AN ABSTRACT OF THE DISSERTATION OF

Eun Bae Kong for the degree of Doctor of Philosophy in Computer Science
presented on February 20, 1995.
Title: Understanding and Improving Error-Correcting Output Coding

Abstract approved: _____

Thomas G. Dietterich

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 and then combining the results of these L evaluations. Previous research has shown that ECOC can dramatically improve the classification accuracy of supervised learning algorithms that learn to classify data points into one of $k \gg 2$ classes. An investigation of why the ECOC technique works, particularly when employed with decision tree learning algorithms, is presented.

It is shown that the ECOC method is a compact form of "voting" among multiple hypotheses. The success of the voting depends on that the errors committed by each of the L learned binary functions are substantially uncorrelated.

By employing the statistical notions of bias and variance, the generalization errors of ECOC are decomposed into bias and variance errors. Like any voting method, ECOC reduces variance errors. However, unlike homogeneous voting, which simply combines multiple runs of the same learning algorithm, ECOC can also reduce bias errors. It is shown that the bias errors in the individual functions are

uncorrelated and that this results from non-local behavior of the learning algorithm in splitting the feature space.

ECOC is also extended to provide class probability information. The problem of computing these class probabilities can be formulated as an over-constrained system of linear equations. Least squares methods are applied to solve these equations. Accuracy of the posterior probabilities is demonstrated with overlapping classes and a simple reject option task.

Understanding and Improving Error-Correcting Output Coding

by

Eun Bae Kong

A Dissertation

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed February 20, 1995
Commencement June 1995

Doctor of Philosophy dissertation of Eun Bae Kong presented on February 20, 1995

APPROVED:

Major Professor, representing Computer Science

Head of Department of Computer Science

Dean of Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Eun Bae Kong, Author

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor and mentor Professor Tom Dietterich. He has been a constant source of ideas and encouragement to me. Without him, this endeavor was simply not possible.

I also thank Dr. Bella Bose who introduced me to the field of error-correcting codes. Through many consultations with him, I also broadened my understanding about error-correcting codes. His help is much appreciated.

Dr. Toshimi Minoura made me think about a lot of things. I learned a lot from him. I thank for his encouragement and hospitality.

Many thanks go to Professor Dietterich and Dr. Tadepalli for their insightful comments on this dissertation. Their comments made this dissertation much more comprehensible.

The support and sacrifice of my wife Young Hea and my boy Woo Tae are deeply appreciated. I also thank my mother and parents-in-law for their encouragements and supports.

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 SUPERVISED LEARNING 2

1.2 INDUCTIVE BIAS IN MACHINE LEARNING 4

1.3 GENERALIZATION ERROR 6

1.3.1 Error Rate 6

1.3.2 Apparent Error Rate 7

1.3.3 Resampling Techniques 8

1.3.3.1 Holdout method 8

1.3.3.2 Cross-validation 9

1.3.3.3 Bootstrapping 9

1.3.4 Statistical Bias and Variance 10

1.4 MEASURING BIAS AND VARIANCE 14

1.5 OBJECTIVES 17

1.6 OUTLINE OF THE THESIS 18

2 BACKGROUND 20

2.1 DECISION TREES 20

2.2 ERROR-CORRECTING OUTPUT CODING 24

2.3 RELATED RESEARCH 30

2.3.1 Option Trees 31

2.3.2 Committee Machines 32

2.3.3 Boosting 33

3 WHY ECOC WORKS 35

3.1 TWO PERSPECTIVES ON ERROR-CORRECTING OUTPUT CODING 36

3.1.1 Error-correcting output coding and communication theory ... 36

3.1.2 Error-correcting output coding and decision boundaries 40

3.2 REDUCING VARIANCE 44

3.3 ECOC REDUCES VARIANCE AND BIAS 52

3.4 BIAS DIFFERENCES ARE CAUSED BY NON-LOCAL BEHAVIOR 56

4 CLASS PROBABILITIES 58

4.1 ESTIMATION OF A POSTERIORI PROBABILITIES 58

4.2 ECOC AND CLASS PROBABILITIES 60

4.3 LEAST SQUARES PROBLEM WITH CONSTRAINTS 61

4.3.1 Least Squares Problem 62

4.3.2 Least Squares with Constraints 64

4.4 EXPERIMENTS ON OVERLAPPING CLASSES 65

4.5 REJECT OPTION 70

5 CONCLUSION 73

5.1 WHY ECOC WORKS 73

5.2 COMPUTING POSTERIOR PROBABILITIES FROM ECOC 75

5.3 FUTURE WORK 76

BIBLIOGRAPHY 77

LIST OF FIGURES

Figure		Page
1.1	A two-class problem with 200 training examples.	13
1.2	Bias errors of C4.5 on the problem from Figure 1.1	15
1.3	Bias errors of C4.5 on a diagonal decision boundary problem.	16
2.1	C4.5 decision trees: (a) example tree and (b) an equivalent **if-then-else** statement.	21
2.2	A neural network for digit recognition.	25
2.3	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 repository (Murphy & Aha, 1994). The POS task is to predict the part-of-speech of unknown words from their context.	29
3.1	A model of inductive learning as a communications problem.	37
3.2	Performance of C4.5 on the Letter Recognition task as a function of the length of the error-correcting code.	38
3.3	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.	39
3.4	A six-class problem in a 2-dimensional feature space. Various shapes show the training examples for the six classes.	41
3.5	The decision boundaries for the learning problem from Figure 3.4 . Each boundary is given a unique name.	42
3.6	Superimposed decision boundaries learned by functions f_1 through f_8 on the problem of Figure 3.4 .	43
3.7	Bias errors of C4.5 on the problem from Figure 3.4 estimated from 200 replications.	45

3.8 Learning curve for C4.5 on the problem from Figure 3.4 . The upper curve is the average performance of C4.5; the lower curve shows the persistent errors of C4.5. The difference between these two curves is the variance. Note that the horizontal scale is logarithmic. 46

3.9 A simple learning problem for which the correct function is a decision tree with axis-parallel splits. 47

3.10 Bias errors of 31-bit ECOC configuration of C4.5. 53

3.11 Bias errors for ECOC bit functions f_1, f_2, f_3, and f_4 measured from 200 replications of the problem in Figure 3.4 54

3.12 Reduction in bias produced by 31-bit ECOC as compared to multi-class 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. 56

4.1 600 training examples for six overlapping classes. 66

4.2 (a) True posterior probability (b) Estimated posterior probability (c) difference between (a) and (b) of class 5. 67

4.3 True versus estimated posterior probabilities of ECOC and C4.5. . . . 68

4.4 Diagonal and off-diagonal elements from confusion matrix. 69

4.5 Rejection curve for the problem of Figure 4.1 72

LIST OF TABLES

Table		Page
2.1	15-bit Error-correcting output code for a 10-class problem	27
2.2	Hamming distances between the string of predicted bits $s_{\mathbf{x}'}$ and the codewords from Table 2.1	28
3.1	Comparison of three configurations of C4.5 on the data from Figure 3.4	45
3.2	Results on five domains (best error rate in **boldface**)	51
3.3	Results of experiments on the problem From Figure 3.4	55

UNDERSTANDING AND IMPROVING ERROR-CORRECTING OUTPUT CODING

1. INTRODUCTION

The central problem with which this thesis is concerned is the classification problem. Classification is the task of assigning an object into one of several possible categories. People are constantly faced with making important classifications. For our very survival we must be able to classify effectively the situations around us. We must recognize and avoid dangerous situations.

Some rules for classification are told to us explicitly. For example, children are told by their parents to avoid hot water, not to swallow coins, not to touch oven, and so on. However, learning from observation is the primary source of our knowledge. Children are rarely told how to tell cats from dogs. Instead, lots of cats and dogs are presented to them, and children themselves learn the classification rules. We learn so many things about our world by observing examples. This amazing ability to learn from observation is indispensable to our survival and to every day life.

It would be useful to have a machine that could classify things around us such as speech patterns, hand-written letters, and diseases. The problem is that the machine must have the rules for these tasks. The conventional way of endowing the machine with the ability to classify is programming. Codifying knowledge is quite time-consuming. It can be very difficult, because we do not have the explicit

knowledge of the rules in many cases. For these reasons, this is often called the "knowledge acquisition problem."

Even if we could enter the knowledge into the computer, the resultant machine would be very inflexible, carrying out only the specific tasks for which it was pre-programmed. For speech recognition, a machine should be able to adjust to different speakers; and for letter recognition, a machine should be able to correctly classify letters of different writers. It is extremely difficult for a programmer to anticipate these changing conditions.

Another approach is to develop systems that learn from experience. These systems can adapt themselves to the changing conditions. In supervised learning, a learning system is provided with a set of training examples, each labeled as belonging to a particular class. The goal of a learning system is to produce a classification rule which would assign new examples to correct classes. By developing computer systems that learn, we can alleviate the knowledge acquisition problem. In addition to getting an accurate classifier, we can also investigate what aspects of the objects are important in producing a classification and enhance our understanding of the problem.

1.1. SUPERVISED LEARNING

In supervised learning, the learning algorithm is given a number of training examples. For each training example, its true state of nature is also provided. The goal of the learning algorithm is to find a rule which will predict the correct state of nature for unseen future examples.

A training example is a pair of a point $\mathbf{x}$ in n-dimensional Euclidean space $\Re^n$ and its corresponding state of nature. The n components of $\mathbf{x}$ are called *features* of $\mathbf{x}$, and will be denoted as $a_1(\mathbf{x}), \ldots, a_n(\mathbf{x})$. The rule can be thought of as a function mapping these feature values to a state of nature. Then supervised learning can be stated as follows:

Given : Training examples $\mathbf{t} = \{\langle \mathbf{x}_1, f(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{x}_m, f(\mathbf{x}_m)\rangle\}$ for unknown function f,

Find : A good approximation $\hat{f}$ to f.

The function f is called the *target concept*, and the output $\hat{f}$ of the learning system is called the *hypothesis*. The set of candidate hypotheses from which $\hat{f}$ is chosen is called the *hypothesis space*.

For classification problems, the range of the function f is a set of discrete class identifiers $C = \{c_1, \ldots, c_k\}$. For regression problem, the range will be continuous real space $\Re$. Since this thesis is mainly concerned with classification problems, it is assumed that the function f takes a class identifier as its value.

The sample size m is typically only a very small fraction of the whole population. We encounter only a small number of cats and dogs throughout our lifetime. From these limited number of training examples we want to find a good approximation $\hat{f}$. There can be many possible candidates for $\hat{f}$ to be considered. Learning is an extremely difficult task.

Suppose that a learning algorithm is required to learn an unknown function f which is defined on n Boolean features. There can be a total 2^n training examples. If the function f is known to be a Boolean function, f can be thought of as a truth table with 2^n entries. Among the 2^n entries, the system is provided with only m training examples, where m is considerably smaller than 2^n. Therefore, the system

has to guess the function values for the remaining $2^n - m$ entries. The function can take 0 or 1 on any of these entries. So, the total number of possible Boolean functions consistent with m training examples is $2^{2^n - m}$. The system will select one among those $2^{2^n - m}$ functions as a hypothesis. Without the knowledge that f is a Boolean function, the situation becomes more difficult, and the system may have to examine a huge number of various kinds of functions. Usually the number of hypotheses consistent with training examples is tremendously large. Therefore, it is necessary to incorporate the additional information to trim the hypothesis space.

1.2. INDUCTIVE BIAS IN MACHINE LEARNING

This additional information is called the *bias* of the learning algorithm (Mitchell, 1980). Bias refers to a restriction on or a preference within the space of hypotheses considered by a learning system. Bias is in effect stating a set of beliefs about how the world is expected to work. In restricting the hypothesis space, the bias makes learning feasible. Without bias, there would be no basis for making inductive generalizations. On the other hand, if a bias prevents a learning system from considering a good hypothesis, then the learned result will not be a satisfactory one. Supervised learning has been described as the study of biases (Shavlik & Dietterich, 1990).

Two forms of bias are generally employed in machine learning algorithm: *restricted hypothesis space bias* and *preference bias* (Dietterich, 1990). A restricted hypothesis space bias assumes that the correct target concept f is a member of some hypothesis set H. The hypothesis set H is usually defined in terms of syntactic representations. Some syntactic hypothesis spaces are simple conjunctions, k-DNF,

decision trees of depth k, perceptrons, and 3-layer neural networks with k hidden units. By restricting the syntactic representation of the hypothesis, the learning system can consider only specific forms of hypotheses. If the chosen restricted hypothesis space bias is appropriate to the problem on hand, the hypothesis space will contain good approximations to the target concept. If the target concept or an approximation to it is a not a member of the hypothesis space, the learning system cannot find the correct concept, and the bias is an incorrect one.

Another form of bias is preference bias. Under this bias, the learner imposes a preference ordering on the hypothesis space rather than restricting the syntactic form of the hypotheses. The preference ordering is usually expressed in terms of some measure of syntactic complexity of hypothesis representation. The learner, then, tries to find the best hypothesis according to this ordering, which is consistent with the training examples. For example, the decision tree algorithms such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993) prefer small trees over large ones in regards to the depth of the tree. Preference bias is a kind of Occam's razor (Blumer et al., 1987; Jefferys & Berger, 1992) which states that the simplest explanation of the observed phenomena is most likely to be the correct one.

