Why Error-Correcting Output Coding Works

Eun Bae Kong
ebkong@research.cs.orst.edu
Thomas G. Dietterich
tgd@cs.orst.edu
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Draft of October 6, 1994.

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 empirical investigation of why the ECOC technique works, particularly when employed with decision-tree learning methods. It concludes that an important factor in the success of the method is the nearly random behavior of decision tree algorithms near the root of the decision tree when applied to learn difficult decision boundaries. The results also show that deliberately injecting randomness into decision tree algorithms can significantly improve the accuracy of "voting" methods that combine the guesses of multiple decision trees to make classification decisions.

1 Introduction

Error-correcting output coding (ECOC) is a method for converting a k-class supervised learning problem into a large number L of two-class supervised learning problems. Any learning algorithm that can solve two-class problems, such as the decision tree algorithm C4.5 (Quinlan, 1992), 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, 1994) 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.

This paper shows that the ECOC strategy can be viewed as a compact form of "voting". 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.

The paper explores three hypotheses to explain why these errors are so uncorrelated: the resampling hypothesis, the decision-boundary alignment hypothesis, and the random-algorithm-behavior hypothesis. Experimental tests show that the random-algorithm-behavior hypothesis is correct and that the other two hypotheses are incorrect. Furthermore, the random-algorithm-behavior hypothesis suggests a new technique for improving the performance of the standard C4.5

algorithm. The paper describes experimental tests of this technique, and shows that it gives performance nearly as good as the ECOC approach itself.

The remainder of this paper is structured as follows. First, we introduce our notation and define the error-correcting output coding method. Second, we briefly summarize previous results for error-correcting output coding. Third, we present two equivalent perspectives on error-correcting output coding: an information-theoretic perspective and a geometric perspective. Both perspectives will demonstrate the importance of learning binary functions with uncorrelated errors. Fourth, we describe each of our three hypotheses in detail. This is followed by a series of experiments designed to test the hypotheses. The paper concludes with a discussion of the experiments and their implications for various methods of combining guesses from multiple learned hypotheses.

2 Background

2.1 Definitions

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: $f(x) \in \{c_1, \ldots, c_k\}$. 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 typically represented by a vector of n *feature values* that describe various 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, 1992) 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 $s_1, \ldots, s_k$ be k distinct binary strings of length L. Choose these strings so that the Hamming distance between every pair of strings s_i and s_j is a 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* representing 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 supervised learning, each of the f_j functions is learned by re-coding the examples to be $\{(x_1, f_j(x_1)), (x_2, f_j(x_2)), \ldots, (x_m, f_j(x_m))\}$ and applying a binary learning algorithm. The result of this is a set of L hypotheses, $\{\hat{f}_1, \ldots, \hat{f}_L\}$.

To classify a new example, x', we compute a vector of binary decisions $\hat{s} = \langle \hat{f}_1(x'), \ldots, \hat{f}_L(x') \rangle$ by applying each of the learned functions $\hat{f}_j$ to x'. 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 $\hat{s} = \langle 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1 \rangle$. We now compute the Hamming distance between this string of predictions and each of the rows s_i of Table 1. The Hamming distances are shown in Table 2. 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 the minimum Hamming distance between any pair of codewords is d, then any $\lfloor (d-1)/2 \rfloor$

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

Class i	Code Word (s_i)														
	f_0	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}
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 $s_{x'}$ 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

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 $f(x) = i$ and zero otherwise (see Nilsson, 1965). We call this the *one-per-class* (or 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 one-per-class approach can be viewed as an application of the ECOC approach where the length of the code is equal to the number of classes, k. Each codeword has only one bit (the i-th bit) set. The Hamming distance between any pair of codewords is only 2, so no errors committed in any bit position can be corrected.

2.2 The C4.5 algorithm

In this paper, our main goal is to explain why error-correcting output coding works so well with decision-tree learning algorithms. Specifically, we employ the C4.5 algorithm (Quinlan, 1992). C4.5 constructs a decision tree by analyzing a collection of examples. An example decision tree is shown in Figure 1. Also shown is an equivalent set of nested **if** statements.

Recall that each example x_i is represented by a vector of features $\langle a_1(x_i), \ldots, a_n(x_i))\rangle$. Each internal node of the tree tests the value of one of the features to see if it is greater than some chosen constant value. If the test fails, control proceeds to the left child of the node. If the test succeeds, control proceeds to the right child of the node. At the leaf node, $\hat{f}(x)$, the predicted class of x, is assigned.

To construct a decision tree, C4.5 operates in two phases: (a) tree growth and (b) tree pruning. During the tree growth phase, C4.5 begins at the root of the tree and chooses the best single test—where a test is a combination of a feature (e.g., $a_2(x)$) and a constant value (e.g., 1.4). The feature and constant value are chosen to maximize a heuristic quantity called the "information gain ratio." The exact details and justification for this heuristic can be found in Quinlan (1992). Intuitively, the goal is to find the test that gives the most information about the identity of the true class of x, $f(x)$.

Once the best test is chosen, the training examples are divided into those examples that fail the test and those that satisfy the test. The algorithm is then called recursively on these two subsets of examples to create the two child subtrees of the root node.

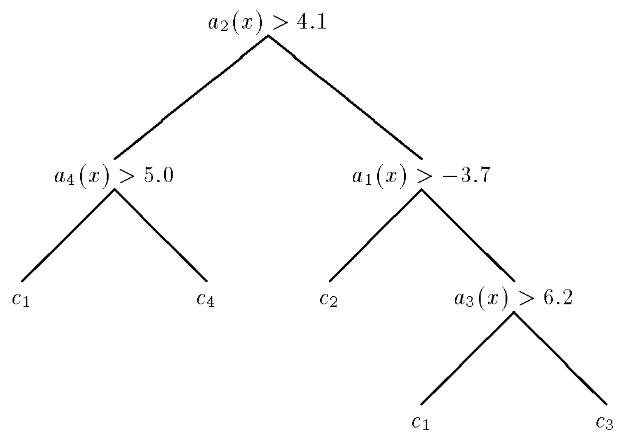
C4.5 continues to partition the data until all of the examples at a node are from the same class. At that point, it creates a leaf node that will assign $\hat{f}(x)$ to have that class.

Once the tree has been grown, C4.5 next proceeds to prune it. The pruning process involves deleting subtrees and replacing them with leaf nodes. These deletions are performed to avoid overfitting the training data (i.e., finding ad hoc patterns in the data that will not generalize to new examples). C4.5 employs a technique called *pessimistic pruning* that computes an estimate of the accuracy of a subtree and compares it to the estimated accuracy of the leaf node that would replace it. If the leaf node is estimated to be more accurate, then the subtree is replaced by the new leaf node.

This description of C4.5 captures the main properties of the algorithm, but it ignores many subtleties. Quinlan (1992) provides full details.

There are two points to note about C4.5 (and other decision-tree algorithms). First, the decision tree constructed by C4.5 can be viewed as partitioning the n-dimensional feature space into a set

(a)



(b)

```
if $a_2(x) > 4.1$
  then if $a_1(x) > -3.7$
          then if $a_3(x) > 6.2$
                  then $\hat{f}(x) := c_3$
                  else $\hat{f}(x) := c_1$
          else $\hat{f}(x) := c_2$
  then if $a_4(x) > 5.0$
          then $\hat{f}(x) := c_4$
          else $\hat{f}(x) := c_1$
```

Figure 1: C4.5 decision trees: (a) example tree and (b) an equivalent **if-then-else** expression.

