\alpha_t &= \frac{1}{2}\ln\left(\frac{\sum_{i:h_t(x_i)=\mu_t(y_i)=1} D_t(i)}{\sum_{i:h_t(x_i)=1\neq\mu_t(y_i)} D_t(i)}\right), \\
\beta_t &= \frac{1}{2}\ln\left(\frac{\sum_{i:h_t(x_i)=\mu_t(y_i)=-1} D_t(i)}{\sum_{i:h_t(x_i)=-1\neq\mu_t(y_i)} D_t(i)}\right);
\end{aligned}$$

and for the symmetric version, the optimum choice is

$$\alpha_t = \beta_t = \frac{1}{2}\ln\left(\frac{\sum_{i:h_t(x_i)=\mu_t(y_i)} D_t(i)}{\sum_{i:h_t(x_i)\neq\mu_t(y_i)} D_t(i)}\right). \quad (2)$$

In the following, we bound the error of the symmetric version of our algorithm. We will then argue that even this bound is better than the one proved for ADABOOST.OC in [12], and of course the asymmetric version will only perform even better. We stress here that our algorithm is **not** simply ADABOOST.OC modified to incorporate the methods of [13] to pick the weights of the votes; together with picking the weights α_t, β_t using the methods of [13]; one significant feature that distinguishes our algorithm from ADABOOST.OC is that the pseudoloss is never computed or used as in ADABOOST.OC. Indeed the fact that even the symmetric version is able to do better than ADABOOST.OC implies that the improvements shown by our approach come not only from the fact that positive and negative votes are weighted differently, but also from the fact that we are choosing the weight α_t for the vote of round t directly in terms of the weak hypothesis h_t's error on D_t without going through a *pseudoloss* measure. Note that this also indicates that ADABOOST.ECC is a simpler and an even more faithful reduction of multiclass learning to binary learning problems.

Theorem 2 *Let $\mu_1, \mu_2, \ldots, \mu_t$ be any sequence of colorings and let $h_1, h_2, \ldots, h_T$ be any sequence of weak hypotheses produced by the weak learner. Let $\varepsilon_t = 1/2 - \gamma_t$ be the error of h_t with respect to the relabeled and reweighted data on which it was trained, and let U_t be as in Figure 2. Then the training error of the final hypothesis H_f output by the* symmetric *version of* ADABOOST.ECC *is bounded by*

$$(k-1)\prod_{t=1}^T (U_t\sqrt{1-4\gamma_t^2} + (1 - U_t)).$$

Proof: In the case when we use symmetric weights, by Equation 2, we have $\alpha_t = \beta_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$. For this choice of

α_t, we have

$$\begin{aligned} Z_t &= \sum_i D_t(i) \exp[-\alpha_t h_t(x_i)\mu_t(y_i)] \\ &= e^{\alpha_t}\varepsilon_t + e^{-\alpha_t}(1-\varepsilon_t) \\ &= \sqrt{1-4\gamma_t^2}. \end{aligned}$$

The statement of the theorem now follows using Theorem 1. □

Lemma 5 *For* $0 \le U_t \le 1$ *and* $0 \le \gamma_t \le 1/2$*, we have*

$$U_t\sqrt{1-4\gamma_t^2} + (1-U_t) \le \sqrt{1-4\gamma_t^2 U_t^2}.$$

Proof: Indeed if $b = \sqrt{1-4\gamma_t^2 U_t^2}$ and $a = U_t\sqrt{1-4\gamma_t^2} + (1-U_t)$, then we can compute

$$\begin{aligned} b^2 - a^2 &= 2U_t(1-U_t)(1-\sqrt{1-4\gamma_t^2}) \\ &\ge 0 \end{aligned}$$

and the lemma follows. □

By the above Lemma and Theorem 2, we have proved a *better* upper bound on the training error of even *symmetric* ADABOOST.ECC as compared to that already known bound of $(k-1)\prod_{t=1}^{T}\sqrt{1-4\gamma_t^2 U_t^2}$ for ADABOOST.OC (see Theorem 1 of [12]). In the next section, we will see that this improvement is also observable in practice in the experiments we ran.

5 Experiments

We performed experiments with our boosting algorithms on two multiclass learning problems from the UCI repository [8]. The first problem is handwritten digit recognition, called PENDIGITS in the sequel. Here, the data set has 7494 training examples and 3498 test examples. The number of classes is 10, one for each of the digits 0-9. Each instance has 16 attributes, each taking an integer value between 0 and 100. The second learning problem is a well-studied letter recognition problem, which we refer to as LETTER. The associated data set has 16000 training examples and 4000 test examples. The number of classes is 26, one for each letter of the English alphabet. Each instance has 16 attributes each of which takes an integer value between 0 and 15.