Biases can also be described along the dimension of strength (Utgoff, 1986). A *strong* bias is one that makes the learning algorithm focus on a relatively small number of hypotheses. A *weak* bias is one that allows the learning algorithm to consider a relatively large number of hypotheses.

1.3. GENERALIZATION ERROR

One of the main areas of research in the field of machine learning has been the issue of generalization. In supervised learning, the machine is provided with a limited number of training examples of correct input-output pairs and required to generate a hypothesis that will produce answers to the *new* questions, rather than just recall the answers to *old* ones. We want the hypothesis to successfully *generalize* what it has learned to input patterns not in the training set. The generalization ability determines the accuracy of the hypothesis of a learning algorithm. One of the most important performance criteria for judging a learning algorithm is how well the hypothesis generated by it generalizes. As machine learning techniques are applied to increasingly many real-world problems, such as pronouncing English text (Sejnowski & Rosenberg, 1987), predicting protein secondary structure (Qian & Sejnowski, 1988), hand-written letter recognition (Le Cun et al., 1990), and predicting occurrence of tsetse flies (Ripley, 1993), generalization ability is the primary concern of the users of learning algorithms. Therefore, improving the generalization ability of learning algorithms is practically very important.

1.3.1. Error Rate

The most important performance measure in classification is the expected error rate of a classifier. The error rate provides a measure of the global performance of a classifier and provides a means of comparing competing classifiers.

The expected error rate is the probability that a randomly chosen input pattern is misclassified by a classifier. If input pattern $\mathbf{x}$ actually belongs to class c_i

and a classifier $\hat{f}$ incorrectly allocates it to some other class, $\mathbf{x}$ is misclassified and the error rate specific to the ith class is denoted as

$$e_i(\hat{f}) = p(\hat{f}(\mathbf{x}) \notin c_i | \mathbf{x} \in c_i)$$

and the overall expected error rate is

$$e(\hat{f}) = \sum_{i=1}^{k} p(\mathbf{x} \in c_i) e_i(\hat{f}).$$

The above true error rate is defined in terms of true probability distribution of input patterns in the population which is unknown. Since a classifier is designed from a given training sample, it is important to have an error rate estimate based on a sample.

1.3.2. Apparent Error Rate

There are two kinds of errors: training errors and testing errors. Training error is the error made by a learning algorithm during the training session. Similarly, the testing error is the error made by the the machine when it is tested with examples not seen before. Testing error is also called generalization error.

As an estimate of true error rate, an obvious and easily computed estimator is the apparent error rate of training errors:

$$error\ rate = \frac{number\ of\ training\ errors}{number\ of\ training\ cases}$$

That is, the apparent error rate is the error rate of the classifier on the training samples that were used to build the classifier. If the number of training cases approaches infinity, the apparent error rate, or resubstitution error rate as it is

often called, will converge to the true error rate. In real problems, the sample sizes are usually relatively modest. So, the apparent error rate is a poor estimator of future performance.

In general, the apparent error rate tends to give an overly optimistic figure since the same training data that was used to build the classifier is used again to measure the error rate. Classifier built based on this overly optimistic estimator can give arbitrarily good fit to the given training data. This is called *overfitting* and leads to poor performance (Schaffer, 1993).

1.3.3. Resampling Techniques

The apparent error rate is a biased estimator in the statistical sense as a consequence of the classifier being tested on the same data from which it has been formed. In this section, three methods of avoiding the bias in the statistical sense are briefly reviewed.

1.3.3.1. Holdout method

The first method is the holdout method. The available data are randomly split into disjoint training and test subsets. The classifier is built using only the training subset, and then the error rate is assessed on the test subset. This method is inefficient in the use of data. However, when the size of the available data is sufficiently large, this method is simple and the estimate is reliable.

1.3.3.2. Cross-validation

The second method is cross-validation. In k-fold cross-validation, the available data is randomly partitioned into k approximately equal-sized disjoint subsets. The classifier is built with data combined from $k-1$ of the subsets and the error rate is assessed on the remaining subset. This is repeated k times. In each iteration, different subset is used as the testing subset. The average error rate over k iterations is the cross-validated error rate.

1.3.3.3. Bootstrapping

Finally, in bootstrapping (Efron & Tibshirani, 1993), a training subset consists of m data sampled with replacement from an original sample of size m. Data not found in the training subset form the testing subset. The estimated error rate is the average of the error rates over a number of iterations.

These resampling techniques can be used for purposes other than estimating the error rate. Performance improvement using the resampling technique is dealt in Chapter 3.

1.3.4. Statistical Bias and Variance

Consider a regression problem based on the training set $\mathbf{t} = \{\langle \mathbf{x_1}, y_1 \rangle, \ldots, \langle \mathbf{x}_m, y_m \rangle\}$ of size m. One of the widely used estimates of error is the mean-squared error:

$$E[(y - f(\mathbf{x};\mathbf{t}))^2 | \mathbf{x}, \mathbf{t}],$$

where $f(\mathbf{x};\mathbf{t})$ denotes the predicted value at $\mathbf{x}$ based on the training set $\mathbf{t}$ and the expectation $E[\cdot]$ is taken with respect to the (unknown) probability distribution $P(y|\mathbf{x})$.

The mean-square error can be rewritten as

$$E[(y - f(\mathbf{x};\mathbf{t}))^2 | \mathbf{x}, \mathbf{t}] = E[(y - E[y|\mathbf{x}])^2 | \mathbf{x}, \mathbf{t}] + (f(\mathbf{x};\mathbf{t}) - E[y|\mathbf{x}])^2$$

where the first term does not depend on the data and is simply the variance of y at $\mathbf{x}$. The second term measures the squared distance between the true regression function, $E[y|\mathbf{x}]$, and the estimator $f(\mathbf{x};\mathbf{t})$. Therefore, the second term displays the effectiveness of f as a predictor of y. The expected value of the second term over all possible realizations of training data of fixed size of m is will give the insight in designing a predictor.

The mean-square error of $(f(\mathbf{x};\mathbf{t}) - E[y|\mathbf{x}])^2$ over all possible realizations of training data $\mathbf{t}$ is decomposed into bias and variance (Geman et al., 1992) in the statistical sense:

$$\begin{aligned}
E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E[y|\mathbf{x}])^2] &= E_{\mathbf{t}}[((f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})]) + (E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}]))^2] \\
&= E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})])^2] + E_{\mathbf{t}}[(E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}])^2] \\
&\quad + 2E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})])(E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}])] \\
&= E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})])^2] + (E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}])^2
\end{aligned}$$

$$+2E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})])(E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}])]$$

$$= \underbrace{(E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})] - E[y|\mathbf{x}])^2}_{\text{bias}} + \underbrace{E_{\mathbf{t}}[(f(\mathbf{x};\mathbf{t}) - E_{\mathbf{t}}[f(\mathbf{x};\mathbf{t})])^2]}_{\text{variance}}$$

Statistical bias is the square of the difference between the average (over all possible $\mathbf{t}$) of $f(\mathbf{x};\mathbf{t})$ and the regression $E[y|\mathbf{x}]$. It captures the idea of a systematic error in the learning algorithm for a given sample size. The variance is the expected value of the squared difference between a particular predicted value $f(\mathbf{x};\mathbf{t})$ and the average of $f(\mathbf{x};\mathbf{t})$. The expectation is taken with respect to $\mathbf{t}$. The variance captures the variation in the algorithm from one training set to another. This variation can result from variation in the training sample, from random noise in the training data, or from random behavior in the learning algorithm itself, such as the random initial weights often used in backpropagation. Both bias and variance contribute to the error. Therefore, in order to improve the performance, it is important to reduce both bias and variance if possible, or to make a compromise between them.

In the same vein, the error in the classification problem can be examined in terms of statistical bias and variance. As discussed in the Chapter 1.2, learning is possible through the introduction of the inductive bias. Hereafter, this inductive bias will be called machine learning (ML) bias to distinguish it from the statistical bias.

The success of learning depends on the appropriateness of the ML bias. The classical example is the exclusive or (**XOR**) problem. Perceptrons can represent only linearly separable patterns. Since the **XOR** is not linearly separable, perceptrons cannot represent it and so cannot learn it (Minsky & Papert, 1969). But with 3-layer neural networks with hidden units, the **XOR** can be successfully learned (Rumelhart et al., 1986).

In other words, if a bias is appropriate to the problem, then the hypothesis space contains a good approximation to the target concept, and the error rate will be low. On the other hand, if a bias is incorrect, then the hypothesis space does not contain a good approximation, let alone the target concept. However large the training sample size is, the error rate will be large in this case. That is, some errors are caused as a consequence of the chosen bias. Depending on the bias, some objects will be persistently misclassified (i.e., will be misclassified with probability greater than 0.5). As noted above, the statistical bias of a learning algorithm is the persistent or systematic error that the learning algorithm is expected to make when trained on training sets of a fixed size m. Hence, the statistical bias is closely related to ML bias.

In Dietterich & Kong (1995), the statistical bias and variance for classification algorithms are formally defined. Suppose that f is a function that maps from the input space $\mathbf{X}$ to a finite set of class labels $\{c_1, \ldots, c_k\}$. Given a set $\mathbf{t}$ of training examples, algorithm A outputs an hypothesis $A(\mathbf{t}) = \hat{f}_{\mathbf{t}}$.

It is convenient to define $\hat{p}_{\mathbf{t}}(\mathbf{x})$ to be the probability that $\hat{f}_{\mathbf{t}}$ misclassifies test point $\mathbf{x}$. This probability is 1 if $\hat{f}_{\mathbf{t}}$ misclassifies $\mathbf{x}$ and 0 otherwise.

$$\hat{p}_{\mathbf{t}}(\mathbf{x}) = \begin{cases} 1 \text{ if } \hat{f}_{\mathbf{t}}(\mathbf{x}) \neq f(\mathbf{x}) \\ 0 \text{ if } \hat{f}_{\mathbf{t}}(\mathbf{x}) = f(\mathbf{x}) \end{cases}$$

Now suppose that, as before, we draw a sequence of training sets $\mathbf{t}_1$, ..., $\mathbf{t}_l$, each of size m, and apply our learning algorithm A to construct hypotheses $\hat{f}_{\mathbf{t}_1}, \hat{f}_{\mathbf{t}_2}, \ldots, \hat{f}_{\mathbf{t}_l}$. We define the *averaged probability of error* to be the average of these $\hat{p}$'s, where the average is taken over all possible training sets $\mathbf{t}$:

$$\overline{\hat{p}}(A, m, \mathbf{x}) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \hat{p}_{\mathbf{t}_i}(\mathbf{x}).$$

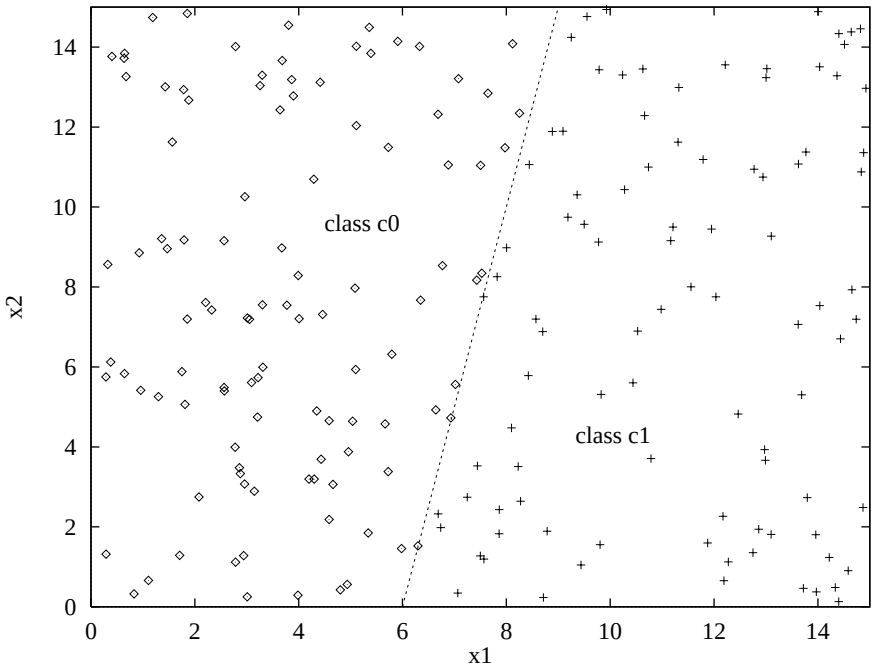


Figure 1.1. A two-class problem with 200 training examples.

Intuitively, $\overline{\hat{p}}(A, m, \mathbf{x})$ is the probability that a hypothesis produced by algorithm A from a training set of size m will misclassify test point $\mathbf{x}$. Another way of saying this is that the expected error rate of A for test point $\mathbf{x}$ is

$$Error(A, m, \mathbf{x}) = \overline{\hat{p}}(A, m, \mathbf{x}).$$

Now consider any point $\mathbf{x}$ such that $\overline{\hat{p}}(A, m, \mathbf{x}) > 0.5$. For any given hypothesis $\hat{f}_{\mathbf{t}}$, we expect on the average that $\mathbf{x}$ will be misclassified. Hence, $\mathbf{x}$ is a systematic error. We will therefore define the bias of any algorithm A trained on training sets of size m to be

$$Bias(A, m, \mathbf{x}) = \begin{cases} 0 & \text{if } \overline{\hat{p}}(A, m, \mathbf{x}) \leq 0.5 \\ 1 & \text{if } \overline{\hat{p}}(A, m, \mathbf{x}) > 0.5 \end{cases}$$

We will define the variance of A at point $\mathbf{x}$ to be the difference between the error rate and the bias:

$$Variance(A, m, \mathbf{x}) = \begin{cases} \overline{\hat{p}}(A, m, \mathbf{x}) & \text{if } \overline{\hat{p}}(A, m, \mathbf{x}) \leq 0.5 \\ \overline{\hat{p}}(A, m, \mathbf{x}) - 1 & \text{if } \overline{\hat{p}}(A, m, \mathbf{x}) > 0.5 \end{cases}$$

The variance is the increase in the error rate at $\mathbf{x}$ relative to the bias.