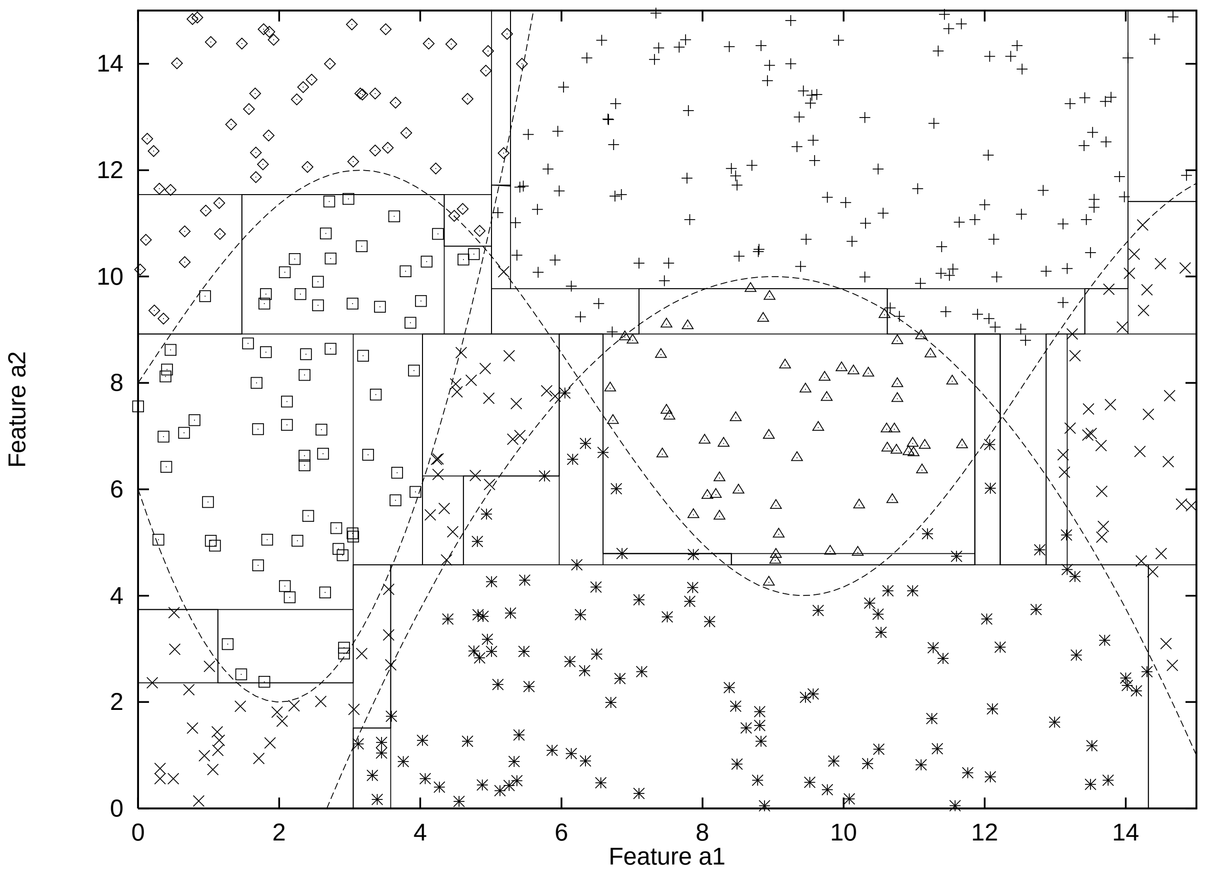


Figure 2: Decision boundaries constructed by C4.5 for a 6-class problem in a 2-dimensional feature space. Dashed lines show the true decision boundaries. Solid lines show splits chosen by C4.5. Various shapes show the training examples for the six classes.

of regions (one region for each leaf node). Each region is labeled with one of the k class labels, $c_1, \ldots, c_k$. The boundaries separating these regions are parallel to the coordinate axes of the feature space. Most of these boundaries separate regions belonging to different classes. A boundary (or part of a boundary) that separates regions belonging to two different classes is called a *decision boundary*. In learning problems where the true decision boundaries are not axis-parallel, C4.5 will need to construct a kind of "staircase approximation" by using many internal nodes.

Figure 2 shows a 6-class problem that we will use for illustration throughout this paper. Each of the 500 training examples has only two features, so it can be plotted as a point in this 2-dimensional feature space. The true decision boundaries are shown as dashed lines. The figure also shows (as solid lines) the boundaries constructed by C4.5 when it is trained on these 500 examples. Notice the "staircase approximations" to the curved decision boundaries.

The second point to note about C4.5 is that the amount of data available to guide the choice of tests is greatest for the choice of the root node and decreases toward the leaves. Several authors have pointed out that C4.5 often makes its worst errors at nodes derived from a small number of examples (Holte, Acker, & Porter, 1989). This is part of the rationale for the pruning phase of C4.5. From this observation, one might believe that C4.5 will make its most reliable choices of

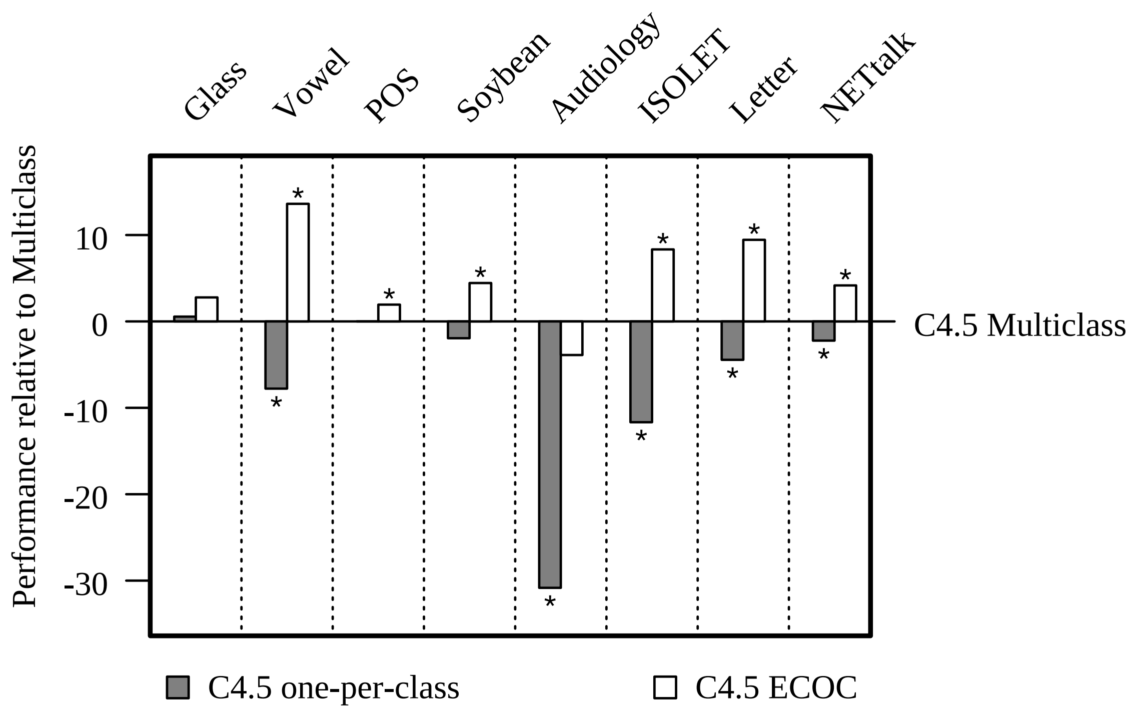


Figure 3: Performance of the one-per-class and ECOC methods relative to the direct multiclass method using C4.5. Asterisk indicates difference is significant at the 0.05 level or better. The glass, vowel, soybean, audiology (standardized encoding), ISOLET, letter recognition, and NETtalk data sets are all from the Irvine respository (Murphy & Aha, 1994). The POS task is to predict the part-of-speech of unknown words (Claire Cardie, personal communication) from their context.

decision-tree tests when it is still at or near the root of the tree (and hence, has a lot of training examples to analyze). However, it is easy to construct problems, such as parity, in which the information gain ratio heuristic cannot distinguish between tests of important features and tests of features that take on random values.

These two points will be important for understanding the remainder of the paper.

2.3 Previous work on error-correcting output coding

We now summarize the results of previous work on error-correcting output coding.

Dietterich & Bakiri (1994) have shown that the error-correcting output coding technique works very well with the decision-tree algorithm C4.5. Figure 3 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) the error-correcting output coding, in which L decision trees are constructed.

In this figure, the light bar shows the performance (in percent correct) of the one-per-class approach, and the dark bar shows the performance of the longest error-correcting code tested. Performance is displayed 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

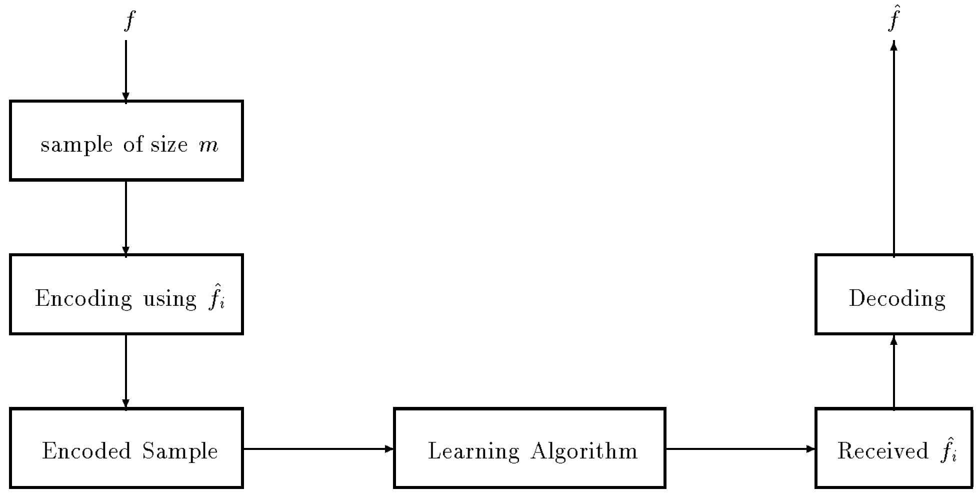


Figure 4: A model of inductive learning as a communications problem.

code approach is significantly superior to the multiclass approach in six of the eight domains and indistinguishable in the remaining two.