We chose these two data sets to test out our algorithms as these are among the more comprehensive and larger data sets in the repository. Also, digit and letter recognition tasks are interesting learning problems in their own right.

We use the simplest of weak learners, namely one-level decision trees. The weak hypothesis makes its prediction based on the result of a single test comparing one of the attributes to a threshold value. The best hypothesis of this form which optimizes the appropriate learning criterion can be found by direct search; we use a more efficient search for this hypothesis in our implementation.

We implemented Schapire's ADABOOST.OC [12], and also our boosting algorithms (described in Section 4.3) with both asymmetric and symmetric weightings of the various weak hypotheses. All these algorithms give test and training errors which are much better than those achieved by the plain ECC based approaches. We find that our algorithms perform significantly better than [12] on our two data sets: we find that both algorithms get nearly 15% improvement in the test error for PENDIGITS at 1000 rounds of boosting, and for LETTER, the asymmetrically weighted version of our algorithm gets over 20% improvement and the symmetric version achieves 8% improvement in the test error at 4000 rounds of boosting. These results are shown in Figures A and B. Another interesting aspect of our results is that we get a test error of 14.15% after 4000 rounds of boosting on LETTER using the asymmetric version of our algorithm, and this improves by over 15% the best previously reported error (16.4%) for boosting methods using the extremely simple weak learners we use (this error was achieved by ADABOOST.MH [13] after 1000 rounds of boosting). On the other hand, our algorithm performs only one-sixth of the computation required by ADABOOST.MH to achieve this error.

Acknowledgments

We would like to thank Raj Iyer for many useful discussions and comments on this draft, as well as for pointing us to several references and for kindly giving us some of the code used in [4]. We are grateful to Leslie Valiant for his encouragement, and for his wonderful course which prompted this research. We also thank all those who contributed to the data sets used in our experiments.

References

[1] E. Bauer and R. Kohavi. *An empirical comparison of voting classification algorithms: Bagging, boosting and variance*. Machine Learning, *to appear*.

[2] T. G. Dietterich and G. Bakiri. *Solving Multiclass Learning Problems via Error-Correcting Output Codes.* Journal of AI Research, 2 (1995), pp. 263-286.

[3] Y. Freund. *Boosting a weak learning algorithm by majority*. Information and Computation, 121 (1995), pp. 256-285.

[4] Y. Freund, R. Iyer, R. E. Schapire and Y. Singer. *An efficient boosting algorithm for combining preferences*. Machine Learning: Proceedings of the 15th International Conference, (1998).

[5] Y. Freund and R. E. Schapire. *A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting*. Journal of Computer and System Sciences, 55 (1997), pp. 119-139.

[6] Y. Freund and R. E. Schapire. *Experiments with a new boosting algorithm*. Machine Learning: Proceedings of the 13th International Conference, (1996), pp. 148-156.

[7] R. Maclin and D. Opitz. *An empirical evaluation of bagging and boosting*. In Proceedings of the 14th National Conference of Artificial Intelligence, pp. 546-551, 1997.

[8] C. J. Merz and P. M. Murphy. *UCI Repository of Machine Learning Databases*. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[9] J. R. Quinlan. *Bagging, boosting, and C.5*. In Proceed-