1.4. MEASURING BIAS AND VARIANCE

We can measure the bias and variance by directly simulating the definitions in the previous section. Figure 1.1 shows the target function for a simple learning problem. There are two features, x_1 and x_2, two classes c_0 and c_1, and a simple linear decision boundary such that points above the line are in class c_0 and points below the line are in class c_1. We define the distribution D over this two-dimensional input space to be the uniform distribution on the square $0 \leq x_1 \leq 15$ and $0 \leq x_2 \leq 15$.

To measure the bias of the C4.5 algorithm, we drew 200 training sets $\mathbf{t}_1$, ..., $\mathbf{t}_{200}$ each containing 200 examples labeled according to this target function (call it f). We also drew a complete test set of 22,801 examples (every possible data point on a grid of points separated by 0.1 along each axis). We then approximated $\overline{\hat{p}}(A, m, \mathbf{x})$ by computing the probability (averaged over all 200 training sets) that each test point $\mathbf{x}$ will be misclassified. This is equivalent to having each of the 200 hypotheses $\hat{f}_{\mathbf{t}_1}, \ldots, \hat{f}_{\mathbf{t}_{200}}$ vote on the classification of each test point.

Figure 1.2 plots all of the test data points whose estimated value of $\overline{\hat{p}}(A, m, \mathbf{x}) > 0.5$. These are the systematic errors. As one would expect, these errors result from C4.5's attempt to approximate the sloped decision boundary by vertical splits on feature x_1.

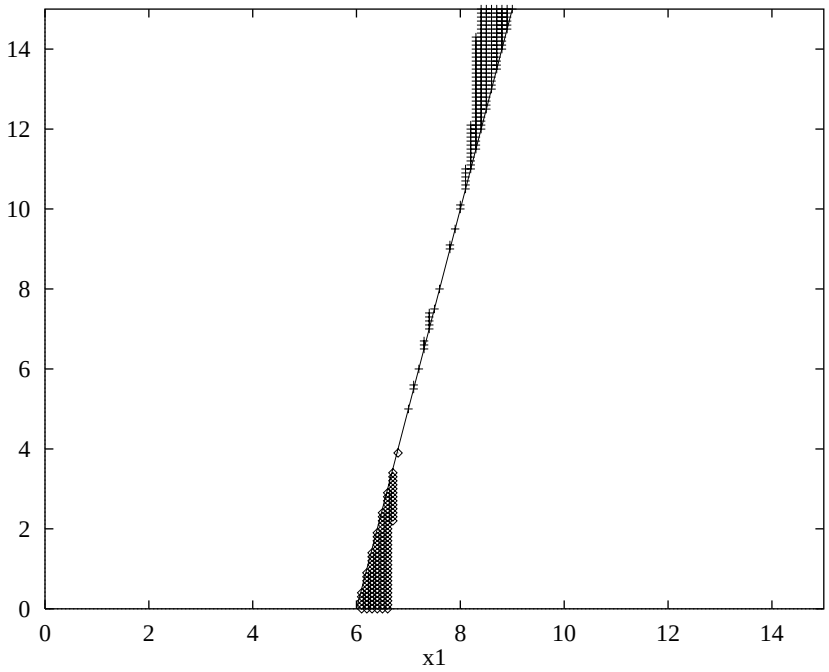


Figure 1.2. Bias errors of C4.5 on the problem from Figure 1.1.

We can compute the mean error rate over any T-element test set by the following:

$$\frac{1}{T}\sum_{j=1}^{T}\overline{\hat{p}}(A,m,\mathbf{x}_j),$$

where $\mathbf{x}_j$ is the j-th element in the test set. This is the average error of each of the 200 trees. We can also compute the bias as

$$\frac{1}{T}\sum_{j=1}^{T}Bias(A,m,\mathbf{x}_j)$$

and the variance by the difference between these two. The results show that the mean error rate is 536 errors (out of the 22,801 test examples), of which 297 are the result of bias and the remaining 239 are the result of variance. This is interesting,

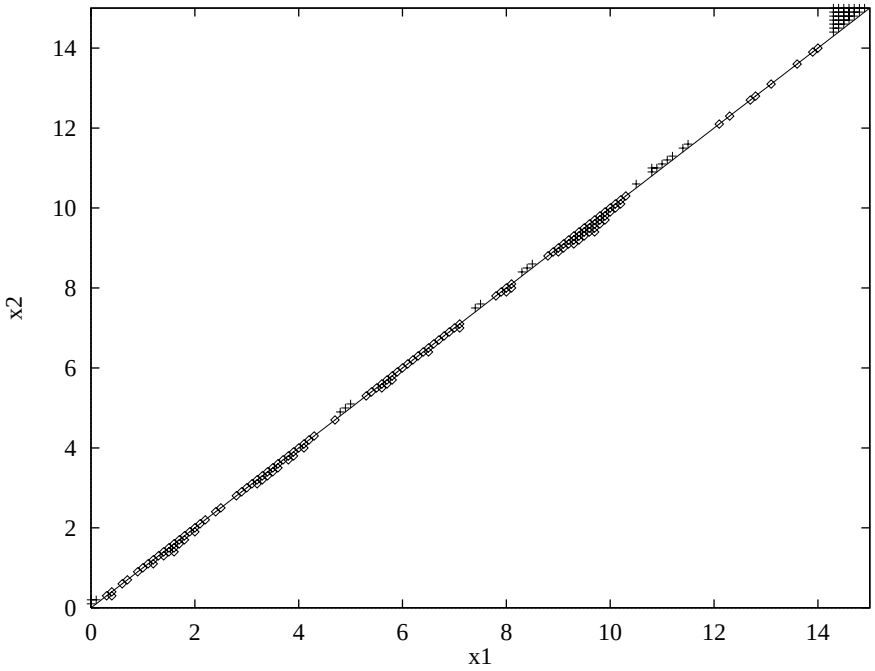


Figure 1.3. Bias errors of C4.5 on a diagonal decision boundary problem.

because it shows that variance accounts for a significant portion of the errors from C4.5 on this problem.

Figure 1.3 shows another example. Here, there is a single linear decision-boundary whose slope is 1. Surprisingly, C4.5 has very low bias for this problem. The bias is restricted to the very ends of the line. This is a result of the uniform distribution over the input space: splits on x_1 and x_2 are equally likely and, when averaged over all 200 training sets, they compensate for each other to give a good approximation to the diagonal decision boundary. The mean error for this problem is 821 errors, but the bias is only 177, so most of the errors result from the variance of 644 errors.

1.5. OBJECTIVES

Error-correcting output coding (ECOC) (Bakiri 1991; Dietterich & Bakiri 1991; Dietterich & Bakiri 1995) is a technique for applying machine learning algorithms to large-scale multiclass learning tasks. ECOC is a method for converting a k-class supervised learning problem into a large number L of binary supervised learning problems. Each of the k output classes, $c_1, \ldots, c_k$ is represented as a binary string (codeword) $\mathbf{S}$ of length L, such that the Hamming distance* between each pair of binary strings $\mathbf{S}_i$ and $\mathbf{S}_j$ is large. Each of the L learning problems consists of learning a classifier f_l that can predict bit $l = 1, \ldots, L$ of these binary strings. A new data point $\mathbf{x}$ is classified by computing $f_l(\mathbf{x}), l = 1, \ldots, L$ to produce a binary string $s_\mathbf{x}$ and then computing the class c_i whose binary string s_i is nearest to $s_\mathbf{x}$.

Previous experimental research (Dietterich & Bakiri, 1991; 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 other approaches to k-class learning problems. However, our understanding of the ECOC technique is still limited.

The goal of this thesis is

1. to understand why and when the ECOC technique works, particularly when employed with decision tree learning methods. Why the ECOC reduces the

*The Hamming distance between two binary L-tuples is the number of places where they differ.

generalization error is investigated. This thorough understanding will lead to a method of designing learning algorithms with even better classification accuracy.

2. to refine the technique so that it can provide class probability information. The posterior probabilities of class membership or their estimates provide a concise way of expressing the uncertainty of the class membership. They also provide useful probabilistic tools for subsequent decision-making processes.

1.6. OUTLINE OF THE THESIS

In Chapter 2, several existing methods are examined which improve the performance of a classifier and are related to the error-correcting output technique. A decision tree induction algorithm C4.5 on which most of the experiments is based will be briefly reviewed and the ECOC method will be formally defined. Previous results on ECOC will also be summarized.

In Chapter 3, two perspectives on error-correcting output coding are presented: communication theory and decision boundaries. The ECOC strategy is viewed as a compact form of "voting". It is noted that in order for the voting to be successful, the participants in the voting should be independent. Classification errors are analyzed in terms of bias and variance. It is shown that the ECOC method reduces both bias and variance by making each of the L functions substantially independent.

Chapter 4 presents a method of estimating the posterior class probabilities by using least-squares methods in the ECOC framework. The effectiveness of these probability estimates will be shown in reject option experiments.

In the concluding chapter, the contributions of the thesis will be summarized and a number of related areas for future research will be outlined.

2. BACKGROUND

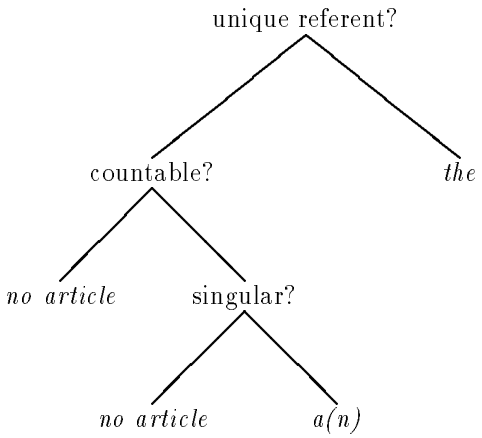
2.1. DECISION TREES

Inferring a good rule from given data has long been studied in statistical pattern recognition (Duda & Hart, 1973; Hand, 1981; Fukunaga, 1990). While a number of ideas from this area have proven useful, statistical pattern recognition often assumes that the given data were produced by special distributions. However, there is a need for inferring a rule regardless of the underlying distributions. With the advance of computing power, most research in practical machine learning dispenses with any distributional assumptions. These techniques which can be used without knowing the underlying distributions are called *nonparametric* procedures. Three representative nonparametric procedures are the nearest-neighbor rule (Fix & Hodges, 1951; Cover & Hart, 1967), decision tree methods (Breiman et al., 1984; Quinlan, 1986), and artificial neural network approaches (Rumelhart et al., 1986; Lippman, 1989).

The main goal of this thesis is to explain why error-correcting output coding works so well with decision tree learning algorithms. In this section, decision tree algorithms, particularly C4.5 (Quinlan, 1993), are reviewed.

A decision tree provides a hierarchical representation of the feature space. A classification is performed by proceeding down the appropriate branches of the tree until a leaf is reached. An example decision tree is shown in Figure 2.1, where a noun is classified into one of three classes according to whether the noun should

(a)



(b)

if a noun has a unique referent

then use *the*

else if countable

then if singular

then use *a(n)*

else use *no article*

else use *no article*

Figure 2.1. C4.5 decision trees: (a) example tree and (b) an equivalent **if-then-else** statement.

use *no article*, the definite article *the*, and the indefinite article *a(n)*. Also shown is an equivalent set of nested **if** statements. Each internal node of the tree tests the value of one of the features. 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, the predicted class is assigned.

Given the training data $\mathbf{t}$, C4.5 constructs a decision tree in two phases: (a) tree growth and (b) tree pruning. In the tree growing phase, a large tree is built by recursively partitioning the feature space. The tree growing procedure will usually result in a complex decision tree which performs badly on unseen cases. This phenomenon is called *overfitting*. In the tree pruning phase, the previously generated complex tree is made simpler by removing parts of the tree that do not contribute to classification accuracy on unseen cases.

In a decision tree, nodes represent the subsets of the feature space. Those subsets that are not further split in the decision tree are called terminal nodes. The root node represents the entire feature space. The decision tree initially consists of a single root node, containing all training examples. Starting at the root node, the best splitting feature variable and its cutoff value are chosen to form a test and the test is used to split the set of training data $\mathbf{t}$ at the root node into subsets of examples with the goal of separating them into sets of examples belonging to a single class. The examples that fail the test form the left child node and those that satisfy the test form the right child node. This procedure is then applied recursively to these two subsets of the training examples. The splitting can be continued until all of the examples at a node are from the same class.