To understand why the ECOC approach works so well with decision trees, we begin by presenting two approaches to understanding error-correcting output coding.

3 Two Perspectives on Error-Correcting Output Coding

3.1 Error-correcting output coding and communication theory

Claude Shannon (1948) showed that whenever the transmission rate required by a communication system is less than the capacity of the communication channel, it is possible to achieve arbitrarily small error rates by using codes with redundancies. The key idea behind his channel coding theorem is that we can use the channel many times in succession independently. Assuming that errors introduced by the channel are independent, then a sufficiently long code can achieve an extremely small probability of error.

We can conceive of inductive learning as a communications problem (see Figure 4). Imagine a communications problem in which the sender wishes to communicate a function f to a receiver through a channel. The "channel" is a two-class inductive learning algorithm. The sender does not have direct access to the function f. Rather, the sender can only obtain a random sample of f of size m—that is, a set of m examples of the form $\langle x, f(x) \rangle$. The sender must communicate with the receiver by sending sets of training examples through the channel. The primary freedom the sender has is that the sender can encode these training examples and transmit them to the learning algorithm as many times as necessary. The sender and receiver can agree on the encoding and decoding procedures before the training examples are given to the sender.

One strategy the sender can use is the error-correcting output code approach. The training set is transmitted to the receiver L times, once for each bit position in the L-bit error-correcting code. When the training set is transmitted for the j-th time, the examples are labeled according to bit position j in the error-correcting code.

The receiver receives a sequence of L hypotheses, $\hat{f}_1, \ldots, \hat{f}_L$. The receiver constructs the com-

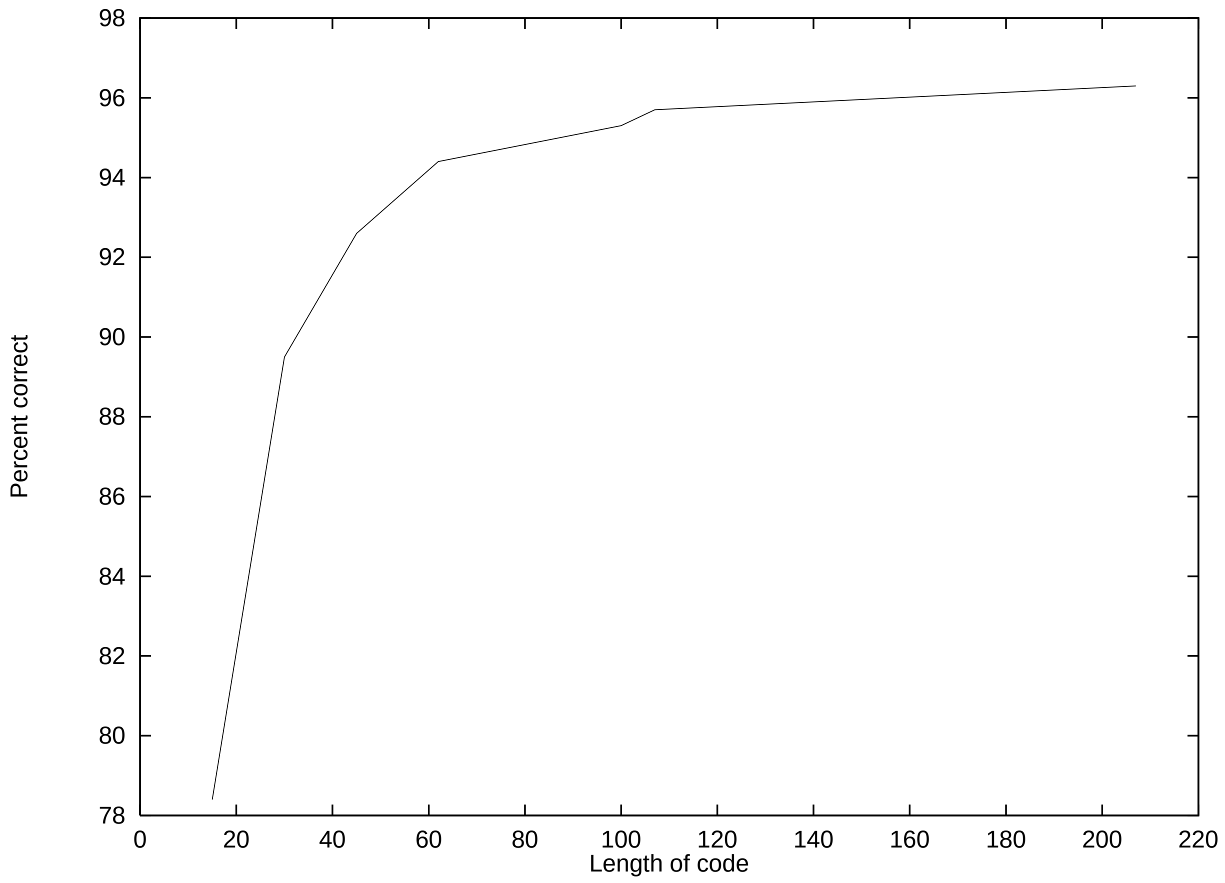


Figure 5: Performance of C4.5 on the Letter Recognition task as a function of the length of the error-correcting code.

bined hypothesis for f by using error-correcting decoding. To compute $\hat{f}(x')$ for a point x', the receiver evaluates $\hat{f}_1(x'), \ldots, \hat{f}_L(x')$ to construct a bit string s'. It then finds the codeword s_i that is nearest to s' in Hamming distance as sets $\hat{f}(x') = c_i$.

We can see that if the errors introduced by the learning algorithm in each of the hypotheses $\hat{f}_l$ are independent, then the error between $\hat{f}$ and f can be made as small as desired by making the code arbitrarily long.

Unfortunately, the errors committed when learning f_i and f_j are not independent. There are many ways to demonstrate this. First, experiments with the ECOC method have shown that as the code is lengthened, the accuracy of the learned function $\hat{f}$ ceases to improve beyond a certain point. Figure 5 shows this for C4.5 on the letter recognition task. At 203-bits, the accuracy of $\hat{f}$ is still increasing, but very slowly. It is unlikely to reach 100% correct regardless of the length of the code. Similar behavior is seen in all eight of the domains we have studied.

More direct evidence for dependencies between the errors of $\hat{f}_i$ and $\hat{f}_j$ is shown in Figure 6. The horizontal axis in this figure shows the Hamming distance between the *columns* of an error-correcting code (in this case, a code of length 207 for the 26-class letter recognition task). If the columns defining f_i and f_j are separated by an intermediate Hamming distance of around 13 bits, then those functions are very different from one another. Conversely, if the Hamming distance is