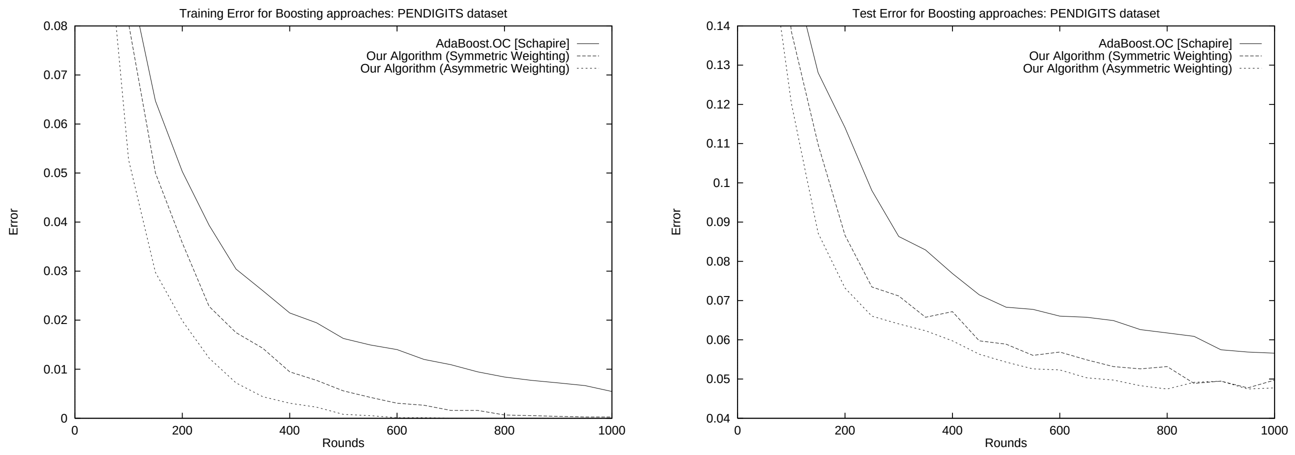

Figure A.

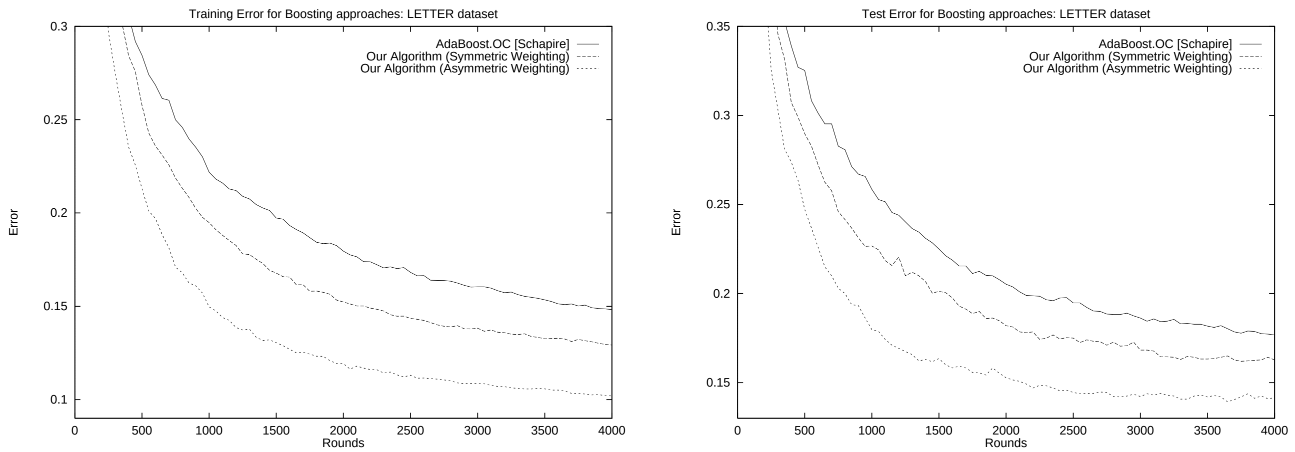

Figure B.

ings of the the 13th National Conference on Artificial Intelligence, pp. 725-730, 1996.

[10] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[11] R. E. Schapire. *The strength of weak learnability*. Machine Learning, 5 (1990), pp. 197-227.

[12] R. E. Schapire. *Using Output Codes to Boost Multi-Class Learning Problems*. Machine Learning: Proceedings of the Fourteenth International Conference, (1997), pp. 313-321.

[13] R. E. Schapire and Y. Singer. *Improved Boosting Algorithms Using Confidence-Rated Predictions*. Proceedings of the Eleventh Annual Conference on Computational Learning Theory, (1998).

[14] T. J. Sejnowski and C. R. Rosenberg. *Parallel networks that learn to pronounce English text*. Journal of Complex Systems, 1 (1987), pp. 145-168.

[15] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer Verlag, 1982.

A Maximum Likelihood Decoding

In the ECC method as proposed by Dietterich and Bakiri, the determination of the classification is done by simply finding the closest codeword in Hamming distance to the vector produced by the binary hypotheses. However, the error rates of the binary hypotheses may vary greatly. Rather than ignore this information, we first propose to use "Maximum Likelihood Decoding" (MLD), under the assumption of error independence. Here, rather than simply finding the codeword closest in Hamming distance, we attempt to find the *most likely* codeword that could have led to the vector produced by the hypotheses. We use the training data to calculate the probability of error p_i for each binary hypothesis h_i. Then, to classify an instance $\mathbf{z}$ which led to the vector $v = (h_1(\mathbf{z}), h_2(\mathbf{z}), \ldots, h_n(\mathbf{z}))$, for each codeword x we calculate the probability (assuming errors were made independently by hypotheses) that x was corrupted into v. If we let E be the set of positions i where $x_i \neq v_i$, then this probability would simply be:

