Error-Correcting Output Coding Corrects Bias and Variance

Eun Bae Kong
Department of Computer Engineering
Chungnam National University
Taejon, 305-764, South Korea
keb@comeng.chungnam.ac.kr

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202
tgd@cs.orst.edu

Abstract

Previous research has shown that a technique called error-correcting output coding (ECOC) can dramatically improve the classification accuracy of supervised learning algorithms that learn to classify data points into one of $k \gg 2$ classes. This paper presents an investigation of why the ECOC technique works, particularly when employed with decision-tree learning algorithms. It shows that the ECOC method—like any form of voting or committee—can reduce the variance of the learning algorithm. Furthermore—unlike methods that simply combine multiple runs of the same learning algorithm—ECOC can correct for errors caused by the bias of the learning algorithm. Experiments show that this bias correction ability relies on the non-local behavior of C4.5.

1 Introduction

Error-correcting output coding (ECOC) is a method for applying binary (two-class) learning algorithms to solve k-class supervised learning problems. It works by converting the k-class supervised learning problem into a large number L of two-class supervised learning problems. Any learning algorithm that can handle two-class problems, such as the decision tree algorithm C4.5 (Quinlan, 1993a), can then be applied to learn each of these L problems. To classify a new (test) example, each of the L learned decision trees is evaluated. Then the ECOC method tells how to combine the results of these L evaluations to predict the class of the test example. Previous experimental research (Bakiri, 1991; Dietterich & Bakiri, 1991; Wettschereck & Dietterich, 1992; Dietterich & Bakiri, 1995) has shown that error-correcting output coding uniformly improves the classification accuracy of decision tree and neural network classifiers when compared with the standard approaches to k-class learning problems.

The goal of this paper is to explain why the ECOC method works so well. This paper shows that the ECOC strategy can be viewed as a compact form of "voting" among multiple hypotheses. The key to the success of this voting is that the errors committed by each of the L learned binary functions are substantially uncorrelated. To explain why the ECOC strategy works, we must therefore explain why the L learned binary functions make such uncorrelated errors.

To understand the causes of these uncorrelated errors, we appeal to the statistical notions of bias and variance. In regression, the squared error can be partitioned into a (squared) bias term and a variance term. The bias term describes the component of the error that results from systematic errors of the learning algorithm. The variance term describes the component of the error that results from random variation and noise in the training sample and random behavior in the learning algorithm. We develop analogous definitions of bias and variance for classification problems and show how bias and variance can be measured experimentally.

By measuring bias and variance, we identify two causes of the uncorrelated errors. First, as Breiman (1994) has recently shown, decision tree algorithms such as CART and C4.5 have high variance—that is, the hypotheses produced by these algorithms can change substantially with small changes in the training set. Second, this paper will show that there is another source of uncorrelated errors in the ECOC strategy—namely that the bias errors made by C4.5 on each of the L problems are substantially different. The bias errors are different because each of the L problems creates different geometrical arrangements of decision boundaries, and this leads C4.5 to make different bias errors. Hence, the near-independence of errors in the various L learning problems results from the combination of variance in C4.5 and variation in the bias due to the variation in the L problems.

There has been an explosion of papers showing that the classification performance of learning algorithms can be significantly improved by generating multiple hypotheses (e.g., as a result of different random seeds in neural network learning or different training sets in decision tree learning), and then voting these hypotheses (Hansen & Salamon, 1990; LeBlanc & Tibshirani, 1993; Perrone, 1993; Perrone & Cooper, 1993; Perrone, 1994; Meir, 1994; Breiman, 1994). We call this *homogeneous voting*, because multiple runs of the same algorithm on the *same* learning problem are combined by voting. We will argue from our definition of bias that homogeneous voting can only reduce variance and not bias.

Some papers have also reported improved performance through *non-homogeneous voting*, that is, voting multiple hypotheses that have been constructed by different learning algorithms applied to the same problem (Bates & Granger, 1969; Makridakis & Winkler, 1983; Clemen, 1989; Schapire, 1990; Hampshire II & Waibel, 1990; Zhang, Mesirov, & Waltz, 1992; Cardie, 1993; Quinlan, 1993b). Non-homogeneous voting can reduce both bias and variance if the bias errors of the various algorithms are different.

Error-correcting output coding involves neither homogeneous nor non-homogeneous voting. Like homogeneous voting, the same learning algorithm is applied many times. However, each of these applications of the algorithm solves a different learning problem, so we obtain some bias-reduction benefits as well, much like non-homogeneous voting.

The remainder of this paper is structured as follows. First, we describe the error-correcting output coding method and summarize previous results. Second, we define bias and variance and show how to measure them. Third, we discuss the conditions under which voting can reduce error and why the ECOC approach is a kind of voting. Fourth, we show that the ECOC method reduces both bias and variance while other methods such as bootstrap aggregating (Breiman, 1994) can only reduce variance. Finally, we report experiments that show that the differences in C4.5's bias errors across the various L problems depend on the non-local behavior of C4.5. Based on this understanding of the reasons for ECOC's success with C4.5, we can predict when it will succeed with other learning algorithms.