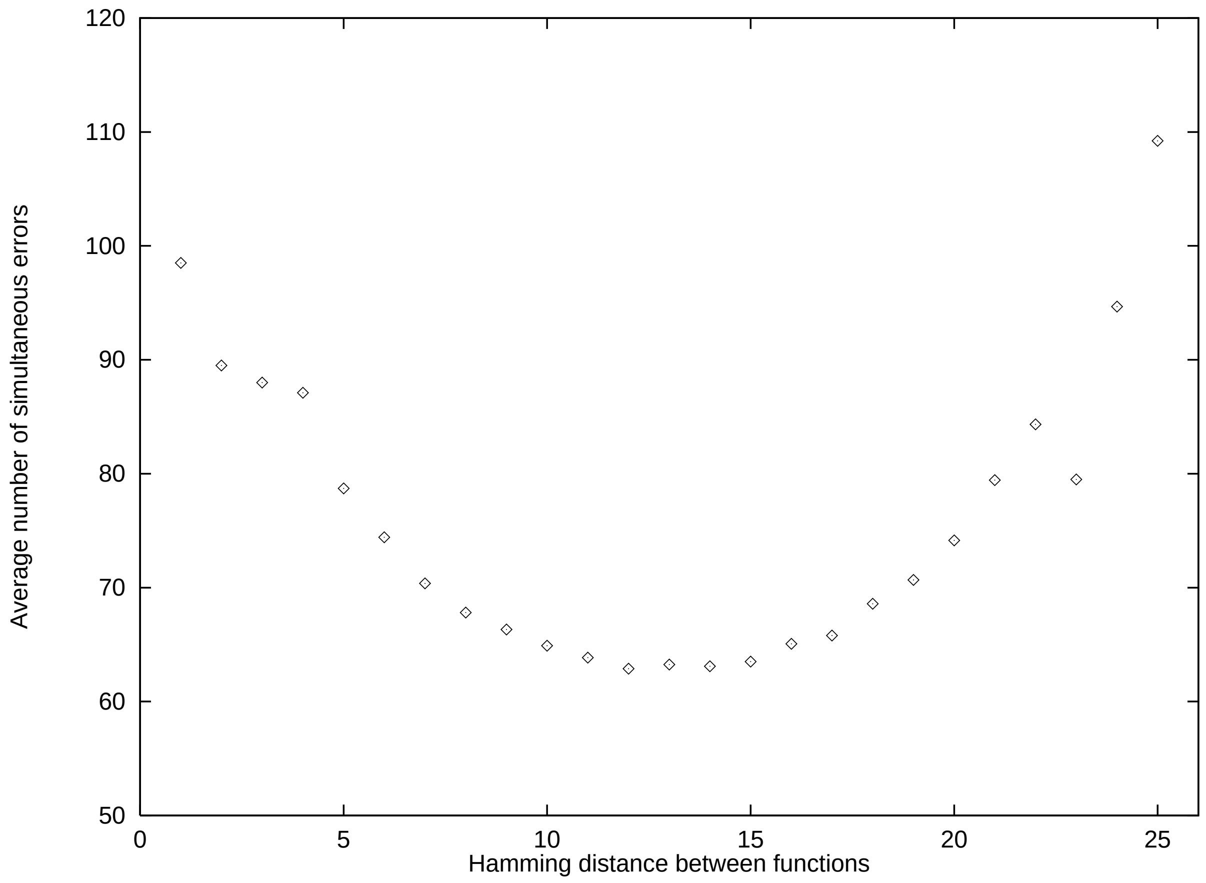


Figure 6: Number of simultaneous errors as a function of the Hamming distance between columns of the code. Each point shows the average number of simultaneous errors (on 1000 test examples) of all pairs of function $\hat{f}_i$ and $\hat{f}_j$ separated by the indicated Hamming distance.

near 0 or near 26, then the functions are nearly identical or nearly complementary. The vertical axis in the figure shows the average number of simultaneous errors committed by all pairs of functions f_i and f_j separated by a given Hamming distance. Clearly, functions that are more similar to one another make a larger number of simultaneous errors. Hence, the errors committed by pairs of functions are not independent—in fact, they can be predicted by knowing the Hamming distance between the functions. Nonetheless, even for the worst function pairs, the error correlation is quite low. The maximum number of simultaneous errors was less than 11% of the 1000 test examples.

3.2 Error-correcting output coding and decision boundaries

The second perspective on ECOC that we will consider is based on considering the behavior of ECOC along the decision boundaries in feature space. Consider Figure 7, which shows the decision boundaries from Figure 2. Each segment of the decision boundaries has been given a unique label (e.g., B35a is one of boundaries separating class c_3 from class c_5). When we applied the C4.5 multiclass algorithm to this problem, it was forced to learn all of the decision boundaries simultaneously.

However, if we consider the ECOC approach, each individual binary function, f_l, must only

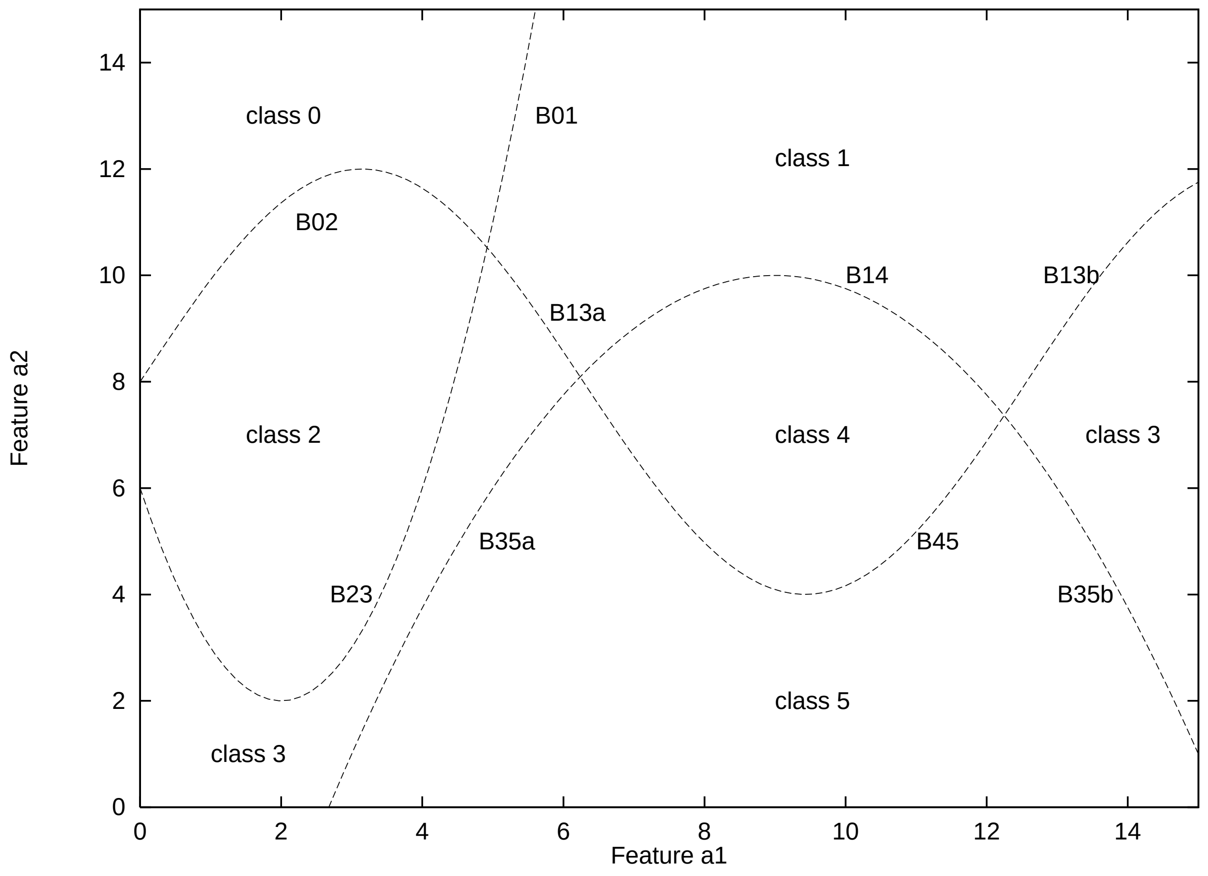


Figure 7: The decision boundaries for our example learning problem. Each boundary is given a unique name.

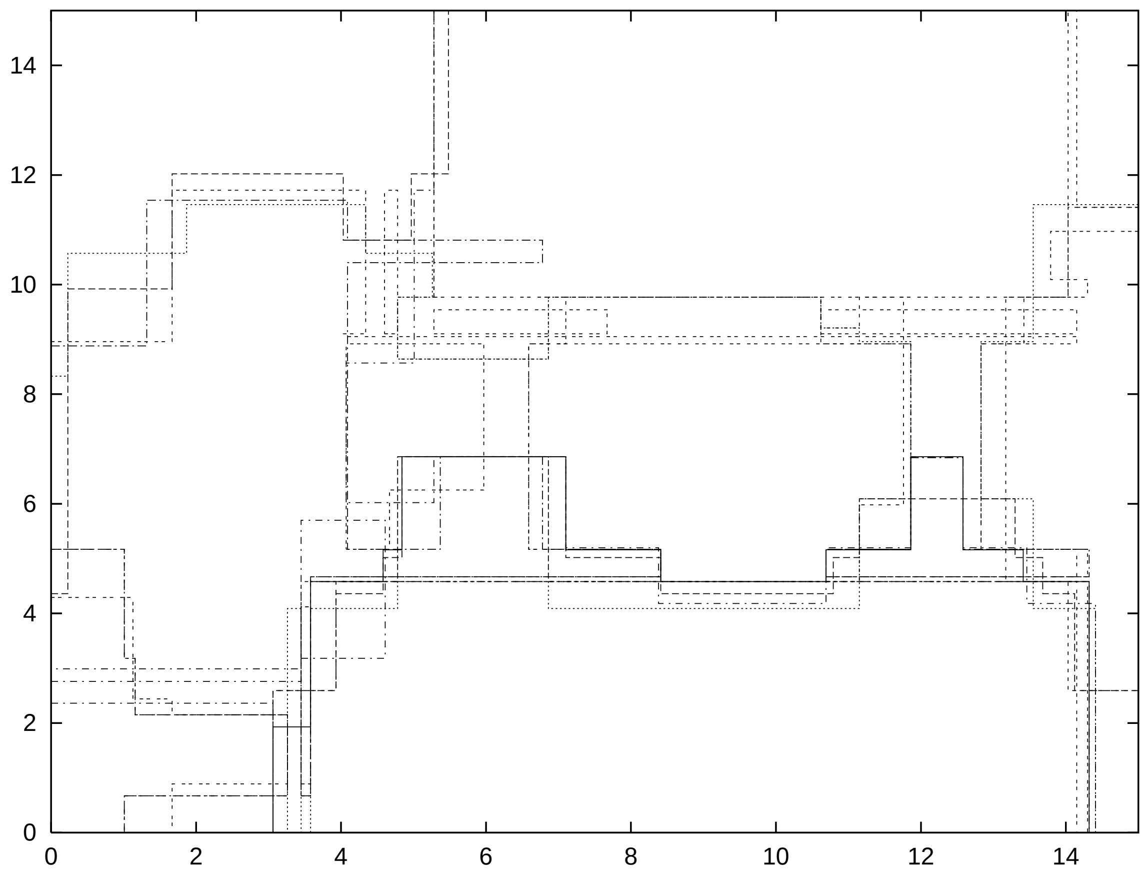


Figure 8: Superimposed decision boundaries learned by functions f_1 through f_8 on our example learning problem.

learn *some* of the decision boundaries, and this set of decision boundaries varies from one function f_l to another. For this 6-class problem, we have constructed an error-correcting output code of length $L = 31$ bits. The function f_1, for example, labels examples from classes c_0 and c_5 as 1 and all others as 0. Hence, it must learn the decision boundaries labeled B01, B02, B35a, B45, and B35b. The function f_{13}, 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. In fact, each boundary is learned exactly 16 times in this 31-bit code.