$$\prod_{i \in E} p_i \prod_{i \notin E} (1 - p_i)$$

Then we classify the instance according to the codeword for which the above probability turns out to be maximum. Note that this can also be viewed as a weighted decoding. In weighted decoding, each hypothesis h_i has an associated weight w_i. To decode, for each codeword x, we compute a sum, adding $+w_i$ if the i'th hypothesis agrees with x_i, and $-w_i$ otherwise. The codeword which leads to the highest sum is then chosen as the best match. It is easy to see that the MLD approach above is equivalent to weighted decoding with hypothesis weights:

$$w_i = \frac{\log(1 - p_i) - \log(p_i)}{2}$$

Note, however, that the example of Lemma 3 can be adapted to apply to this decoding procedure. Thus, even for "Maximum Likelihood Decoding," in the worst case, the overall error could be as bad as is 2 times the minimum error of the individual binary hypotheses.

Along these lines, we can further refine the decoding by considering the probability of each hypothesis misclassifying a positive example as a negative one and vice versa separately. In this case, decoding would correspond to a weighted decoding where the weight would depend on whether the hypothesis is predicting a positive or negative example. We also attempt this type of decoding, which we call *asymmetric* MLD, and discuss its performance in Appendix B. We note, however, that Lemma 3 could again be made to apply, except now with twice the minimum one-sided error probability.

B Experiments based on the ECC approach

We now discuss our experiments using the algorithms based on the ECC approach. The binary hypotheses (corresponding to each column of the codewords) are obtained by running the binary version of ADABOOST for up to a certain number of rounds (around 500 in our experiments). Our objective here was to improve the performance of the ECC approach using better decoding techniques. Figure C shows the performance of several of our algorithms on the PENDIGITS data set and Figure D on the LETTER data set. For these experiments we used a random code of blocklength 60 where each column has a equal number of -1's and 1's. We used four kinds of decoding algorithms: the straightforward Hamming distance based nearest codeword decoding used by Dietterich and Bakiri [2], the Maximum Likelihood Decoding and its asymmetric version discussed in Appendix A, and finally, a decoding method we propose that tries to account for error correlation information. We sketch this (heuristic) method now. The method attempts to weight hypotheses according to their correlations with other hypotheses. We first calculate the pairwise error correlation on the training examples to estimate

$$\Delta_{i,j} = \Pr_x[h_i(x) \text{ is wrong AND } h_j(x) \text{ is wrong}].$$

We then set up the following linear system, in which we attempt to force hypotheses with correlated errors to share weight, so that several hypotheses that make highly correlated errors together get a weight that is comparable to the weight of a single hypothesis that makes independent uncorrelated errors. For each i, we have the constraint:

$$w_i + \sum_{j \neq i} \frac{\Delta_{i,j}}{\Delta_{i,i} + \Delta j, j} w_j = 1$$

We find that for PENDIGITS, the above correlation based approach improves the test error by over 15% at 500 rounds of boosting, and for LETTER, it improves by 13% again at 500 rounds of boosting.

Dietterich and Bakiri conjecture that high inter-column Hamming distance is useful in controlling the error correlation of the various hypotheses. We observe that Hadamard-matrix codes have this property to almost the best possible extent that can be hoped for. For example, for PENDIGITS, we picked 10 rows from a 16 $\times$ 16 Hadamard matrix, and used this as our code in the ECC approach. We observed that when combined with asymmetric MLD, this performs very well on test error as shown in Figure E. But we observe that our error correlation based decoding achieves almost the same performance starting with a random code, indicating that taking into account correlation information in the decoding stage has helped us to eliminate some of the bad column correlations of the random code.

One drawback of the ECC approach is that it seems to throw away the real world nature of the data by picking random partitions of the labels to create "artificial" binary classification problems. The one-per-class method obviously does not suffer from this drawback, as the binary problems it creates correspond to the real learning problems of whether an instance belongs to a particular class or not. Indeed, for the recognition data we used, the boosted learners performed so much better on the one-per-class learning problems that the performance of one-per-class was usually comparable to our best ECC-based algorithms. We observed that in the one-per-class approach, the errors of the binary hypotheses tend to end up being highly uncorrelated. This means that when exactly one of the class-hypotheses votes positively on an instance, one can be quite confident in the classification. When this does not occur, it means that the instance came from one of k almost disjoint sets where the individual class-hypotheses fail. The fact that many class-hypotheses probably were able to deal with the instance correctly gives hope that many of the binary hypotheses generated by the ECC approach might also be able to classify the instance correctly. Hence, we consider an algorithm combining one-per-class and the ECC approach. This algorithm trains binary hypotheses for both one-per-class and ECC. Then, to classify an instance, it firsts sees if exactly one of the class-hypotheses votes positively on the instance. If so, it outputs the corresponding class. Otherwise, we pass to the ECC decoding algorithm and answer accordingly. We compare each individual approach to the combined approach experimentally in Figures F (for PENDIGITS) and G (for LETTER). Experimentally, we find that this combined approach yields test errors that are sometimes over 25% better than those of either of the individual approaches.