2 Definitions and Previous Work

The goal of supervised learning for classification is to learn a classification function $f(x)$ that takes a description x of an input object and classifies it into one of k classes: $f(x) \in \{c_0, \ldots, c_{k-1}\}$. To learn this classification function, a learning algorithm analyzes a set of training examples $\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_m, f(x_m))\}$. Each training example is a pair consisting of a description of an object, x_i, and its correct classification, $f(x_i)$. Each x_i is represented by a vector of n *feature values* that describe properties of x_i. We will refer to the feature values describing example x_i as $a_1(x_i), \ldots, a_n(x_i)$. We can view this vector of feature values as a point in an n-dimensional *feature space*.

Some learning algorithms, such as C4.5 (Quinlan, 1993a) and CART (Breiman, Friedman, Olshen, & Stone, 1984), can solve such k-way classification problems directly. However, many learning algorithms are designed to solve binary (2-class) classification problems. The error-correcting output coding (ECOC) technique studied in this paper is one of several techniques for converting a k-way classification problem into a set of binary classification problems. It works as follows.

Let $s_0, \ldots, s_{k-1}$ be k distinct binary strings of length L, chosen so that the Hamming distance between every pair of strings s_i and s_j is as large as possible. (The Hamming distance between two binary strings is the number of bit positions in which the strings differ.) We will call each string s_i the *codeword* for class c_i. Table 1 shows an example of a set of 10 such binary strings. The Hamming distance between each pair of these strings is always at least 7 bits.

Now, define L binary classification functions, $f_1, \ldots, f_L$ so that $f_j(x) = 1$ if $f(x) = c_i$ and the j-th bit of s_i is 1. Otherwise, $f_j(x) = 0$. These correspond to the columns of Table 1.

During learning, each of the f_j functions is learned by re-coding the examples to be $\{(x_1, f_j(x_1)), \ldots, (x_m, f_j(x_m))\}$ and applying a two-class learning algorithm to learn $\hat{f}_j$. The result of this is a set of L hypotheses, $\{\hat{f}_1, \ldots, \hat{f}_L\}$.

To classify a new example, x', we apply each of the learned functions $\hat{f}_j$ to x' to compute a vector of binary decisions $\hat{s} = \langle \hat{f}_1(x'), \ldots, \hat{f}_L(x') \rangle$. Then, we determine which codeword s_i is nearest to this vector (using the Hamming distance). The predicted value of $f(x')$ is the class c_i corresponding to the nearest codeword s_i. For example, suppose that the predicted outputs for x' are $\hat{s} = \langle 1,1,0,0,0,1,1,1,0,0,1,0,0,0,1 \rangle$. Table 2 shows the Hamming distances between this string of predictions and each of the rows s_i of Table 1. The string s_4 for class c_4 has the smallest Hamming distance (3), so we predict $\hat{f}(x') = c_4$.

The advantage of this scheme is that the codewords $\{s_1, \ldots, s_k\}$ constitute an error-correcting code. If

Table 1: 15-bit Error-correcting output code for a 10-class problem

Class i	Code Word (s_i)														
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

Table 2: Hamming distances between the string of predicted bits $\hat{s}$ and the codewords from Table 1

Class	Hamming Distance
c_0	6
c_1	9
c_2	7
c_3	6
c_4	3
c_5	6
c_6	10
c_7	7
c_8	5
c_9	8

the minimum Hamming distance between any pair of codewords is d, then any $\lfloor (d-1)/2 \rfloor$ errors in the individual f_j's can be corrected, because the nearest codeword will be the correct codeword.

By contrast, the standard approach to converting a k-way classification problem into a set of binary classification problems is to define one function f_i for each class, such that $f_i(x) = 1$ if $f(x) = c_i$ and zero otherwise (see Nilsson, 1965). We call this the *one-per-class* (OPC) method. During learning, a set of hypotheses, $\{\hat{f}_1, \ldots, \hat{f}_k\}$, is learned. To classify a new example, x', we compute the value of $\hat{f}_i(x')$ for each i. The predicted value of $f(x')$ is the class c_i for which $\hat{f}_i(x')$ is maximized. (This approach works best for learning algorithms that produce a probability or activation as the output.)

The step of mapping $\hat{s}$ to the nearest codeword can be extended to handle learning algorithms that produce activations or probabilities rather than simple classifications. In place of the Hamming distance, we can sum the absolute value of the difference between each component of $\hat{s}$ and the corresponding component of a codeword. This is the method that we use in all of our experiments.