The splitting feature variable and its cutoff value are chosen to maximize a quantity called the *gain ratio*. Assume node X consists of $K_{X,1}, K_{X,2}, \ldots, K_{X,k}$ ex-

amples from class $c_1, c_2, \ldots, c_k$, respectively. The entropy $H(X)$ (Cover & Thomas, 1991) of the class at node X is

$$H(X) = -\sum_{i=1}^{k} \frac{K_{X,i}}{|X|} \log(\frac{K_{X,i}}{|X|}).$$

The entropy is a measure of the uncertainty about the class. If test Y splits X into $X_1, \ldots, X_n$ children nodes, the conditional entropy $H(X;Y)$ after testing Y is

$$H(X;Y) = \sum_{j=1}^{n} \frac{|X_j|}{|X|} H(X_j)$$

which is a measure of uncertainty about the class after testing Y. The decrease in uncertainty about the class after testing Y is therefore

$$I(X;Y) = H(X) - H(X;Y).$$

$I(X;Y)$ is called the mutual information (Cover & Thomas, 1991), and it is called the *information gain* in C4.5. The information gain prefers to split a node into a large number of children nodes since that would make the children nodes more homogeneous. The resulting tree may have less predictive power. In order to correct this deficiency, the information gain is normalized by the amount of information produced by splitting X into n subsets. Then, the gain ratio is defined as

$$gain\ ratio = \frac{I(X;Y)}{-\sum_{j=1}^{n} \frac{|X_j|}{|X|} \log(\frac{|X_j|}{|X|})}.$$

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.

2.2. ERROR-CORRECTING OUTPUT CODING

Some learning algorithms, such as C4.5 and CART, can solve k-way classification problems directly. However, many learning algorithms are designed to solve binary (2-class) classification problems.

In a k-class classification problem, the standard way of denoting the class identifier of an entity $\mathbf{x}$ is to employ a k-dimensional vector $\mathbf{z}$ of zero-one indicator variables. The ith component of $\mathbf{z}$ is defined to be one or zero according as $\mathbf{x}$ belongs or does not belong to the ith class c_i: that is,

$$z_i = \begin{cases} 1 \text{ if } \mathbf{x} \in c_i \\ 0 \text{ if } \mathbf{x} \notin c_i \end{cases}$$

for $i = 1, \ldots, k$.

A natural way of using this encoding of class identifiers in machine learning is the discriminant functions approach (Nilsson, 1965), where k individual functions $f_1, \ldots, f_k$ are learned such that for all $\mathbf{x}$ in class c_i, $f_i(\mathbf{x}) > f_j(\mathbf{x})$ for $i, j = 1, \ldots, k, j \neq i$. A new object $\mathbf{x}$ is assigned the class c_l whose discriminant function f_l returns the largest value. This method will be called the *one-per-class* (OPC) approach, since one Boolean function is learned for each class.

The one-per-class approach is widely used in neural networks. For a k-class classification problem, there will be k output units. As an example, consider a neural network for digit recognition. Since there are 10 digits, there are 10 output

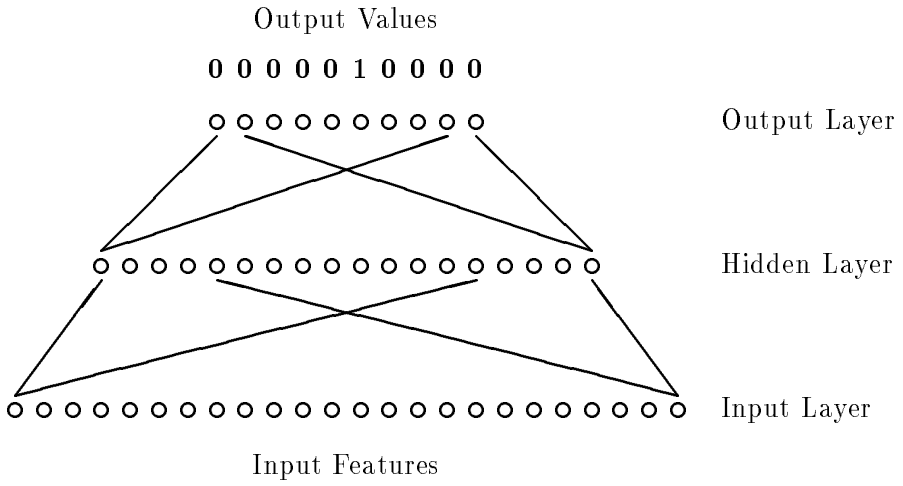


Figure 2.2. A neural network for digit recognition.

units in the output layer, one for each digit (See Figure 2.2). Each output unit is trained to be on only when the corresponding digit is presented to the network. For recognizing the digit 6, the $6th$ output unit is trained to return 1 and the rest of output units 0.

More generally, each class identifier can be represented by a bit string of length L. For classification, the only requirement for the encoding is that distinct classes are represented by distinct bit strings of length L; we will refer to these bit strings as "codewords." A code is a mapping of class identifiers into codewords. A code can be thought as a $k \times L$ matrix of 0's and 1's. Each row of the matrix corresponds to a codeword for a class. Each column corresponds to a Boolean function $f_l : \Re^d \mapsto \{0, 1\}$ so that $f_l(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ belongs to class c_i and (i, l)-th bit of the code is 1. The function f mapping objects to class identifiers can be made by functional projection from the set of Boolean functions $\{f_1, \ldots, f_L\}$:

$$f = (f_1, \ldots, f_L)$$

such that $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_L(\mathbf{x}))$. The learning task for f is reduced to the learning of the f_l's. Then L Boolean functions are learned, one for each bit position of the bit strings.

The one-per-class approach is one method of encoding. An alternative is employed in the NETtalk system (Sejnowski & Rosenberg, 1987), where meaningful properties in the domain such as "nasal", "voiced", "silent", and so on are considered when choosing the set $\{f_1, \ldots, f_L\}$. By having enough distinctive properties, the set $\{f_1, \ldots, f_L\}$ will give a unique codeword for each class. We will call this a *distributed output code* approach. During classification, a new object $\mathbf{x}$ is classified by evaluating L Boolean functions to form an L-bit string s.

The Boolean functions f_l are learned from finite examples. They are just approximations to the target functions. Therefore, the learned value of $f_l(\mathbf{x})$ may not be equal to the target value. In the one-per-class and distributed output code approaches, the Hamming distance between codewords is small and errors of even a single individual Boolean function $f_l(\mathbf{x})$ may result in misclassification.

Error-correcting codes (Peterson & Weldon, 1972) are widely employed in data communications systems. In a practical data communications system, there are occasional errors on the transmission channel due to perturbation noise. Codes have been developed that correct any pattern of t or fewer errors in a block of n symbols. By using such an error-correcting code, the errors made in learning the individual Boolean functions can be corrected.

In an application of error-correcting codes to machine learning, k distinct binary strings $s_1, \ldots, s_k$ of length L are chosen so that the Hamming distance between every pair of strings s_i and s_j is as large as possible. We will call each string s_i the *codeword* representing class c_i. Table 2.1 shows an example of a set of 10 such

Table 2.1. 15-bit Error-correcting output code for a 10-class problem

	Code Word (s_i)														
Class 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}
1	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
2	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
3	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
4	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
5	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
6	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
7	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
8	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
9	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
10	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

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 $f_j(\mathbf{x}) = 1$ if $f(\mathbf{x}) = c_i$ and the j-th bit of s_i is 1. Otherwise, $f_j(\mathbf{x}) = 0$. These correspond to the columns of Table 2.1.

During supervised learning, the j-th function, f_j, is learned by re-coding the examples to be $\{(\mathbf{x}_1, f_j(\mathbf{x}_1)), (\mathbf{x}_2, f_j(\mathbf{x}_2)), \ldots, (\mathbf{x}_m, f_j(\mathbf{x}_m))\}$ and applying a two-class learning algorithm. This is performed for each value of j to produce a set of L hypotheses, $\{\hat{f}_1, \ldots, \hat{f}_L\}$.

To classify a new example, $\mathbf{x}'$, we compute a vector of binary decisions $\hat{s} = \langle \hat{f}_1(\mathbf{x}'), \ldots, \hat{f}_L(\mathbf{x}') \rangle$ by applying each of the learned functions $\hat{f}_j$ to $\mathbf{x}'$. Then, we determine which codeword s_i is nearest to this vector (using the Hamming distance). The predicted value of $f(\mathbf{x}')$ is the class c_i corresponding to the nearest codeword s_i. For example, suppose that the predicted outputs for $\mathbf{x}'$ are

Table 2.2. Hamming distances between the string of predicted bits $s_{\mathbf{x}'}$ and the codewords from Table 2.1

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

$\hat{s} = \langle 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1 \rangle$. We now compute the Hamming distance between this string of predictions and each of the rows s_i of Table 2.1. The Hamming distances are shown in Table 2.2. The string s_5 for class c_5 has the smallest Hamming distance (3), so we predict $\hat{f}(\mathbf{x}') = c_5$.

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 $\lfloor (d-1)/2 \rfloor$ errors in the individual f_j's can be corrected, because the nearest codeword will be the correct codeword.

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 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.

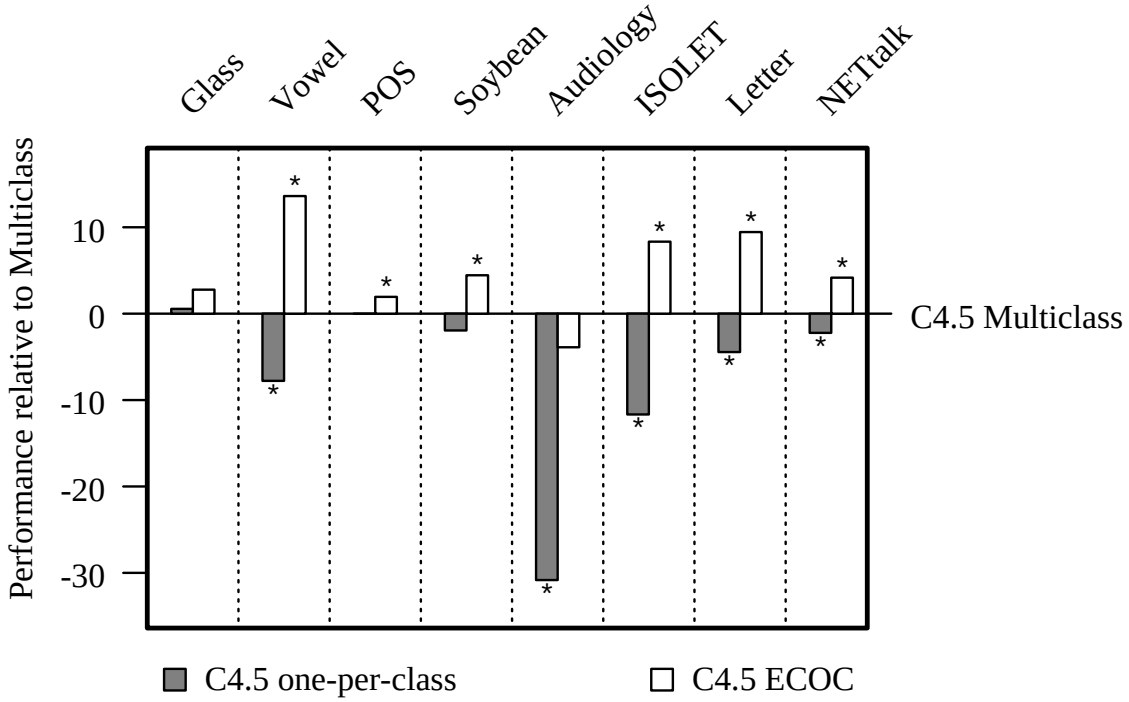


Figure 2.3. Performance of the one-per-class and ECOC methods relative to the direct multiclass method using C4.5. Asterisk indicates difference is significant at the 0.05 level or better. The glass, vowel, soybean, audiology (standardized encoding), ISOLET, letter recognition, and NETtalk data sets are all from the Irvine repository (Murphy & Aha, 1994). The POS task is to predict the part-of-speech of unknown words from their context.

Dietterich & Bakiri (1991,1995) have shown that the error-correcting output coding technique works very well with the decision-tree algorithm C4.5. Figure 2.3 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) 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 thesis.

2.3. RELATED RESEARCH

As the accuracy of learned hypothesis is the major concern of the learning algorithm, a lot of research effort has been exercised to improve accuracy. There has been work to improve the performance of tree induction algorithms with better pruning by employing more accurate error estimation (Crawford, 1989), and with better splitting rules (Mingers, 1989; Chou, 1991; Buntine & Niblett, 1992).

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; Wolpert, 1992; Xu et al., 1992; Breiman, 1993; LeBlanc & Tibshirani, 1993; Perrone, 1993; Perrone & Cooper, 1993; Meir, 1994; Breiman,

1994). ECOC can be viewed as a compact form of voting among multiple hypotheses. In this section, several methods are reviewed that improve the classification performance of learning algorithms by generating multiple hypotheses and then voting these hypotheses.

There can be two kinds of voting: *homogeneous voting* and *non-homogeneous voting*. In homogeneous voting, multiple runs of the *same* algorithm on the same learning problem are combined by voting. While in non-homogeneous voting, multiple runs of *different* algorithms on the same learning problem are combined by voting.

Error-correcting output coding involves neither homogeneous voting nor non-homogeneous voting. Like homogeneous voting, the same learning algorithm (e.g. C4.5) is applied many times. However, each of these applications of the algorithm solves a different learning problem.