Figure 8 shows the decision 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 decision boundaries has been learned approximately 4 times, and that the various boundaries are *not identical.* This means that in the neighborhood of the decision boundary, different binary functions $\hat{f}_l$ are making different decisions. To classify a new point x' near the decision boundary, these various binary functions effectively "vote" to determine on which side of the boundary the x' should be placed. Our 31-bit code can correct 7 errors. Since each decision boundary is learned 16 times, this means that x' will be properly classified if the majority of the $\hat{f}_l$ have made the correct prediction.

This decision boundary perspective on error-correcting output coding also explains why the

Table 3: Comparison of three configurations of C4.5 on the data from Figure 2

Method	% Incorrect
C4.5 one-per-class	13.0
C4.5 multiclass	9.1
C4.5 31-bit ECOC	7.3

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 decision boundary participating in a "vote", there is no way to recover from any errors.

To establish baseline performance on this 6-class problem, we applied C4.5 in its multiclass configuration as well as C4.5 with the OPC and 31-bit ECOC configurations. A training set of 500 training points (and an independent set of 500 test points) was drawn uniformly at random. As Table 3 shows, the error-correcting code approach performs better than either of the other methods. The difference between OPC and multiclass C4.5 is statistically significant ($p < 0.05$); the difference between OPC and ECOC is also statistically significant ($p < 0.001$).

4 Three Hypotheses

Based on the two perspectives presented in the previous section, we can formulate three hypotheses to explain why the errors made by individual f_l functions are fairly independent.

4.1 The Resampling Hypothesis

Consider again Figure 7 and the decision-boundary segment B35a. Any function f_l that labels c_3 and c_5 differently must learn this boundary. But sometimes, when we are learning this boundary, c_2 and c_3 are grouped together, while at other times, c_2 and c_3 are labeled differently. When c_2 and c_3 are grouped together, all of their combined training examples lie to the left of segment B35a and can help a learning algorithm learn that segment. On the other hand, when c_2 is assigned a different label than c_3, only the training examples from c_3 are available to help learn boundary B35a. Let us define the set of training examples within a fixed distance D of a decision boundary to be the *relevant training examples* for that boundary.

From this example, we can see that the set of relevant training examples for a given decision segment changes from one f_l to another. The *resampling hypothesis* asserts that it is these different relevant examples that influence the learning algorithm and cause it to make uncorrelated errors in the different f_l's.

We call this the "resampling hypothesis", because it is related to the statistical sampling technique of bootstrapping (Efron & Tibshirani, 1991; Efron & Gong, 1983). In bootstrapping, a learning algorithm is repeatedly applied to various subsets of the training set. From a training set of size m, each subset is constructed by making m draws—with replacement—from the full training set. If the resampling hypothesis correctly explains ECOC, it suggests that a similar boost in performance could be obtained by drawing many bootstrap samples, applying C4.5 OPC to each sample, and taking a vote among the learned hypotheses.

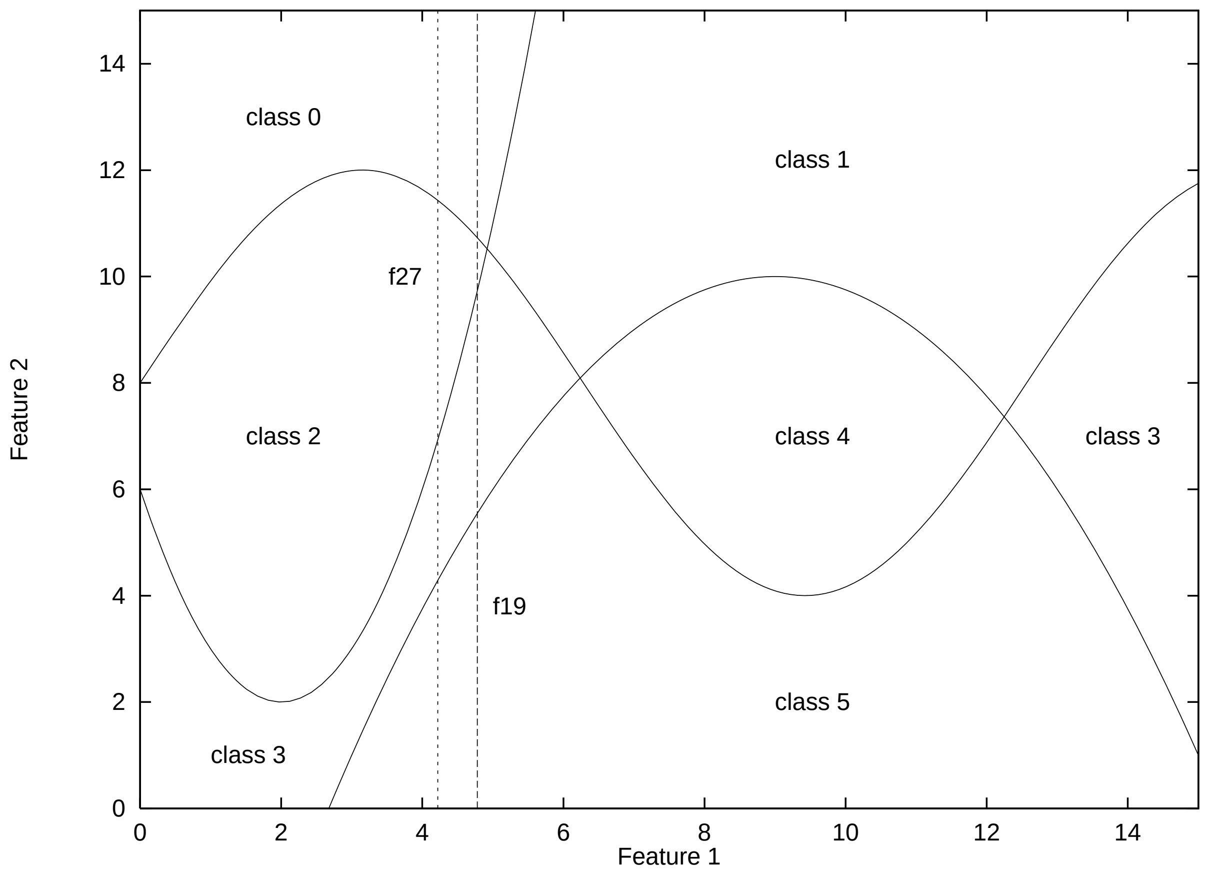


Figure 9: The root splits computed by C4.5 for function f_{19} (the right-most vertical line) and f_{27} (the left-most vertical line).

4.2 The Decision-Boundary Alignment Hypothesis

The C4.5 algorithm is a global algorithm. By this we mean that the position of a decision boundary near any given point x' can be influenced by training examples that are very far away from that point. This is quite different from algorithms, such as the nearest neighbor algorithm, in which only the training examples nearest to x' influence the position of any nearby decision boundary.

The decision-boundary alignment hypothesis is based on the observation that different f_l's will be learning different sets of decision boundaries. Different global alignments of these boundaries may influence C4.5 to place its splits in different positions, and this could produce the observed low correlation among the errors in the $\hat{f}_l$'s.

Consider again Figure 7. Suppose we are learning boundaries B01 and B35a for a boolean function f_l. These boundaries are approximately aligned vertically, so C4.5 will be encouraged to choose a vertical split located along these segments. In fact, for f_{19}, the classes c_1, c_4, and c_5 are grouped together, and Figure 9 shows the first split chosen by C4.5.