Dietterich & Bakiri (1991, 1995) have shown that the

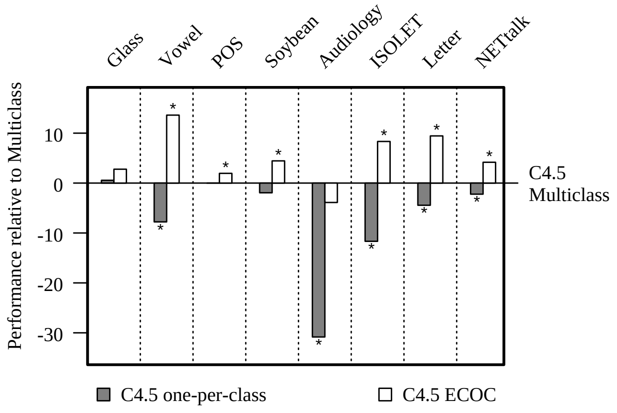

Figure 1: Performance of the one-per-class and ECOC methods relative to the direct multiclass method using C4.5. Asterisk indicates difference is significant at the 0.05 level or better. The glass, vowel, soybean, audiology (standardized encoding), ISOLET, letter recognition, and NETtalk data sets are all from the Irvine repository (Murphy & Aha, 1994). The POS task is to predict the part-of-speech of unknown words (Claire Cardie, personal communication) from their context.

error-correcting output coding technique works very well with the decision-tree algorithm C4.5. Figure 1 compares the performance of C4.5 in eight domains. Three configurations of C4.5 are compared: (a) multiclass, in which a single decision tree is constructed to make the k-way classification, (b) one-per-class, in which k decision trees are constructed, and (c) error-correcting output coding, in which L decision trees are constructed.

In this figure, the dark bars show the performance (in percent correct) of the one-per-class approach, and the light bars show the performance of the longest error-correcting code tested. Performance is plotted as the number of percentage points by which each algorithm differs from the multiclass approach. An asterisk indicates that the difference is statistically significant at the $p < 0.05$ level according to the binomial test for the difference of two proportions.

From this figure, we can see that the one-per-class method performs significantly worse than the multiclass method in five of the eight domains and is statistically indistinguishable in the remaining three domains. Much more important is the observation that the error-correcting output code approach is significantly superior to the multiclass approach in six of the eight domains and indistinguishable in the remaining two.

These results are quite exciting, but they do not explain *why* the ECOC method works. That is the goal of this paper.

3 Decomposing the Error Rate into Bias and Variance Components

Consider a regression algorithm R applied to a set S of training examples to produce an hypothesis $R(S) = \hat{f}_S$. Suppose we could draw a sequence of training sets $S_1, \ldots, S_l$, each of size m, and apply R to construct hypotheses $\hat{f}_{S_1}, \ldots, \hat{f}_{S_l}$. The *expected error* of R at point x is defined as its average squared error, where the average is taken over all of these training sets in the limit as l becomes large:

$$Error(R, m, x) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \left(\hat{f}_{S_i}(x) - f(x) \right)^2 .$$

The *ideal voted hypothesis*, $\hat{f}^*$, is defined as

$$\hat{f}^*(x) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \hat{f}_{S_i}(x).$$

The *bias* of algorithm R at x is defined as the (unsquared) error of this ideal voted hypothesis:

$$Bias(R, m, x) = \hat{f}^*(x) - f(x).$$

Finally, the *variance* of algorithm R is the expected value of the squared difference between the ideal voted hypothesis and each individual hypothesis:

$$Var(R, m, x) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \left[\hat{f}_{S_i}(x) - \hat{f}^*(x) \right]^2$$

A well-known theorem in regression states that

$$Error(R, m, x) = Bias(R, m, x)^2 + Var(R, m, x).$$

In words, the expected squared error of algorithm R on test data point x when trained on a sample of size m is equal to the sum of the squared error of the ideal voted hypothesis and the variance of each individual hypothesis with respect to this ideal voted hypothesis.

Extending notions of bias and variance to classification problems is not straightforward, and many alternative approaches are possible (Efron, 1978). Our approach, which we have not previously seen, is based on idealized voting. Let A be a classification learning algorithm, and let $\hat{f}_{S_i}$ be the hypothesis produced by A when trained on training set S_i. For a test set example x, $\hat{f}_{S_i}(x)$ is equal to one of k possible classes $\{c_0, \ldots, c_{k-1}\}$.

Define p_j to be the probability that $\hat{f}_{S_i}(x) = c_j$, where the probability is taken over all possible random training sets S_i of size m. The average error rate of algorithm A at x can be written as

$$Error(A, m, x) = 1 - p_j, \text{ for } c_j = f(x)$$

This is because if the correct class $f(x) = c_j$, then the average error rate on test point x is just 1 minus the probability that x is correctly classified.

The ideal voted hypothesis chooses the class with the highest probability—the class with the most votes from the individual $\hat{f}_{S_i}$'s:

$$\hat{f}^*(x) = \underset{c_j}{\text{argmax}}\ p_j .$$

We define the *bias* to be the error of the ideal voted hypothesis:

$$Bias(A, m, x) = \begin{cases} 0 & \text{if } \hat{f}^*(x) = f(x) \\ 1 & \text{if } \hat{f}^*(x) = f(x) \end{cases}$$

In other words, the bias at a particular data point x is either 0, if the ideal voted hypothesis classifies x correctly, or 1 if it makes an error.

We will define the *variance* to be the difference between the expected error rate and the ideal voted hypothesis error rate:

$$Var(A, m, x) = Error(A, m, x) - Bias(A, m, x).$$

This has the unintuitive property that the variance at a particular test set point can be negative. This occurs when the learning algorithm usually misclassifies point x incorrectly but occasionally gets it right. In that case the average error is reduced by the occasional "lucky" correct classifications. This is analogous to the situation in regression where the bias may be high and yet from time to time the regression algorithm may correctly predict the value of $f(x)$. Because regression uses the squared error, however, the variance is always positive.