2.3.1. Option Trees

Usual decision-tree induction algorithms return just a single tree out of numerous potential trees. Given the training examples, each tree will have a different posterior probability. If the best tree has a dominant peak posterior probability, returning the best tree may work. But, if we have many trees that have comparable posterior probabilities, by choosing a single best tree we ignore all the other many trees that could also be good for the problem.

Employing a Bayesian approach, Buntine (1990, 1992) considers averaging over many trees. Instead of returning a single tree, many different trees with high posterior probabilities are grown, and the average over them is sought. In option

trees (Buntine, 1990), several splittings, for example those whose evaluation is within some neighborhood of the best, are grown at a node. The resulting tree will be a single tree with many optional nodes. New data is classified by passing it down several branches at once and averaging the resulting probability vectors.

Kwok & Carter (1990) also build many different trees of comparable accuracy by selecting different splits close to the best at the top tree levels. Then they use each tree independently to classify data points and average the results.

These two methods are homogeneous voting. They use the same learning algorithm on the same learning problem and then vote.

2.3.2. Committee Machines

Neural networks (Lippmann, 1989; Hertz et al., 1991; Ripley, 1994) are widely used in classification. Generalization in neural networks has been extensively studied (Carnevali & Patarnello, 1987; Denker et al., 1987; Baum & Haussler, 1989; Schwartz et al., 1990; Haussler, 1992; Watkin et al., 1993). Ensembles of neural networks are used to improve the performance over a single network (Hansen & Salamon, 1990; Van den Broeck & Parrondo, 1993; Perrone & Cooper, 1993; Battiti & Colla, 1994). In a typical application, many different networks are generated with different hidden layer sizes, with different random initialization of weights, and/or by training with independent nonoverlapping training examples. The networks will differ in the values of the weights. These different weights produce different ways of generalizing the training data. Different ways of generalizing make individual networks commit errors on different subsets of the input space. The classifications of these networks

can be combined through various voting schemes to produce an ensemble classifier that is more likely to be correct than a single network.

2.3.3. Boosting

In Valiant's *probably approximately correct* (PAC) model of learning (Valiant, 1984; Dietterich, 1990; Natarajan, 1991), the learner is required to produce with high probability a hypothesis which is approximately correct. In the weak learnability model (Kearns & Valiant, 1989; Kearns, 1990), the requirement of approximate correctness is dropped; the learning algorithm is only required to find a hypothesis which performs slightly better than random guessing. It was proved that a weak learning algorithm can be converted into one that achieves arbitrarily high accuracy (Schapire, 1990; Freund, 1992 & 1995). This is called *boosting*.

In Schapire's *boosting by filtering* algorithm, the distribution of examples is modified in such a way that the weak learning algorithm (called WeakLearner) focuses its attention on the harder-to-learn parts of the distribution. Boosting begins by generating a first set of training examples D_1, applying WeakLearner to D_1, and producing a first hypothesis h_1. A second set of training examples D_2 is then generated by first flipping a fair coin: if the coin is heads, the examples oracle outputs examples until one which h_1 classifies correctly is output; otherwise, the oracle outputs examples until one which h_1 misclassifies is output. This process is repeated until enough examples are collected into D_2. In other words, D_2 consists of examples, half of which h_1 classifies correctly and half incorrectly. WeakLearner is applied to D_2 and the second hypothesis h_2 is output. Finally, a third training set D_3 is produced by filtering from the oracle those instances on which h_1 and h_2 agree.

WeakLearner is applied to D_3, producing hypothesis h_3. The hypothesis finally output is the majority vote of h_1, h_2, and h_3. The above procedure is recursively applied until the desired error rate is achieved. This idea of a boosting algorithm is implemented and demonstrated to be effective with neural networks (Drucker et al., 1993, 1994a, & 1994b). Freund (1992 & 1995) gives an iterative version of the boosting algorithm.

3. WHY ECOC WORKS

This chapter 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.

As defined in chapter 1, the statistical bias of a learning algorithm is the set of test examples that the learning algorithm will persistently misclassify. The statistical variance of a learning algorithm is the difference between the average error rate of the algorithm and the statistical bias. Using these definitions of bias and variance, we show that there are 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 chapter 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.

The remainder of this chapter is structured as follows. First, we present two equivalent perspectives on error-correcting output coding: an information-theoretic perspective and a decision boundary perspective. Both perspectives will demonstrate the importance of learning binary functions with uncorrelated errors. Second, we perform experiments to measure the bias and variance of ECOC. Then, we analyze the error of ECOC with respect to bias and variance. The chapter concludes with a discussion of the experiments and their implications for various methods of combining guesses from multiple learned hypotheses.

3.1. TWO PERSPECTIVES ON ERROR-CORRECTING OUTPUT CODING

3.1.1. 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 3.1). Imagine a communications problem in which the sender wishes to

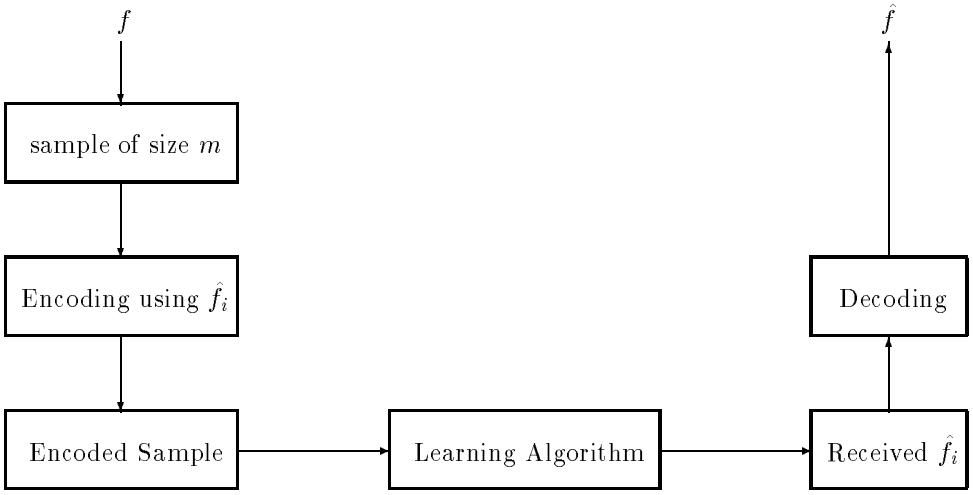


Figure 3.1. A model of inductive learning as a communications problem.

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 $\langle \mathbf{x}, f(\mathbf{x}) \rangle$. 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.

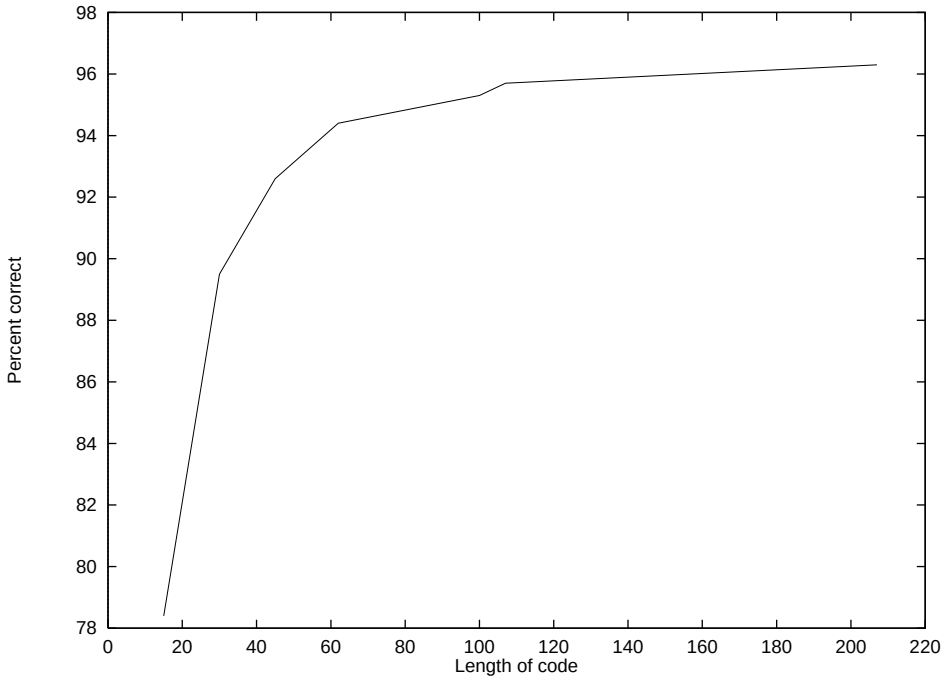


Figure 3.2. Performance of C4.5 on the Letter Recognition task as a function of the length of the error-correcting code.

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

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

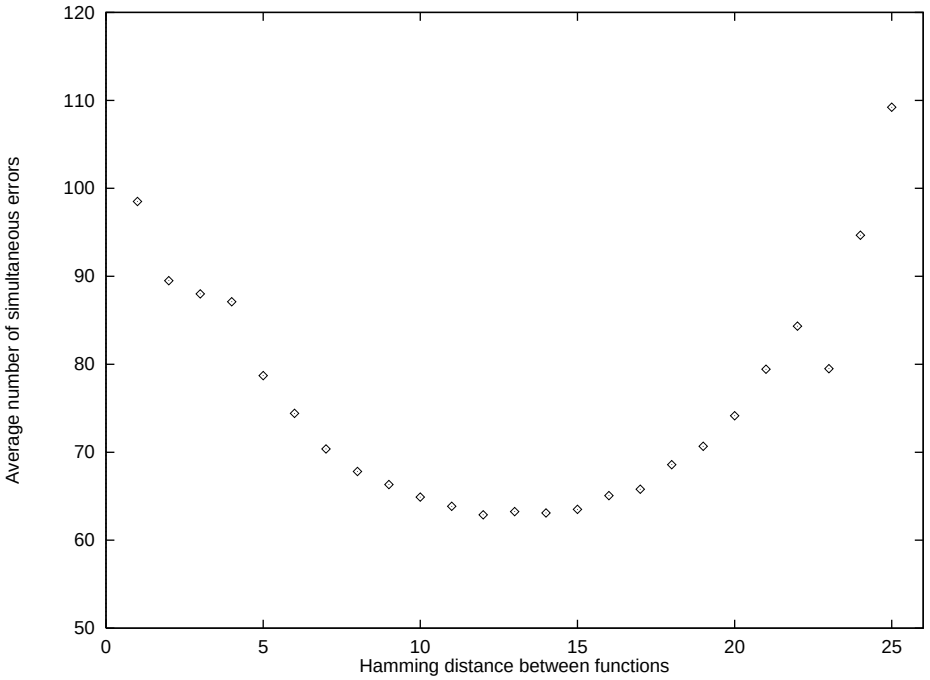


Figure 3.3. Number of simultaneous errors as a function of the Hamming distance between columns of the code. Each point shows the average number of simultaneous errors (on 1000 test examples) of all pairs of function $\hat{f}_i$ and $\hat{f}_j$ separated by the indicated Hamming distance.

function $\hat{f}$ ceases to improve beyond a certain point. Figure 3.2 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 in Figure 2.3.

More direct evidence for dependencies between the errors of $\hat{f}_i$ and $\hat{f}_j$ is shown in Figure 3.3. 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 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.

3.1.2. 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. For illustration we introduce a simple problem. Figure 3.4 shows a 6-class problem that will be extensively used throughout this chapter. Each of the 200 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 lines. For the error estimation, we use the holdout method. Since any reasonable classifier should be able to classify correctly an object whose category is obvious from its feature values, a test set containing 7,670 examples, which focused primarily on the decision boundaries, was also constructed. To apply the error-correcting output coding method to the problem in Figure 3.4, a 31-bit length error-correcting output code is constructed via the "exhaustive" method described in (Dietterich & Bakiri, 1995).

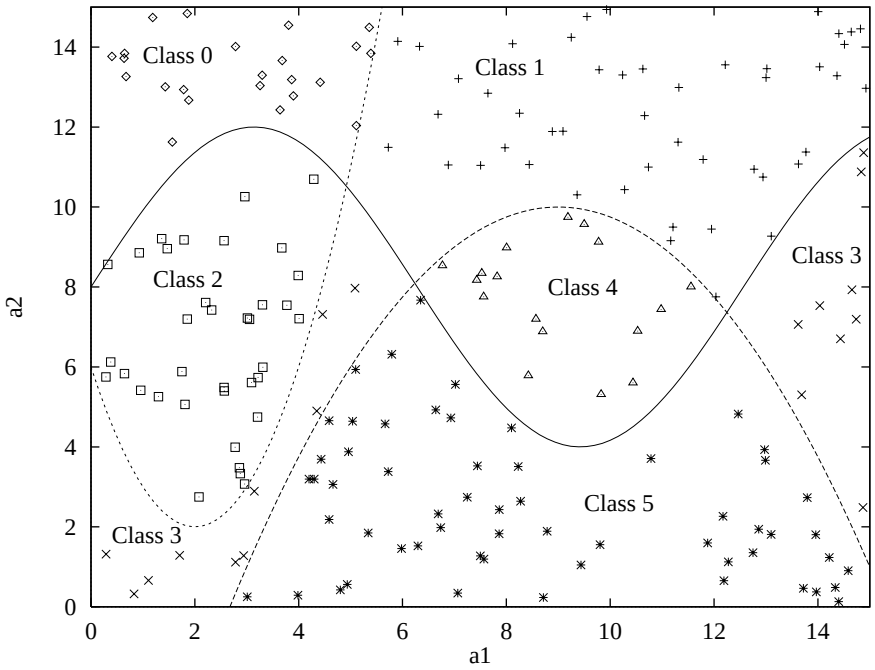


Figure 3.4. A six-class problem in a 2-dimensional feature space. Various shapes show the training examples for the six classes.