By contrast, if we consider function f_{27}, it groups together classes c_1, c_3, c_4, and c_5. Figure 9 shows the first split chosen by C4.5 in this case too. Notice that the position has shifted to the left. Hence, along boundary B01, changes in a distant decision boundary (e.g., B23 instead of B35a) have changed the position of C4.5's decision boundary.

To test this hypothesis, we could force the learning algorithm to be local by considering only training examples near the decision boundary. Specifically, suppose that we delay performing any runs of C4.5 until we receive a test example x' to classify. Then, we compute a set of the r nearest points to x' and run C4.5 (with ECOC) on those points to classify x'. According to this boundary alignment hypothesis, such a "local" C4.5 with ECOC would not exhibit the performance advantage over C4.5 OPC that we have observed with global C4.5.

4.3 The Random-Algorithm-Behavior Hypothesis

Our third hypothesis is based on the observation that the decision boundaries learned by ECOC are more complex than the boundaries learned in the one-per-class configuration of C4.5. The random-algorithm-behavior hypothesis asserts that these boundaries are so complex that the behavior of the learning algorithm near the root of the decision tree becomes random, and this high level randomness creates the uncorrelated errors.

There is a large amount of evidence to show that the decision boundaries learned by ECOC are more complex than the boundaries learned by the OPC configuration. In our simple problem in Figure 7, the OPC method must learn only 18 boundary segments (each of the nine segments is learned twice), for an average of 3 segments per decision tree. By comparison, ECOC must learn each segment 16 times (for a total of 144 segments). This gives an average of 4.6 segments per decision tree.

Experimentally, we can compare the size and error rate of the individual binary decision trees learned by C4.5 in the OPC and ECOC configurations. The one-per-class configuration constructs trees with an average of 18.67 leaves per tree (after pruning). We can measure the error rate of these binary trees individually on the test set. The average individual error rate is 3.3%. In comparison, the ECOC configuration constructs trees with an average of 26.68 leaves per tree (after pruning) and an average individual error rate of 5.0%. From these statistics, we can see that the ECOC trees are more complex and have higher individual error rates (on average) than the OPC trees.

When C4.5 is confronted with a very difficult learning problem, none of the splits available at the root of the tree appears very promising, and many splits have nearly equal information gain ratio scores. Hence, the choice of the split at the root is somewhat random. The random-algorithm-behavior hypothesis claims that this randomness near the root of the tree influences subsequent split decisions, so that the staircase approximations constructed for the decision boundaries become uncorrelated.

This hypothesis predicts that if the behavior of C4.5 near the root is forced to be constant (i.e., independent of the data), then the errors between different f_l's will become strongly correlated. It also makes a prediction that has the potential for boosting the performance of multiclass and OPC C4.5. Suppose we construct several sets of OPC decision trees while deliberately injecting randomness into the split decisions made near the root of the decision trees. The random-algorithm-behavior hypothesis predicts that these multiple OPC trees will make fairly uncorrelated errors, and hence, if they are combined by voting, a substantial performance improvement will be obtained.

5 Experimental Tests of the Hypotheses

5.1 Comparison of Bootstrap OPC with ECOC

To test the resampling hypothesis, we generated eight random subsets of the training set (for our example problem from Figure 7), and trained one collection of one-per-class decision trees on each of these training sets. We chose this 8-fold bootstrap so that each decision boundary would be

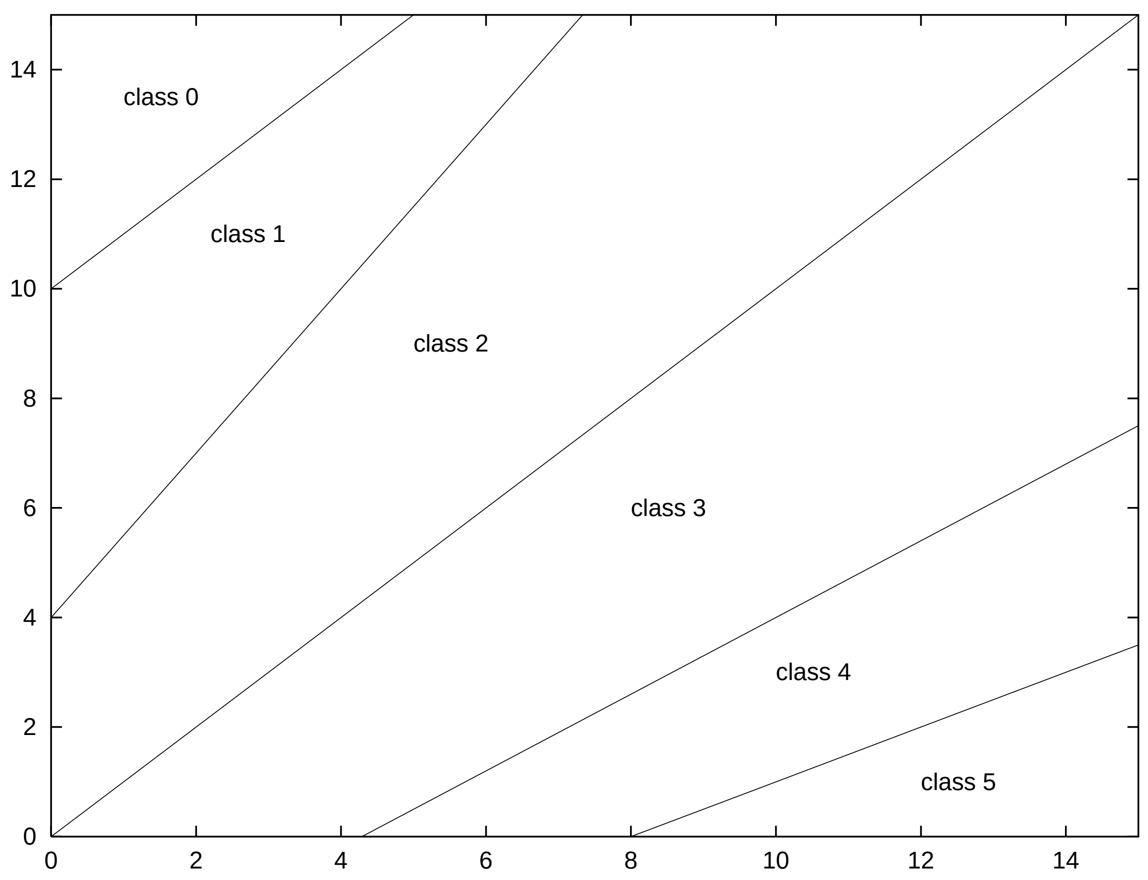


Figure 10: A synthetic problem with no alignments among decision boundaries

learned an equivalent number of times in the OPC and ECOC configurations. Each collection of OPC trees learns the 9 decision boundaries twice, so eight replications gives 144 decision boundary hypotheses—the same number as the 31 ECOC trees.

With the 8-fold bootstrap, the error rate of OPC decreased from 13.0% to 10.0%. This is still substantially worse than the 7.3% error obtained by ECOC. Even with a 25-fold bootstrap, the error rate for OPC voting only decreased to 9.7%. From this experiment, we conclude that while the boostrap hypothesis may explain some of the difference between OPC and ECOC, it does not explain all of the performance difference.

5.2 Tests of the Decision-Boundary Alignment Hypothesis

To test the decision-boundary alignment hypothesis, we constructed a problem in which none of the decision boundaries are aligned (see Figure 10). Despite this absense of alignments among decision boundaries, Table 4 shows that ECOC still works much better than the one-per-class method and slightly better than multiclass C4.5 on this problem.

As we suggested above, another way of testing the decision-boundary alignment hypothesis is to modify C4.5 to make it a local, lazy algorithm. No analysis of the training data is performed until a test example, x' is presented. Then, we compute the r training example data points nearest to x' (in Euclidean distance) and construct a training set containing only these data points. C4.5 is then trained using this very small training set (and using either the OPC or ECOC configuration).