For a test set of T examples, the expected error rate is $\frac{1}{T} \sum_{t=1}^{T} Error(A, m, x_t)$. The component of this due to bias is $\frac{1}{T} \sum_{t=1}^{T} Bias(A, m, x_t)$, and the component due to variance is $\frac{1}{T} \sum_{t=1}^{T} Var(A, m, x_t)$.

For artificial (simulated) learning problems, we can measure the bias and variance by directly simulating the definitions. We start by defining a target function f over some input space $\mathbf{X}$ and some probability distribution $D(\mathbf{X})$. We then draw a series of training sets of size m by sampling according to D and labeling each example according to f. We also draw a large test set (possible consisting of all of $\mathbf{X}$). Our goal is to estimate the p_j's for each x in the test set. We do this by training C4.5 on each of the training sets and then computing for each test example the proportion of times that it was classified into each of the k classes. We can then compute the expected error rate, bias, and variance directly from the definitions.

We followed a slightly different procedure that gave better results experimentally. Instead of producing a simple classification of each test example, C4.5 can produce a class probability vector that gives the probability (according to $\hat{f}$) that the example belongs to each of the k classes. We sum these class probability vectors (component-wise) for each of the hypotheses, $\hat{f}_{S_i}$, constructed from the series of training sets to obtain the voted class probability vector, which we then normalize so that the probabilities sum to 1. The ideal voted hypothesis classifies each test example using the class whose voted probability is maximum.

We have made these computations for C4.5 (Release 1, no pruning, no windowing) for the simple 6-class learning problem shown in Figure 2. We constructed 200 training sets each of size 200 by sampling uniformly with replacement from the region $\mathbf{X} = [0, 15] \times [0, 15]$. A test set containing 7,670 examples, which focused primarily on the decision boundaries, was also constructed. On this test set, the mean error rate for the 200 replications of C4.5 was 0.3695. Of this, the bias component was 0.2331 and the variance component was 0.1364. Figure 3 shows the bias errors.

As Breiman (1994) has shown, the error rate of algorithms like C4.5 can be significantly reduced by voting multiple bootstrap runs. We measured the bias and variance of Breiman's procedure by voting 200 bootstrap replicates of multiclass C4.5 on each of the 200 training sets. This reduced the mean error from 0.3695 to 0.3452. However, the bootstrap voting actually *increases* the bias component from 0.2331 to 0.2695 and decreases the variance component from 0.1364 to 0.0757. Hence, we see that bootstrap aggregation only reduces variance, not bias.

This makes sense, because bootstrap aggregation can be viewed as a way of approximating the ideal voting algorithm. Unlike ideal voting, where each training set S_i is drawn at random from the entire input space, each bootstrap replicate must reuse a randomly-selected subset of the given training set S. To the extent that bootstrap aggregation approximates ideal voting, the error rate of bootstrap aggregation will approach the error rate of ideal voting, which is, by our definition, the bias. Hence, bootstrap aggregation cannot reduce bias, but it can reduce variance.

This is true for any homogeneous voting scheme, where the same algorithm is applied multiple times to a given data set. Multiple votes can produce better approximations of the ideal probabilities p_j, but they cannot change their true values. Hence, homogeneous voting can reduce variance but not bias.

4 ECOC and Voting

To apply the error-correcting output coding method to the problem in Figure 2, we constructed a 31-bit error-correcting code (via the "exhaustive" method described in Dietterich & Bakiri, 1995). We can understand this error-correcting code graphically by considering Figure 4, which shows the decision boundaries from Figure 2. Each decision-boundary segment has been given a unique label (e.g., B35a is one of boundaries separating class c_3 from class c_5). When we apply multiclass C4.5 to this problem, it is forced to learn all of the decision boundaries simultaneously. In contrast, the ECOC method has three important properties. First, each individual binary function f_l must only learn *some* decision boundaries, and this set of decision boundaries varies from one f_l to another. Second, each boundary is learned many times. Third, when a predicted binary string is "decoded" by the error-correcting code procedure (i.e., by mapping it to the nearest codeword), this is equivalent to a vote among those functions f_l that learned the relevant boundaries. Let us expand each of these observations.

To see why each individual binary function learns only some of the decision boundaries, consider function f_2. This function labels examples from classes c_0 and c_5 as 1 and all others as 0. Hence, it must learn the boundaries labeled B01, B02, B35a, B45, and B35b. The function f_{14}, on the other hand, labels examples from classes c_2, c_3, and c_5 as 1 and all others as 0. It must learn boundaries B02, B13a, B45, and B13b. Note that boundaries B02 and B45 must be learned by both f_2 and f_{14}.

It turns out that each boundary is learned exactly 16 times in this 31-bit code. This is because in our 31-bit code, each codeword is Hamming distance 16 away from every other codeword. That means a given point x must cross 16 learned boundaries to move from one class to another (i.e., 16 functions f_l must change their classifications).