Consider Figure 3.5, which shows the decision boundaries from Figure 3.4. 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, there are three important observations. First, each individual binary function f_l must only learn *some* decision boundaries, and this set of decision boundaries varies from one function f_l to another. Second, each decision boundary is learned many times. Third, when a predicted binary string is "decoded" by the error-correcting code procedure (i.e., by mapping it

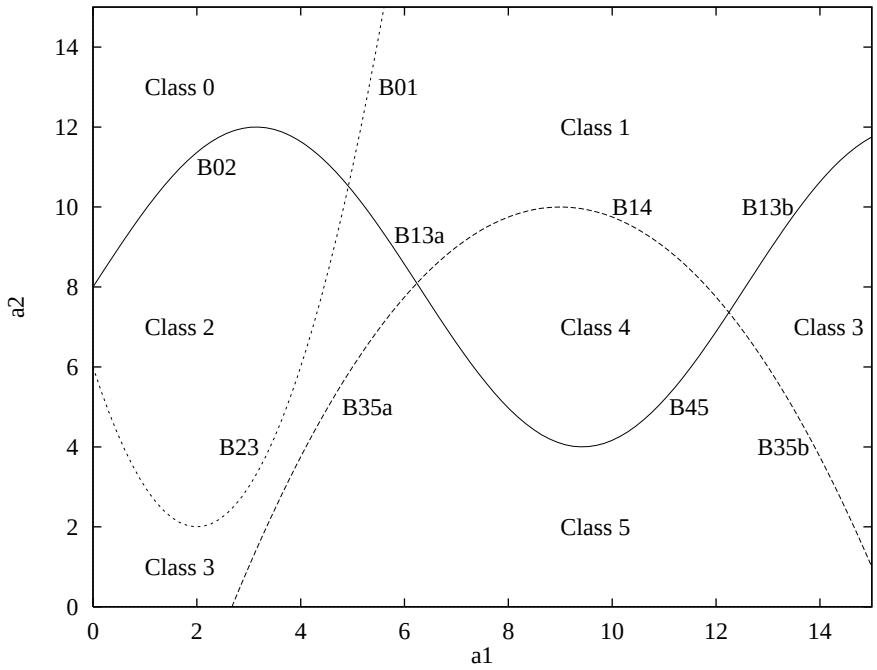


Figure 3.5. The decision boundaries for the learning problem from Figure 3.4. Each boundary is given a unique name.

to the nearest codeword), this is equivalent to a vote among those functions f_l that learned the relevant decision 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 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. Note that boundaries B02 and B45 must be learned by both f_2 and f_{13}.

In fact, 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

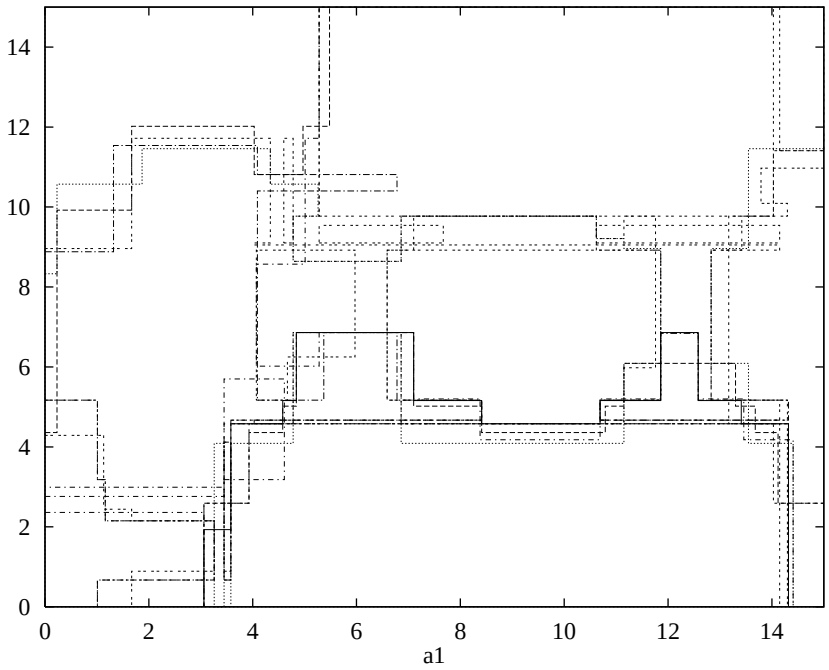


Figure 3.6. Superimposed decision boundaries learned by functions f_1 through f_8 on the problem of Figure 3.4.

16 away from every other codeword. That means that 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 equivalent to a form of voting. Suppose that 6 of the learned $\hat{f}_l$'s misclassify a test point $\mathbf{x}'$. There are still 10 learned $\hat{f}_l$'s that correctly classified it (i.e., that placed $\mathbf{x}'$ on the correct side of their decision boundaries), so $\mathbf{x}'$ will be correctly classified. The nearest codeword in Hamming 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 $\mathbf{x}'$.

Figure 3.6 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 learned boundaries are *not identical*. A point $\mathbf{x}'$ 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 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 3.1 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 ($p < 0.001$).

3.2. REDUCING VARIANCE

As noted in Breiman (1994), C4.5 has high variance. We have computed the bias and variance of C4.5 for the simple 6-class learning problem shown in Figure 3.4.

Table 3.1. Comparison of three configurations of C4.5 on the data from Figure 3.4

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

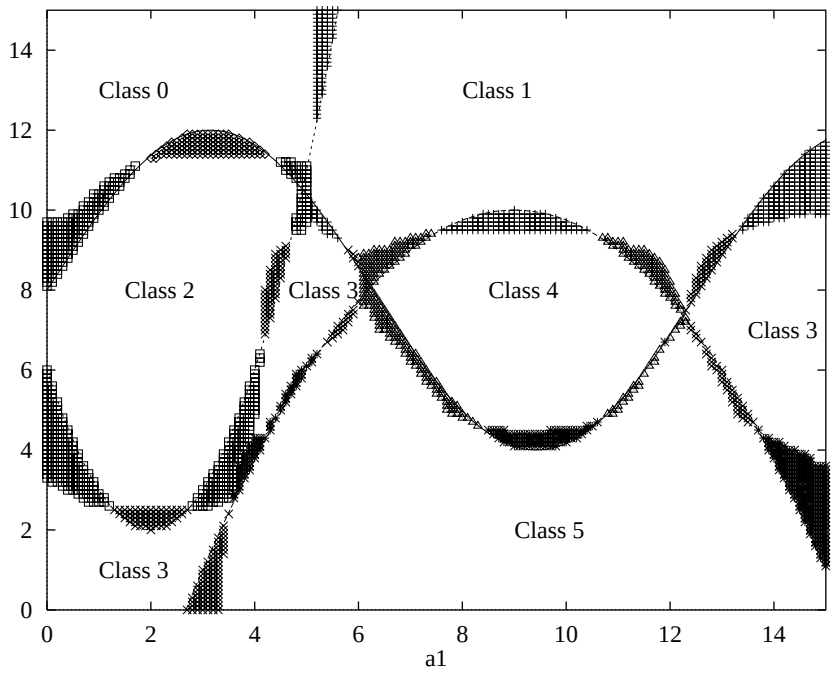


Figure 3.7. Bias errors of C4.5 on the problem from Figure 3.4 estimated from 200 replications.

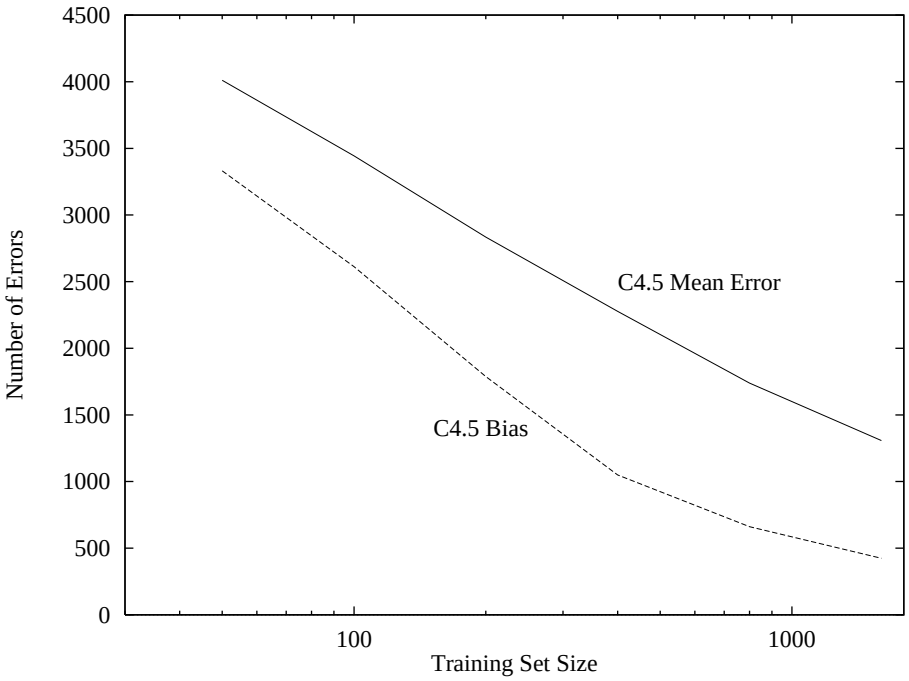


Figure 3.8. Learning curve for C4.5 on the problem from Figure 3.4. The upper curve is the average performance of C4.5; the lower curve shows the persistent errors of C4.5. The difference between these two curves is the variance. Note that the horizontal scale is logarithmic.

The statistical bias of this algorithm is 1788 and the variance is 1046, when training on training sets of size 200. Figure 3.7 shows the persistent errors of C4.5 on the problem from Figure 3.4.

We computed a learning curve for C4.5 on this problem which shows the mean error, statistical bias, and variance of C4.5 as a function of the size of the training sample (see Figure 3.8). Note that as the training sample gets larger, the bias of C4.5 decreases but the variance remains large.

Consider another problem where the decision boundaries can be exactly represented by an axis-parallel decision tree (see Figure 3.9). The mean error on this

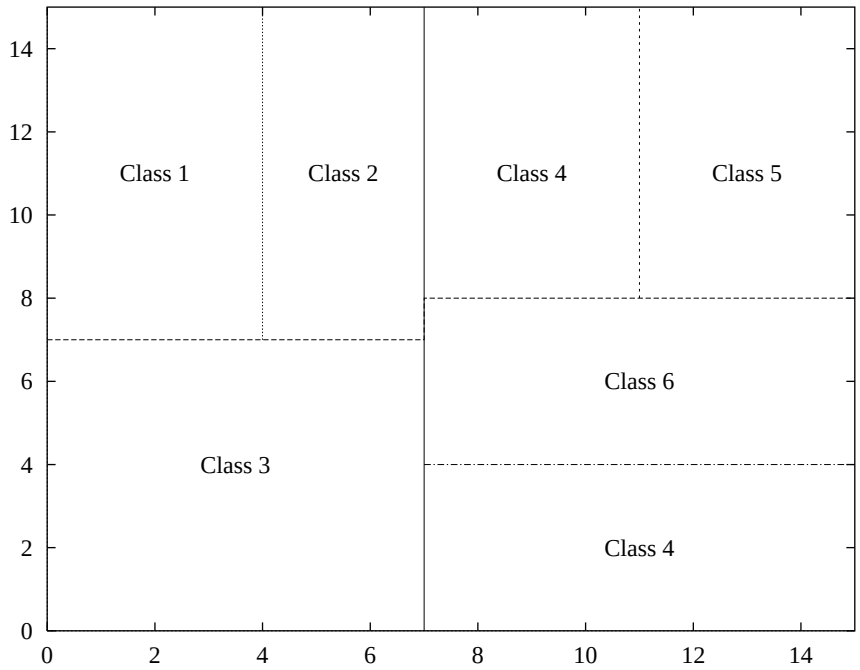


Figure 3.9. A simple learning problem for which the correct function is a decision tree with axis-parallel splits.

problem is only 17, with all 17 errors resulting from variance in the algorithm; the bias is zero. From these experiments, we can confirm that the variance is the significant component of errors in C4.5.

Because statistical bias and variance result from the basic design of a learning algorithm, any change to the algorithm can change the bias and variance. Any change that increases the representational power of an algorithm can reduce its statistical (and ML) bias. Any change that expands the set of available alternatives for an algorithm or makes them depend on a smaller fraction of the training data can increase the variance of the algorithm.

In particular, one important source of variance in C4.5 is the fact that the algorithm must choose a single split at each node in the tree. With most splitting

criteria (e.g., information gain, gain ratio, or GINI index), the choice of split can be altered by the addition or removal of a single training example. Furthermore, the subsequent splits at the descendent nodes in a tree are influenced by the split at the root, so that a change of one training example can produce a cascade of changes in subsequent splits and alter the entire tree. This is a plausible cause of the variance of C4.5.