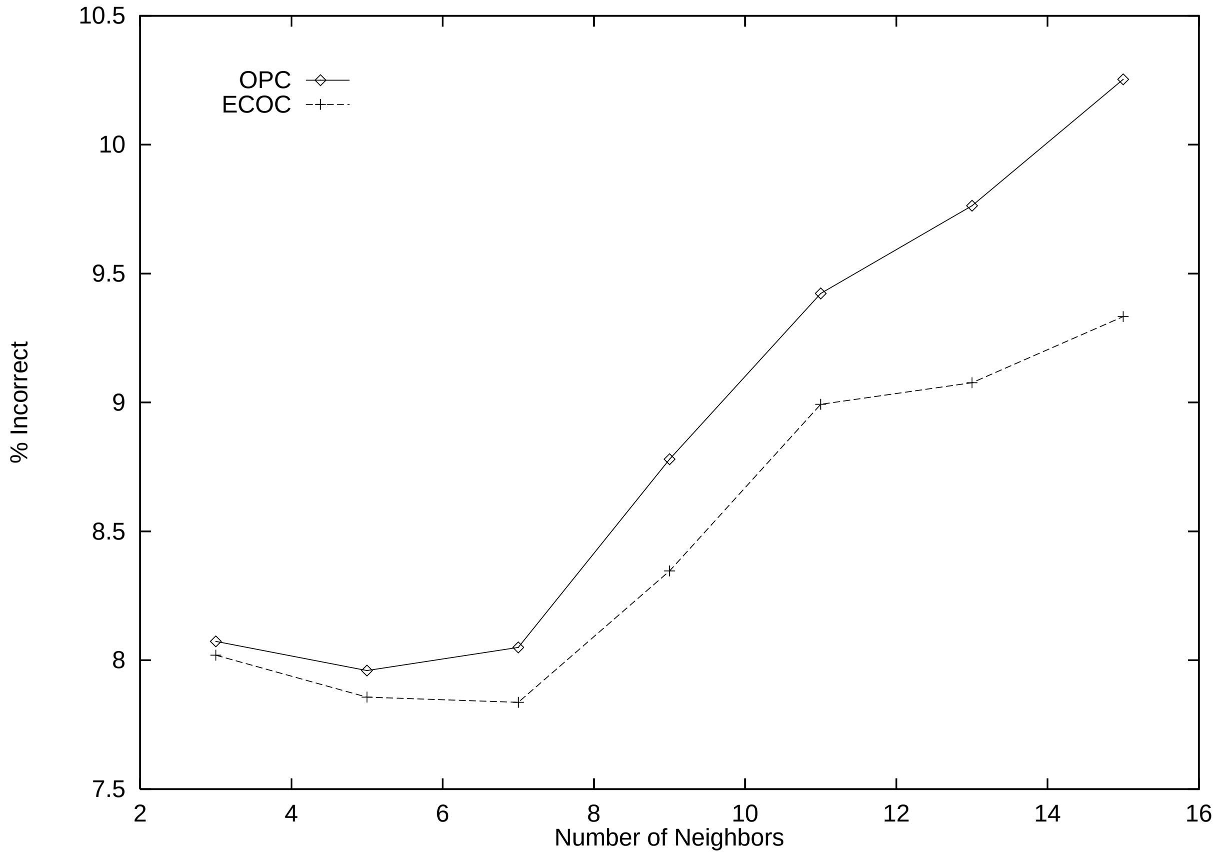


Figure 11: Performance of local versions of C4.5. Decision trees for either the one-per-class (OPC) or 31-bit error-correcting output code (ECOC) configuration were trained on the r nearest neighbors of each test data point.

Table 4: Performance on the problem from Figure 10

Method	% Incorrect
C4.5 multiclass	11.2
C4.5 one-per-class	17.2
C4.5 31-bit ECOC	10.6

The test point x' is then classified using the resulting decision trees. For each test data point, a set of local decision trees must be produced in this fashion.

Figure 11 compares the performance of C4.5 OPC and 31-bit ECOC as a function of the value of r. Because there are 500 training examples, all values of r less than 15 are certainly preventing any distant decision boundaries from influencing the behavior of C4.5. Nonetheless, we see that there is still a substantial performance advantage for the ECOC configuration, even when r is very small. The decision-boundary alignment hypothesis would have predicted that there would be no difference in the performance of these algorithms for small r values.

Based on these two experiments, it is quite clear that the decision-boundary alignment hypothesis is incorrect.

5.3 Tests of the Random-Algorithm-Behavior hypothesis

The random-algorithm-behavior hypothesis predicts that if the behavior of the learning algorithm is forced to be constant near the root of tree, then the errors between different f_l's will become strongly correlated. To test this hypothesis, we implemented a "constant splits" version of C4.5 in which the first four levels of the decision tree have fixed values such that the feature space is split into 16 square regions of equal size. After these fixed splits, the information gain heuristic of C4.5 is permitted to choose any further splits that are needed. We then computed the number of simultaneous errors committed by these 31 decision trees on each test data point and plotted them in Figures 12 and Figure 13.

Figure 13 shows the distribution of simultaneous errors for the standard ECOC configuration. It clearly shows that most of the errors are committed simultaneously by only a small fraction of the 31 decision trees. Because the 31-bit code can correct any 7 errors made by individual trees, all cases with 7 or fewer simultaneous errors will be correctly classified.

In contrast, Figure 12 shows the distribution of simultaneous errors when the top four levels of the tree were held constant. We can see many test cases having 8 or 16 simultaneous errors. These test cases will be incorrectly classified. Hence, it is not surprising that the error rate of ECOC worsens from 7.3% for the standard 31-bit ECOC to 9.0% for the constant-splits ECOC configuration. Interestingly, the error rate for individual decision trees on the constant-splits configuration actually improved (from 5.0% for standard 31-bit ECOC to 4.4% for the constant-splits ECOC). However, because these 4.4% errors were more highly correlated with one another, the aggregate performance worsened. This provides support for the random-algorithm-behavior hypothesis.

Another test of the random-algorithm-behavior hypothesis can be obtained by injecting randomness into the OPC configuration of C4.5, learning several sets of OPC trees, and having them vote to classify test examples. We implemented a version of C4.5 that computes the seven best candidate splits and then chooses randomly from those splits. We only permit these randomized splits for the first 8 levels of the decision tree. Beyond that point, the algorithm reverts to choosing the best possible split.

This approach of injecting randomness is similar to previous methods of constructing multiple decision trees and then having them vote to classify test examples. Kwok & Carter (1990) con-