This also explains why the decoding step is a form of voting. Suppose that 6 of the learned $\hat{f}_l$'s misclassify a test point x'. There are still 10 learned $\hat{f}_l$'s that correctly classified it (i.e., that placed x' on the correct side of their decision boundaries), so x' will be correctly classified. The nearest codeword in Ham-

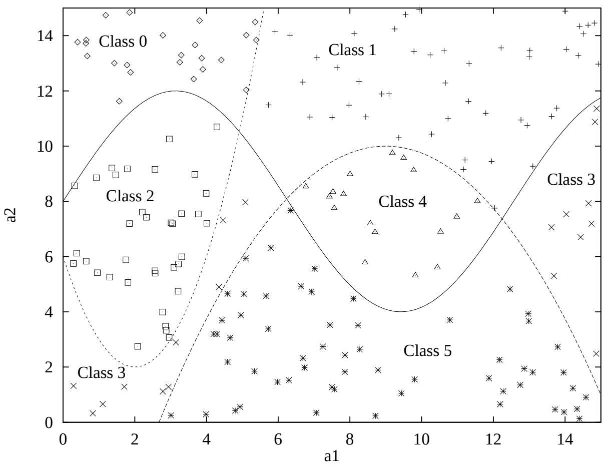

Figure 2: A six-class problem with 200 training examples.

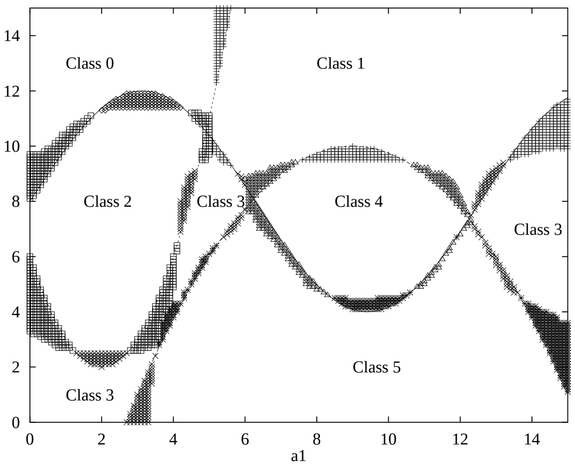

Figure 3: Bias errors of C4.5 on the problem from Figure 2 estimated from 200 replications.

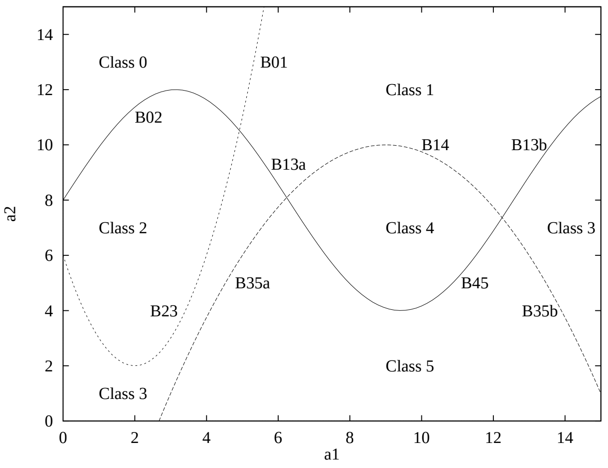

Figure 4: The decision boundaries for the learning problem from Figure 2. Each boundary is given a unique name.

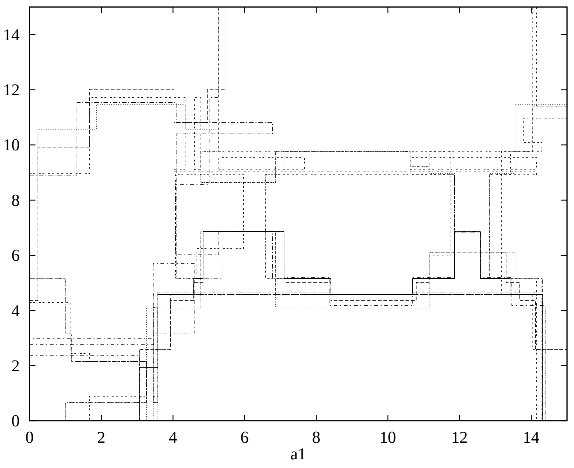

Figure 5: Superimposed decision boundaries learned by functions f_1 through f_8 on our example learning problem.

ming distance corresponds to the class whose "territory" can be reached by crossing the fewest number of learned boundaries (i.e., changing the fewest number of bits). Hence, the learned boundaries vote to classify x'.

Figure 5 shows the boundaries learned by functions f_1 through f_8 of our 31-bit error correcting output code (the remaining 23 functions were omitted to improve readability). We can see that each of the boundaries has been learned approximately 4 times and that the various learned boundaries are *not identical*. A point x' that is misclassified by only one of these learned boundaries would still be classified correctly after being decoded.

This decision boundary perspective on error-correcting output coding also explains why the one-per-class approach works poorly. In the OPC method, each boundary is learned twice. For example, boundary B02 is learned once when we attempt to discriminate class c_0 from all of the other classes, and it is learned again when we try to discriminate class c_2 from all of the other classes. With only two hypotheses along each boundary participating in a "vote", there is no way to recover from any errors.