This analysis suggests that the variance of C4.5 could be reduced by making the choice of splits more robust to slight changes in the training set or by making the splits "softer." The C4.5 system has a facility for softening splits such that test examples near the split threshold of a continuous feature are routed down both sides of the split. The class probability estimates resulting from those subtrees are then combined by weighted sum (where the weights depend on how close the example was to the split threshold). We ran this configuration of C4.5 on the program from Figure 3.4 and it resulted in a decrease in the mean error (from 2834 errors to 2740), a slight increase in the statistical bias (from 1788 to 1851) and a significant decrease in the variance (from 1046 to 889). Further improvements can probably be made here. We predict that the Markov Tree models of Jordan et al. (1991 & 1994), which are soft, stochastic versions of decision trees, would have much lower variance than C4.5. We also predict that other learning systems, such as RL (Clearwater & Provost, 1990), which learn collections of independent rules, will have lower variance also, because they reduce the cascading of decisions that results from the top-down construction of decision trees.

Many authors have suggested that an important source of variance in decision tree algorithms is the choice of splits and classifications for the leaves of the tree. These choices are based on a very small number of training examples, so one would expect them to have high variance. The conventional remedy for this is to prune

the tree to remove high-variance leaves. We tested this remedy using the pessimistic pruning procedure of C4.5 (with pruning confidence level 0.10) on the problem from Figure 3.4. We found that this had almost no effect on the variance (reducing it from 1046 errors to 1015), but that it increased the bias (from 1788 to 1927) and, hence, increased the mean error from 2834 to 2942. From this and many other test problems, we conclude that pruning does not necessarily work the way its advocates claim. In our experience, it rarely produces improvement in classification accuracy (Dietterich & Bakiri, 1995).

A very general technique for reducing variance is to construct a set of hypotheses and then have them vote on the classification of test cases. For example, if a relative ML bias does not have a strong preference for one hypothesis over another, the two hypotheses could both be generated and then voted. There are many different ways of producing and voting the hypotheses, and this is a very active topic of research, particularly in the neural network community (Hansen & Salamon, 1990; Wolpert, 1992; Xu et al., 1992; Breiman, 1993; LeBlanc & Tibshirani, 1993; Perrone, 1993; Perrone & Cooper, 1993; Meir, 1994; Breiman, 1994). There are strong Bayesian justifications for voting as well (Buntine, 1990).

We explored two methods for generating multiple hypotheses. The first is bootstrapping (Efron & Tibshirani, 1993; Breiman, 1994). Many equally-plausible decision trees can be constructed by the following procedure. Let S be the available training set of size m. We can draw a *bootstrap replicate training set* by drawing a set of examples S^1 of size m by sampling at random with replacement from S. We then apply C4.5 to this bootstrap replicate data set to obtain a bootstrap replicate hypothesis, $\hat{f}^1$. In our experiments, we constructed 200 bootstrap replicate decision trees. To classify a new example $\mathbf{x}$, we voted these trees by summing their class probability vectors component-wise and then choosing the class with the highest

probability. This relies on the ability of C4.5 to estimate the probability that $\mathbf{x}$ belongs to each of the k classes, $c_1, \ldots, c_k$. (This is in fact how we compute our estimates of $\overline{\hat{p}}$ throughout this chapter.)

Breiman (1994) calls this procedure *bagging* (for bootstrap aggregating). He shows that it produces very significant improvements in performance for the CART algorithm applied to several real-world data sets. When we followed this procedure with C4.5 on the problem from Figure 3.4, the mean error rate dropped from 2834 to 2648, the bias increased quite a bit from 1788 to 2067, but the variance was reduced by almost half from 1046 down to 581. Hence, in this problem at least, bootstrap aggregation trades a slight increase in bias for a major decrease in variance to yield a significant improvement in performance.

The second procedure that we explored for generating alternative hypotheses was randomization, which was first introduced in a simple form by Kwok & Carter (1990). We modified C4.5 so that at each node in the decision tree, it computes the 20 best splits according to its gain ratio criterion and then chooses randomly among them to select the attribute and value to split on. As with bootstrapping, we constructed 200 trees in this fashion and voted them by summing their class probability vectors component-wise. The result was that the mean error rate dropped from 2834 to 2627, which is almost exactly the same reduction obtained from bootstrapping (2648). However, unlike bootstrapping, with randomized trees, the bias remains virtually unchanged (1788 for a single tree versus 1772 for 200 random trees), while the variance decreases from 1046 to 855.

Randomization is paradoxical, because at first glance it seems to increase variance by deliberately introducing variation into the splits in the decision tree. However, it can also be seen as a way of smoothing or averaging out the effects of several equally-good splits by trying them all and then voting them.

Table 3.2. Results on five domains (best error rate in **boldface**)

Task	Test set size	C4.5	200-fold bootstrap C4.5	200-fold random C4.5
Vowel	462	0.5758	0.5152	**0.4870***
Soybean	376	0.1090	**0.0984**	0.1090
Part-of-Speech	3060	0.0827	**0.0765**	0.0788^{a}
NETtalk	7242	0.3000	0.2670***	**0.2500*****
Letter Recognition	4000	0.2010	0.0038***	**0.0000*****

Difference from C4.5 significant at $p < 0.05^{*}$, 0.001^{***}. ^{a}256-fold random.

We tested these two voting methods (bootstrap and randomization) on five interesting problems: the vowel, soybean, letter recognition, and NETtalk, and a part-of-speech task. The results are summarized in Table 3.2. Bootstrap voting improves performance on all five problems. The measured improvement in the accuracy of the hypotheses produced by the algorithms is significant for the bootstrap in the NETtalk and letter recognition task and for randomized C4.5 in the vowel, NETtalk, and letter recognition tasks. The letter recognition task is particularly astonishing. Bootstrap voting reduces the error from more than 20% to less than 1%, and randomized voting classifies the test set with perfect accuracy!

We have considered voting as a general technique for variance reduction. There is a very close connection between our definitions of bias and variance, and voting that helps explain how voting can improve performance. Consider a test point $\mathbf{x}$ and suppose that $\overline{\hat{p}}(\mathbf{x}) = 0.4$. This means that in any particular run of C4.5

(on a training set of size m), there is a probability of 0.4 that $\mathbf{x}$ will be misclassified. However, if C4.5 is run several times, only 40% of the trees will make errors and 60% will predict the correct class. The majority vote will correctly classify $\mathbf{x}$. Note however, that if C4.5 has no variance—so that it always outputs the same tree regardless of changes in the training set—then $\overline{\hat{p}}$ would always be either 1.0 or 0.0 and voting would have no effect.

Another way of viewing this is that if $\overline{\hat{p}}(\mathbf{x}) = 1$, then the errors being made on point $\mathbf{x}$ by different runs of C4.5 are very highly correlated and voting has no effect. However, if $\overline{\hat{p}}(\mathbf{x}) < 0.5$, then any tree that misclassifies $\mathbf{x}$ shares this error with fewer than half of the other trees, so the errors are fairly uncorrelated and voting can remove them.

Notice also that because a bias error is defined to be a point $\mathbf{x}$ such that $\overline{\hat{p}}(\mathbf{x}) > 0.5$, bias errors cannot be removed by voting multiple runs of a single algorithm on different training sets. Multiple runs can produce a more accurate estimate of $\overline{\hat{p}}$, but they cannot change its value (Perrone, 1993).

Variance can be reduced through averaging or voting over multiple hypotheses. However, as the experiments on the bagging and the voting among randomized trees demonstrated, voting could increase bias because details of the target function may be lost.

3.3. ECOC REDUCES VARIANCE AND BIAS

We measured the bias and variance of the ECOC approach on the problem from Figure 3.4. With ECOC, the mean error was reduced from 2834 to 2646. This is virtually the same reduction obtained by bootstrap aggregating (where bias

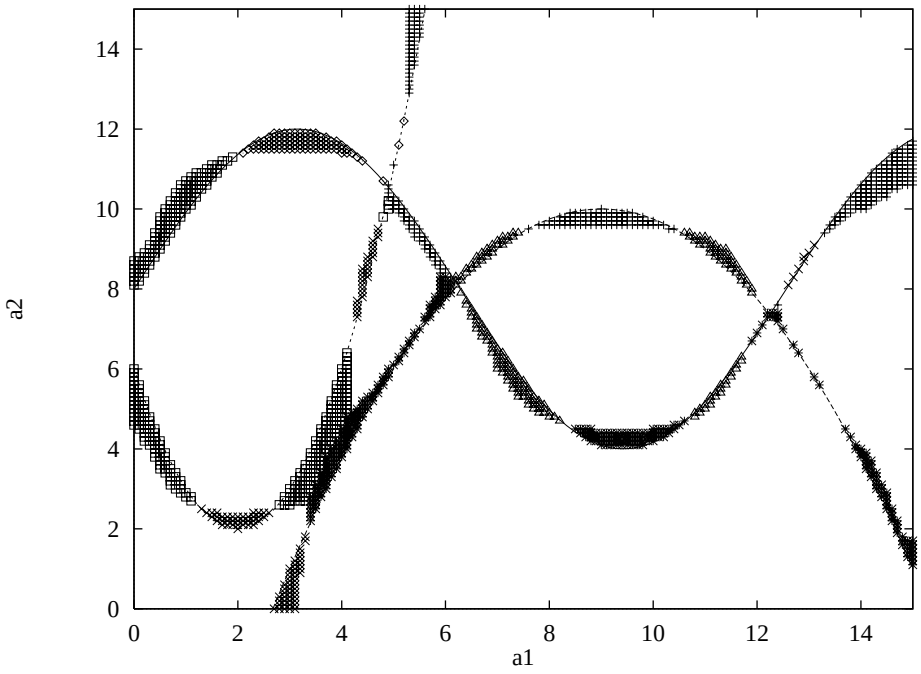


Figure 3.10. Bias errors of 31-bit ECOC configuration of C4.5.

increased and variance decreased). In this case, however, the improvement resulted from reductions in both the bias (from 1788 to 1669) and variance (from 1046 to 977). 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 is bias is more interesting. The bias errors of ECOC are plotted in Figure 3.10. The figure can be compared with Figure 3.7 to see where the bias has been reduced.

We also measured and plotted the bias errors of individual $\hat{f}_l$'s. Figure 3.11 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.

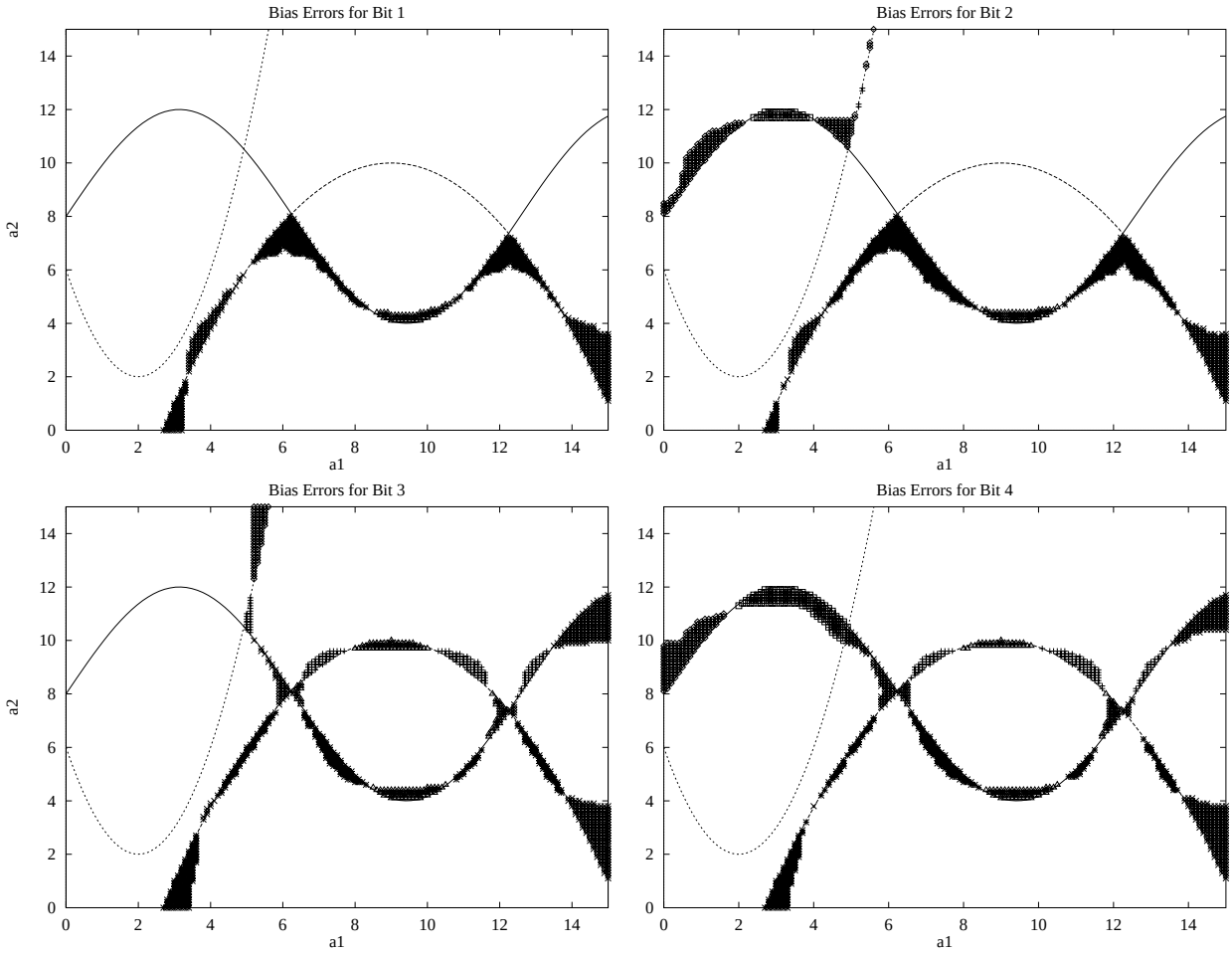


Figure 3.11. Bias errors for ECOC bit functions f_1, f_2, f_3, and f_4 measured from 200 replications of the problem in Figure 3.4.