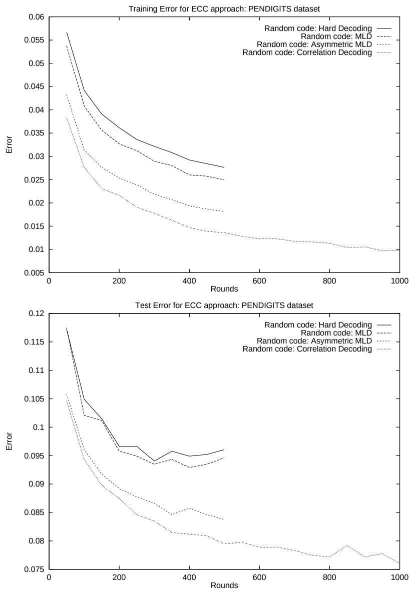

Figure C

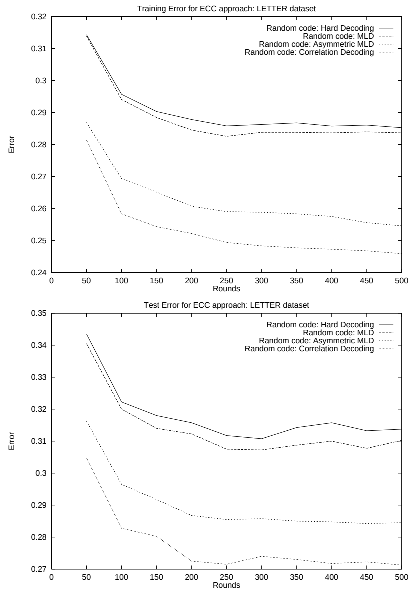

Figure D

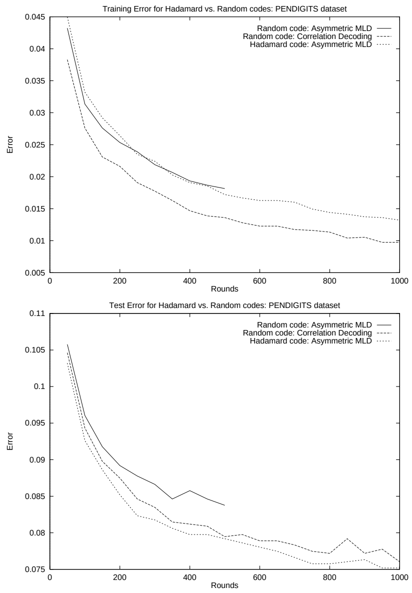

Figure E

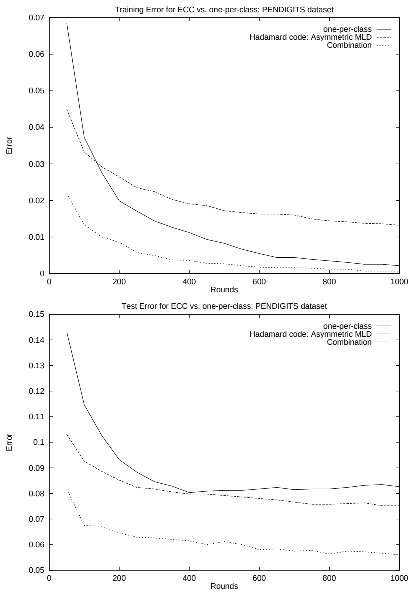

Figure F

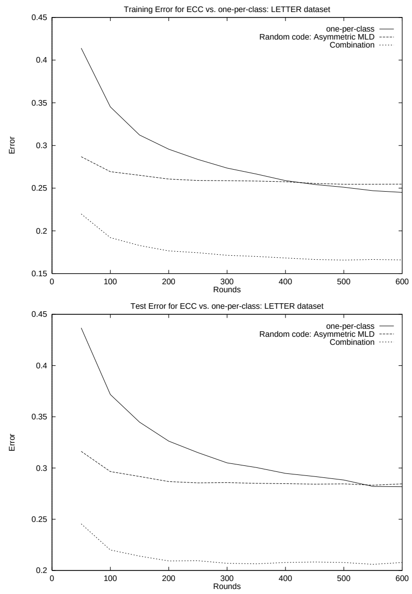

Figure G