As we established in the previous section, voting only improves performance if the errors made by the various "voters" are not highly correlated. This observation holds for ECOC as well. The errors made by the various $\hat{f}_l$'s must be uncorrelated. Otherwise, all of the "bad" votes near a decision boundary will be the same, and points will be misclassified.

From this discussion, we can see that the effectiveness of ECOC depends on whether the errors made in the different bit positions of the code are relatively uncorrelated.

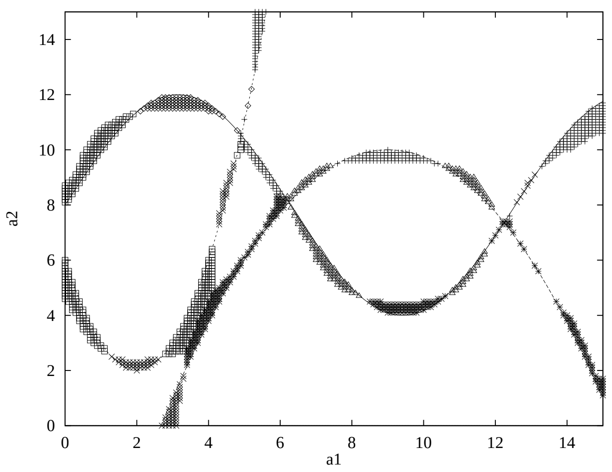

Figure 6: Bias errors of 31-bit ECOC configuration of C4.5.

5 ECOC Reduces Variance and Bias

We measured the bias and variance of the ECOC approach on the problem from Figure 2. With ECOC, the mean error was reduced from .3695 to .3450. This is virtually the same reduction obtained by bootstrap aggregating (where bias increased and variance decreased). In this case, however, the improvement results from reductions in both the bias (from .2331 to .2176) and variance (from .1364 to .1274). Hence, this shows that ECOC can reduce both bias and variance. The variance reductions are to be expected, since ECOC is a form of voting. The reduction in bias is more interesting. The bias errors of ECOC are plotted in Figure 6. The figure can be compared with Figure 3 to see where the bias has been reduced.

We also measured and plotted the bias errors of individual $\hat{f}_l$'s. Figure 7 shows the bias errors for $\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$, and $\hat{f}_4$. We can see that the bias errors made are quite different (e.g., along the B35a, B45, and B35b decision boundaries). Hence, they can be corrected by the error-correcting code.

The variance reduction for C4.5 using ECOC is not very large in comparison to the variance reduction (from 0.1364 to 0.0757) that we obtained by performing Breiman's 200-replicate bootstrap. One explanation is that on this problem C4.5 is only voting 16 decision trees along each boundary; this may not reduce variance as much as voting 200 trees. This suggests a composite strategy in which we construct each $\hat{f}_l$ by performing a 200-fold bootstrap and vote. The results of these 31 separate "elections" can then be combined (by the decoding step of ECOC) to classify each test example. Measurements confirmed this prediction: The mean error rate was reduced to .3138 (bias .2037 and variance .1102). Hence, the ECOC strategy can be fruitfully combined with Breiman's bootstrap voting to achieve even better improvements in performance.

6 Bias Differences are Caused by Non-Local Behavior

The fact that we observe uncorrelated errors in the $\hat{f}_l$'s leaves open the question of *why* these errors are uncorrelated. Examination of plots such as Figure 7 suggests that C4.5 makes different bias errors because of non-local interactions between training examples. Consider, for example, the bias errors in bits f_1 and f_2 in the Figure 7 in the region where $a_1(x) = 12$ and $a_2(x) = 7$. These bias errors result from the decision by C4.5 to approximate the B45 boundary by a horizontal line. It places this line near $a_2(x) = 7$, and hence "chops off" the two "pointed regions" where Class 5 meets Class 1. However, these bias errors are not made in bits f_3 and f_4, because now C4.5 must also learn other boundaries including B14, B13a, and B13b. C4.5 chooses to make some vertical splits instead, so the bias shifts dramatically. In short, because C4.5 places splits based on global heuristics, the presence of different distant boundaries in the different f_l's leads to different bias errors.

To test this hypothesis, we implemented a version of C4.5 that uses only local information to place splits. Specifically, our local-C4.5 works like a nearest neighbor algorithm: for each test example x', the k nearest neighbors are identified and used to construct a decision tree. This tree is applied to classify x' and then discarded. Because these local trees only use local information, we predict that the bias errors in different f_l's will become correlated with one another. This should eliminate the ability of ECOC to correct bias errors.

Figure 8 shows bias reduction obtained by applying this local C4.5 algorithm in the multiclass and ECOC configurations (with 200 replicates to measure the bias and mean error). We can see that if 9 or fewer neighbors are given to C4.5, the bias rate of ECOC is the same as the bias rate of multiclass C4.5, so there is no bias reduction from error correction. However, once we provide 15 or more neighbors, ECOC is clearly correcting some bias errors and improving classification accuracy.