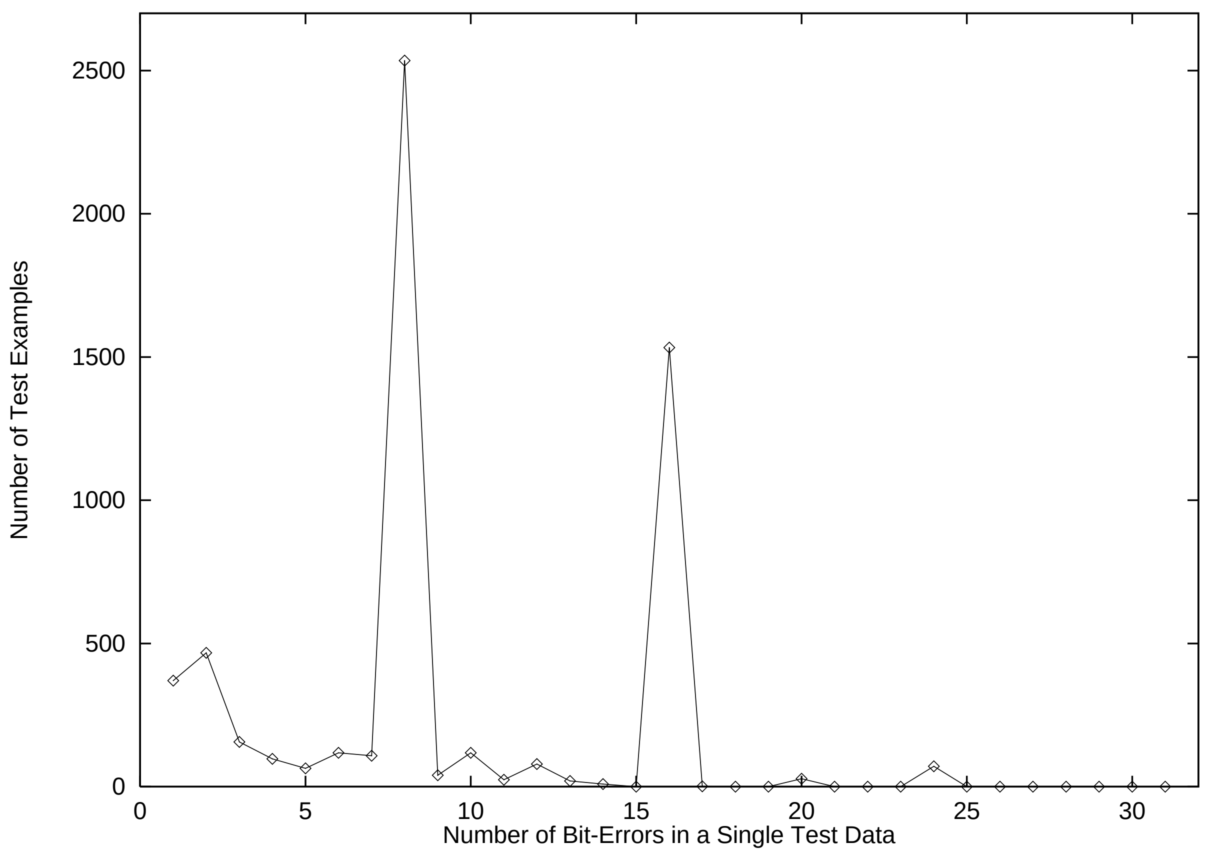


Figure 12: Bit-Error Distribution in Fixed-Split Learning

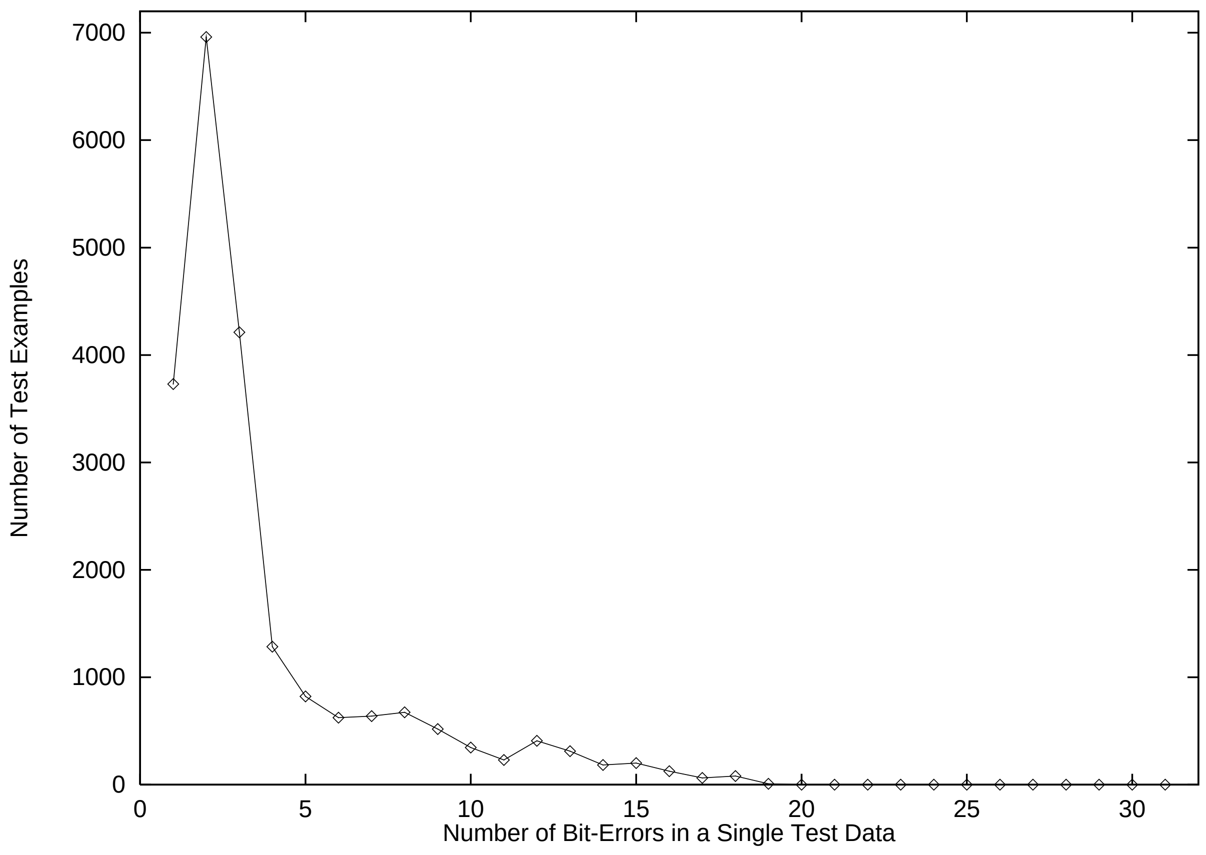


Figure 13: Bit-Error Distribution in ECOC

structed multiple trees by manually altering the choices of C4.5 near the root of the decision tree. Buntine (1990) evaluates an algorithm in which all equally promising splits at a node are pursued to construct an "option tree" (essentially a kind of decision DAG with multiple paths from the root to each leaf). When a new example is classified, all of these multiple paths are followed, and the results are averaged. Both of these previous studies showed improved performance.

Because ECOC is combining the votes of 31 trees, we chose to combine the votes of a nearly equal number of OPC trees—specifically, 30 trees (i.e., 5 full sets of 6 trees each). Five trees were learned for each class. To classify a test example, x', the C4.5 probability outputs of those five trees were summed to determine a score for that class. This was repeated for each of the six classes. The class with the highest score was chosen as the predicted class of the test example.

The error rate of this 5-fold randomized OPC was 7.8%, which is statistically indistinguishable from the error rate of 7.3% achieved by our standard 31-bit ECOC configuration. Based on this result, we conclude that the primary explanation for the excellent performance of error-correcting output coding is the random behavior of C4.5 near the root of the decision tree.

6 Discussion

So far, we have only considered artificial 2-dimensional problems for ease of illustration. In this section, we compare results on the Vowel speech recognition task obtained from the CMU connectionist benchmark collection (also available from the UC Irvine database).

The Vowel problem has 11 classes. Each training example is represented by 11 continuous features. Table 5 shows the performance of five different configurations of C4.5 on this task. As we

Table 5: Performance of five configurations of C4.5 on the Vowel problem.

Method	% Incorrect
C4.5 one-per-class	64.5
C4.5 multiclass	57.8
C4.5 6-fold randomized OPC	50.9
C4.5 64-bit ECOC	48.1
C4.5 384-bit ECOC	45.5

have seen in our synthetic problems, the OPC and multiclass configurations of C4.5 perform much worse than the ECOC configurations. However, notice that a 6-fold randomized OPC configuration (which has 66 trees) performs almost as well as the 64-bit ECOC configuration (which has 64 trees). This experiment therefore confirms that injecting randomness into C4.5 and voting multiple trees can obtain performance similar to that obtained by error-correcting output codes.

7 Concluding Remarks

The experiments in this paper show that the most plausible explanation of the success of error-correcting output coding is that the complex decision boundaries constructed by error-correcting output codes cause C4.5 to make random choices for splits near the roots of the decision trees. These random choices decorrelated the errors made by the various binary functions learned in the ECOC approach.

The paper also showed that the key factor determining the performance of the ECOC approach is the degree of independence in the errors committed by the learned binary functions. If we view inductive learning as a communications problem, the independent errors allow us to apply results from coding theory. If we view inductive learning as a form of voting along the decision boundaries in feature space, then independent errors enable us to estimate the correct decision boundaries much more accurately.

These observations hold for *any* method that combines multiple hypotheses through some kind of voting scheme. There has recently been an explosion of interest in combining multiple hypotheses, including, for example, the work of Schapire (1990), Freund (1992), and Drucker et al. (1992) on the "boosting" method and the work of Perrone & Cooper (1993) on taking linear combinations of neural networks trained from different starting seeds. Much attention has been paid to finding the optimal way of combining the votes of multiple hypotheses, but this ignores the most important question: Are the multiple hypotheses involved in the voting process making independent errors? If so, almost any voting scheme will give good results. If not, then no voting scheme can correct the correlated errors.

The key open problem in this research area is to develop methods for constructing multiple hypotheses so that their errors are uncorrelated. This paper describes two such methods: the error-correcting output coding method and the method of injecting randomness near the root of the decision tree in C4.5. The success of the boosting method may also derive directly from its algorithm for constructing multiple hypotheses. More methods need to be developed and evaluated, and all of these methods need to be compared to one another.

The error-correcting output coding method is only applicable to problems with at least five classes, and it shows promise of scaling to problems with a very large number of classes. These may turn out to be precisely the cases where this method outperforms all other methods. An important problem for future research is to study the effectiveness of various voting techniques on problems having hundreds or thousands of classes.

References

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

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

Buntine, W. L. (1990). *A Theory of Learning Classification Rules*. Ph.D. thesis, School of Computing, University of Technology, Sydney.

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

Dietterich, T. G., & Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes. Tech. rep., Department of Computer Science, Oregon State University, Corvallis, OR. Available via ftp://ftp.cs.orst.edu/users/t/tgd/papers/tr-ecoc-ps.gz.

Drucker, H., Schapire, R., & Simard, P. (1992). Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann.

Efron, B., & Gong, G. (1983). A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, *37*(1), 36–48.

Efron, B., & Tibshirani, R. (1991). Statistical data analysis in the computer age. *Science*, *253*, 390–395.

Freund, Y. (1992). An improved boosting algorithm and its implications on learning complexity. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pp. 391–398. ACM Press, New York, NY.

Holte, R. C., Acker, L. E., & Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 813–818. Morgan Kaufmann, San Francisco, CA.

Kwok, S. W., & Carter, C. (1990). Multiple decision trees. In Schachter, R. D., Levitt, T. S., Kannal, L. N., & Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence 4*, pp. 327–335. Elsevier Science, Amsterdam.

Murphy, P., & Aha, D. (1994). UCI repository of machine learning databases [machine-readable data repository]. Tech. rep., University of California, Irvine.

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

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. (1992). *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA.

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

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*, 379–423,623–656.

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*, Vol. 4, pp. 1133–1140. Morgan Kaufmann, San Francisco, CA.