We conclude that the lack of correlation in the bias errors of the individual f_l functions results from the geometry of the decision boundaries. This can vary from one learning problem to another, so it is difficult to make a general statement about the degree to which the bias errors in the f_l's will be uncorrelated. The observed experimental superiority of ECOC suggests that suitable decision boundary geometry arises quite often in practice.

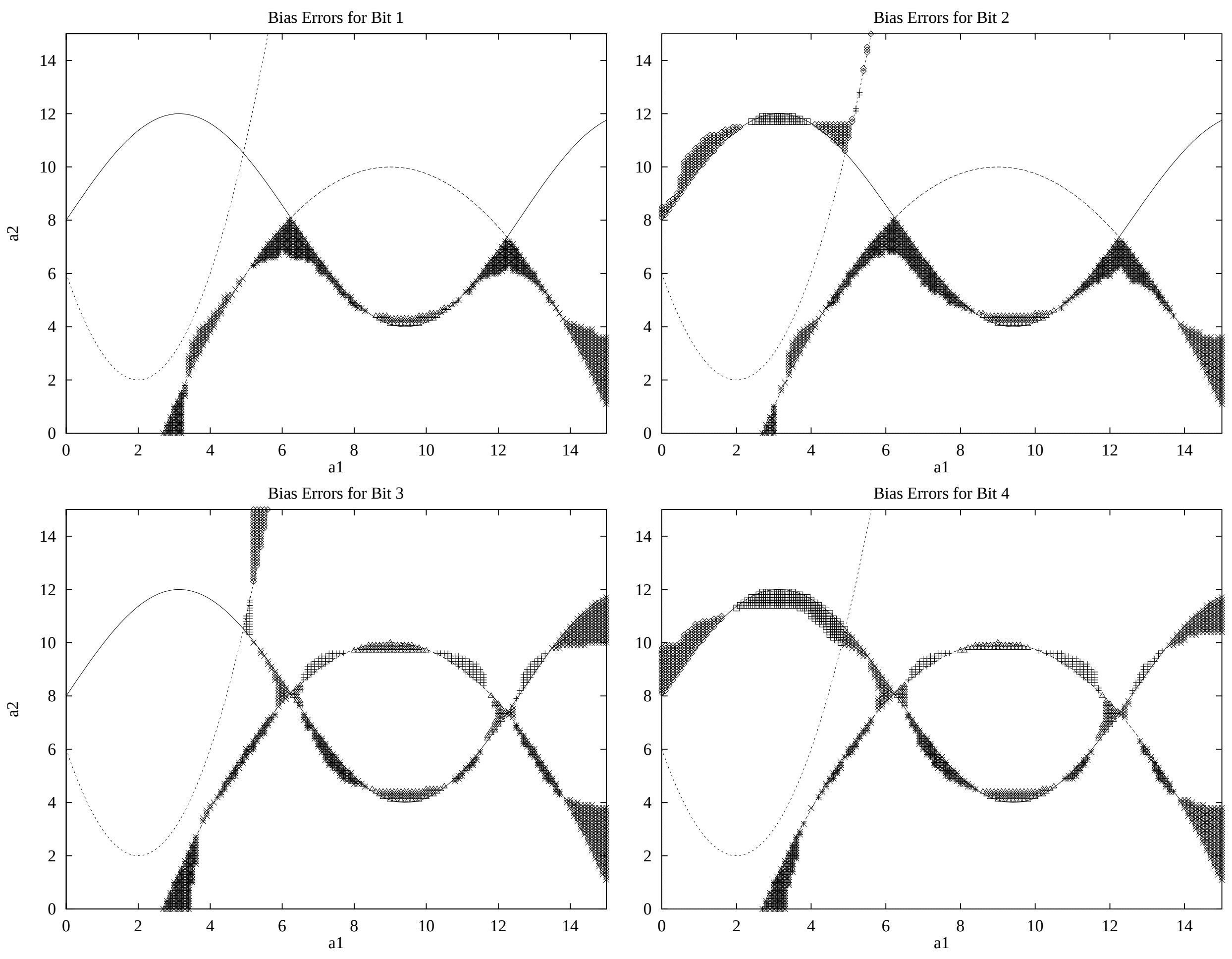

Figure 7: Bias errors for ECOC bit functions f_1, f_2, f_3, and f_4 measured from 200 replications of the problem in Figure 2

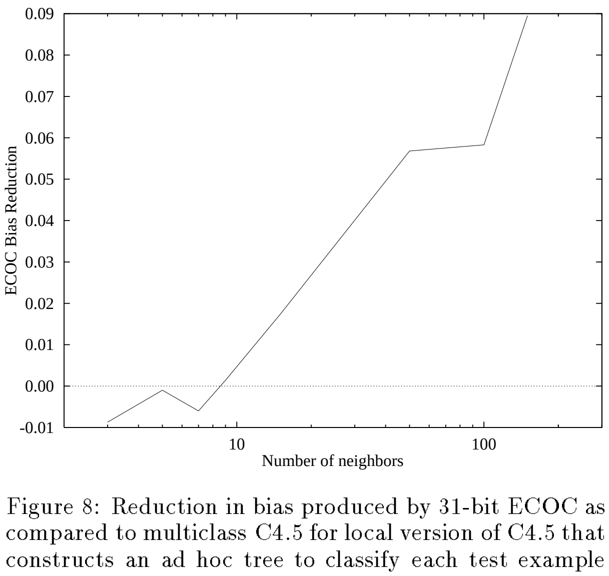

Figure 8: Reduction in bias produced by 31-bit ECOC as compared to multiclass C4.5 for local version of C4.5 that constructs an ad hoc tree to classify each test example using its k nearest neighbors. Note the log scale on the horizontal axis.

7 Discussion and Conclusions

This paper has shown that the error-correcting output code approach to multiclass problems is a kind of voting. However, unlike homogeneous voting, ECOC can reduce bias errors as well as variance errors if the bias errors in the individual voting functions, f_l, are somewhat uncorrelated. The experiments in the previous section showed that these bias errors are somewhat uncorrelated in our simulated problem (Figure 2) and that this results from non-local interactions between decision boundaries.

From these experiments, we can also predict that other algorithms with non-local behavior (such feed-forward sigmoidal networks and perceptrons) will also benefit from error-correcting output coding. However, algorithms such as k-nearest neighbor and radial basis functions, which are highly local, will not benefit much from error-correction. This prediction is borne out by the experiments reported in Wettschereck and Dietterich (1992), where ECOC gave only a small improvement with the generalized radial basis function (GRBF) algorithm.

The error-correcting output code approach imposes a distributed output representation on the learning algorithm. Hence, the results of this study may help explain why distributed representations work well.

An important open problem for future research is to find ways of converting ECOC hypotheses (and other collections of "voted" hypotheses) into representations that are smaller and more efficient to evaluate. In some applications, the cost of storing and evaluating a large number of decision trees (or neural networks) would make it infeasible to apply the ECOC approach. Can we obtain the advantages of voting without actually having to hold elections?

Acknowledgements

The authors thank Leo Breiman and Rob Tibshirani for helpful pointers. The authors gratefully acknowledge the support of the NSF under grants IRI-9204129 and CDA-9216172.

References

Bakiri, G. (1991). Converting English text to speech: A machine learning approach. Tech. rep. 91-30-2, Oregon State University, Corvallis, OR.

Bates, J. M., & Granger, C. W. J. (1969). The combination of forecasts. *Op. Res. Qtly.*, *20*, 319–325.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth.

Breiman, L. (1994). Bagging predictors. Tech. rep. 421, Dept. of Statistics, Univ. of Cal., Berkeley, CA.

Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proc. 10th Intl. Conf. Mach. Learn.*, pp. 17–24 San Francisco, CA. Morgan Kaufmann.

Clemen, R. T. (1989). Combining forcasts: A review and annotated bibliography. *Int. J. of Forecasting*, *5*, 559–583.

Dietterich, T. G., & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc. AAAI-91*, pp. 572–577. AAAI Press/MIT Press.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *J. of Art. Int. Res.*, *2*, 263–286.

Efron, B. (1978). Regression and ANOVA with zero-one data: Measures of residual variation. *J. Am. Stat. As.*, *73*(361), 113–121.

Hampshire II, J. B., & Waibel, A. H. (1990). A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Trans. Neural Networks*, *1*(2), 216–228.

Hansen, L., & Salamon, P. (1990). Neural network ensembles. *IEEE PAMI*, *12*, 993–1001.

LeBlanc, M., & Tibshirani, R. (1993). Combining estimates in regression and classification. Tech. rep., Dept. of Statistics, Univ. of Toronto.

Makridakis, S., & Winkler, R. L. (1983). Averages of forecasts: Some empirical results. *Management Sci.*, *29*(9), 987–996.

Meir, R. (1994). Bias, variance, and the combination of estimators; The case of linear least squares. Tech. rep., Technion, Israel.

Murphy, P., & Aha, D. (1994). UCI repository of machine learning databases [machine-readable data repository]. Univ. of Cal., Irvine.

Nilsson, N. J. (1965). *Learning Machines.* McGraw-Hill, New York.

Perrone, M. P. (1993). *Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization.* Ph.D. thesis, Brown Univ.

Perrone, M. P. (1994). Putting it all together: Methods for combining neural networks. In Cowan, J. D., Tesauro, G., & Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, 6, 1188–1189. Morgan Kaufmann, San Francisco, CA.

Perrone, M. P., & Cooper, L. N. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In Mammone, R. J. (Ed.), *Neural networks for speech and image processing.* Chapman and Hall.

Quinlan, J. R. (1993a). *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA.

Quinlan, J. R. (1993b). Combining instance-based and model-based learning. In Utgoff, P. E. (Ed.), *Proc. Int. Conf. on Mach. Learn.* San Francisco, CA. Morgan Kaufmann.

Schapire, R. E. (1990). The strength of weak learnability. *Mach. Learn.*, *5*(2), 197–227.

Wettschereck, D., & Dietterich, T. G. (1992). Improving the performance of radial basis function networks by learning center locations. In Moody, J. E., Hanson, S. J., & Lippmann, R. P. (Eds.), *Advances in Neural Information Processing Systems*, 4, 1133–1140. Morgan Kaufmann, San Francisco, CA.

Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *J. of Mol. Biol.*, *225.*