Table 3.3. Results of experiments on the problem From Figure 3.4

Configuration	Mean Error	Bias	Variance
C4.5 multiclass	2834	1788	1046
C4.5 multiclass pruned	2942	1927	1015
C4.5 multiclass softened	2740	1851	889
C4.5 200-fold bootstrap	2648	2067	581
C4.5 200-fold random	2627	1772	855
C4.5 31-bit ECOC	2646	1669	977
C4.5 31-bit ECOC + 200-fold bootstrap	2407	1562	845

The variance reduction for C4.5 using ECOC is not very large in comparison to the variance reduction that can be obtained by performing a 200-replicate bootstrap and then voting the 200 resulting decision trees. After all, on this problem C4.5 is only voting 16 decision trees along each decision boundary, and this will not reduce variance as much as voting 200 trees would. This suggests a composite strategy in which we construct each $\hat{f}_l$ by performing a 200-fold bootstrap and voting those trees to produce $\hat{f}_l$. The results of these 31 separate "elections" can then be combined (by the decoding step of the error-correcting code) to classify each test example. Measurements confirmed this prediction: The mean error rate was reduced to 2407 (bias 1562 and variance 845). Hence, the ECOC strategy can be fruitfully combined with Breiman's bootstrap voting to achieve even better improvements in performance.

The results of all of our experiments are summarized in Table 3.3.

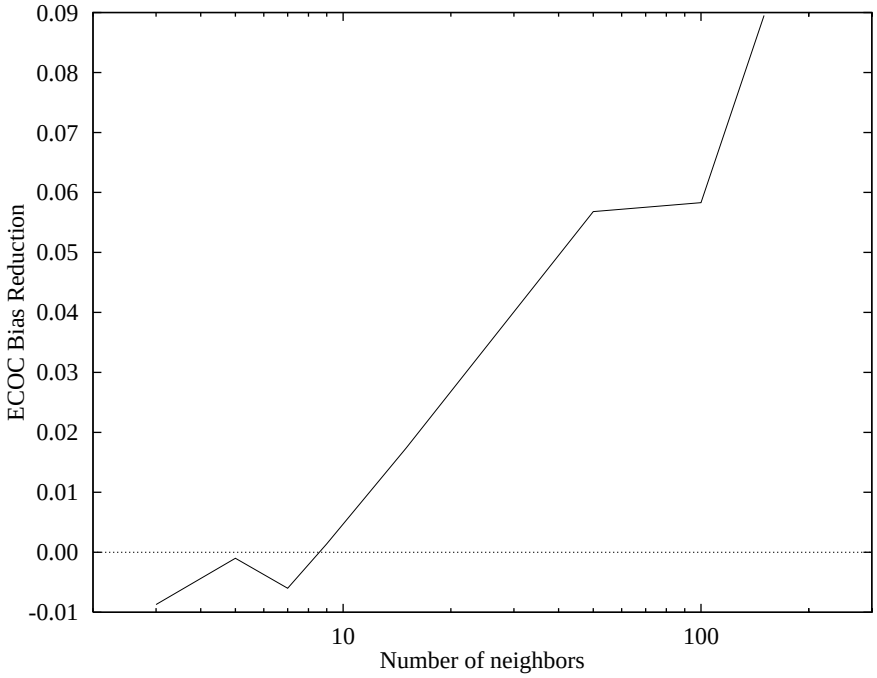


Figure 3.12. Reduction in bias produced by 31-bit ECOC as compared to multiclass C4.5 for local version of C4.5 that constructs an ad hoc tree to classify each test example using its k nearest neighbors. Note the log scale on the horizontal axis.

3.4. 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 3.11 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 3.11 in the region where $a_1(\mathbf{x}) = 12$ and $a_2(\mathbf{x}) = 7$. These bias errors result from the decision by C4.5 to approximate the B45 boundary by a horizontal line. It places this line near $a_2(\mathbf{x}) = 7$, 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 decision boundaries includ-

ing B14, B13a, and B13b. C4.5 chooses to make some vertical splits instead, so the bias shifts quite dramatically. In short, because C4.5 places splits based on global considerations, the presence of different distant decision boundaries in the different f_l's leads to different biases.

To test this hypothesis, we implemented a version of C4.5 that uses only local information to place decision boundaries. Specifically, our local-C4.5 works somewhat like a nearest neighbor algorithm: for each test example $\mathbf{x}'$, the k nearest neighbors are identified and used to construct a decision tree just for the purpose of classifying $\mathbf{x}'$. Because these local trees can 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 3.12 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 nine or fewer neighbors are given to C4.5, then 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.

From this experiment and discussion, we can 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.

4. CLASS PROBABILITIES

Many classification algorithms are able to supply more useful information than just a class identifier as an output. Various distance classifiers supply the distance between $\mathbf{x}$ and a prototype sample of each class, and Bayes classifiers supply posterior probabilities. This detailed information may be very useful. In many applications, the output of a classification algorithm is an input to subsequent decision-making processes. For example, in speech recognition the output of a phoneme classifier could be taken as input to a hidden Markov model for word recognition. In many cases, instead of producing just an output classification, it is better for the algorithm to produce a vector of posterior probabilities. In this chapter, we will develop methods for estimating such probabilities in ECOC framework.

4.1. ESTIMATION OF A POSTERIORI PROBABILITIES

The relative size of *a posteriori* probabilities $p(c_i|\mathbf{x})$ of classes can form the basis of classification rule. In the Bayesian approach, we will decide that the true class would be the class with maximum *a posteriori* probability. This rule will minimize the average probability of error. The probabilistic classification rule suggested by *a posteriori* probabilities $p(c_i|\mathbf{x})$ or their estimates $\hat{p}(c_i|\mathbf{x})$ also provides a concise way of conveying the uncertainty of the classification when the observed feature vector is $\mathbf{x}$.

A common approach to nonparametric estimation of probability is either Parzen-window technique or k-nearest neighbor technique (Duda & Hart, 1973; Silverman, 1986). Let m be the number of training examples. When a cell of volume V is placed around $\mathbf{x}$ and this cell contains d examples of which d_i are labeled c_i, then the obvious estimate for the joint probability $p(\mathbf{x}, c_i)$ is

$$\hat{p}(\mathbf{x}, c_i) = \frac{d_i/m}{V}.$$

Thus, a reasonable estimate for $p(c_i|\mathbf{x})$ is

$$\hat{p}(c_i|\mathbf{x}) = \frac{\hat{p}(\mathbf{x}, c_i)}{\sum_i \hat{p}(\mathbf{x}, c_i)} = \frac{d_i}{d}.$$

That is, the estimate of the *a posteriori* probability is just the fraction of examples within the cell that are labeled c_i. If there is an arbitrarily large number of training examples and the cell volume can become arbitrarily small, then the estimates will converge to the true probabilities.

In decision tree induction algorithms, a leaf contains a subset of training examples and denotes a class. Some of these leaves may contain training examples from more than one class. In order to estimate the reliability of a classification arising from a leaf, Quinlan (1987, 1993) estimates the class probabilities. Let n be the number of training examples in the leaf and e be the number of examples that do not belong to the class designated by the leaf. The class probability is estimated as

$$\frac{n-e}{n}.$$

This is called the *central estimate* (Quinlan, 1987). For leaves with very small n, this simple ratio may not give a reliable estimate of the probability. In this situation, a more *pessimistic estimate* can be used

$$\frac{n - e - 0.5}{n}.$$

4.2. ECOC AND CLASS PROBABILITIES

The output of ECOC for feature vector $\mathbf{x}$ is the bit strings that define the codeword $\hat{s}$. The Hamming distance between $\hat{s}$ and each codeword indicates the likelihood of a class. We used this likelihood as a classification rule. The class whose codeword is closest to the output bit string is more likely the true class. As the distance becomes longer and longer, the class is less likely.

One approach to estimate the class probabilities in the ECOC framework is to determine empirically a relationship between the Hamming distance to the class codewords and the class probabilities. We will not pursue this approach any further.

For neural networks, the output of the network can be interpreted as the *a posteriori* probabilities (Bridle, 1990;Wan, 1990; Richard & Lippmann, 1991; Denker & LeCun, 1991). In ECOC each individual function f_l is a binary classifier that groups an object into class 1 or class 0. Thus we can estimate the class probabilities using the central or pessimistic estimate described in the previous section.

Assuming that the outputs of the individual $f_l(\mathbf{x})$ are themselves class probabilities $\hat{f}_l(\mathbf{x}) = P(f_l(\mathbf{x}) = 1)$, we are given a vector of probabilities for each bit position of the error-correcting output code:

$$\mathbf{p^T} = \langle P(f_1(\mathbf{x}) = 1), P(f_2(\mathbf{x}) = 1), \ldots, P(f_L(\mathbf{x}) = 1) \rangle.$$

Since the class probabilities are mutually exclusive, $P(f_l(\mathbf{x}) = 1)$ is the summation of probabilities of classes which are classified as 1 by Boolean function f_l. For example, if the function f_l classifies the classes c_i and c_j as 1 and the rest as 0,

$$P(f_l(\mathbf{x}) = 1) = P(c_i|\mathbf{x}) + P(c_j|\mathbf{x}).$$

Then, we can readily formulate linear systems. We have L linear equations with k unknown probabilities in a k-class problem.

Let $s_1, \ldots, s_k$ be the codeword for class $c_1, \ldots, c_k$, respectively. Then, the code defined by $s_1, \ldots, s_k$ can be thought of as a $k \times L$ matrix of 0 or 1 as shown in Table 2.1. Let $\mathbf{C}$ denote a code matrix and $\mathbf{z}^T = \langle P(c_1|\mathbf{x}), \ldots, P(c_k|\mathbf{x}) \rangle$ be a vector of the k unknown probabilities that we seek to compute. Then, we want to solve the following systems of linear equations:

$$\mathbf{C}^T\mathbf{z} = \mathbf{p}$$

with the constraints $z_i \geq 0$ and $\sum z_i = 1$ since $\mathbf{z}$ is a probability vector.

Such systems of equations are usually inconsistent, since the number of equations does not match the number of unknowns. However, if we realize the fact that the equations are not exact but inexact equalities whose right-hand side vector is an estimate, we may solve the equations by the method of least squares (Lawson & Hanson, 1974).

4.3. LEAST SQUARES PROBLEM WITH CONSTRAINTS

The least squares method finds the closest point in a given subspace within a feasible region. In this section, we will briefly review the methods of least squares.

4.3.1. Least Squares Problem

We begin with the basic linear least squares problem.

LSP *Given an $m \times n$ matrix A of rank $r^* \leq min(m,n)$, and given an m-vector $\mathbf{b}$, find a real n-vector $\mathbf{x_0}$ minimizing the Euclidean length of $A\mathbf{x} - \mathbf{b}$.*

Orthogonal matrices† preserve the Euclidean length under multiplication. Thus for any m-vector $\mathbf{y}$ and any $m \times m$ orthogonal matrix Q,

$$\|Q\mathbf{y}\| = \|\mathbf{y}\|.$$

In minimizing the Euclidean length of $A\mathbf{x} - \mathbf{b}$, we have

$$\|Q(A\mathbf{x} - \mathbf{b})\| = \|QA\mathbf{x} - Q\mathbf{b}\| = \|A\mathbf{x} - \mathbf{b}\|$$

for any $m \times m$ orthogonal matrix Q and any n-vector $\mathbf{x}$.

The following theorem shows that by orthogonal decomposition of the matrix A, the least squares problem can be solved.

Theorem 1 *Suppose that A is an $m \times n$ matrix of rank r and that can be written as the product of matrices such as*

$$A = UVW^T$$

where

- *U is an $m \times m$ orthogonal matrix.*

---

*The maximal number of vectors that can be linearly independent is the *dimension* of a vector space. The *rank* of a matrix is the dimension of the row and column spaces.

†A square matrix Q is *orthogonal* if $Q^TQ = I$.

- *V is an $m \times n$ matrix of the form*

$$V = \begin{bmatrix} V_{11} & 0 \\ 0 & 0 \end{bmatrix}.$$

- *V_{11} is an $r \times r$ matrix of rank r.*

- *W is an $n \times n$ orthogonal matrix.*

Define the vectors $\mathbf{h}$ *and* $\mathbf{y}$

$$U^T\mathbf{b} = \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix}$$

$$W^T\mathbf{x} = \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$

where $\mathbf{h}_1$ *and* $\mathbf{y}_1$ *have r elements, and* $\mathbf{h}_2$ *and* $\mathbf{y}_2$ *have $m - r$ elements. Then,*

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\| &= \|UVW^T\mathbf{x} - \mathbf{b}\| \\ &= \|U^T(UVW^T\mathbf{x} - \mathbf{b})\| \\ &= \|U^TUVW^T\mathbf{x} - U^T\mathbf{b}\| \\ &= \|VW^T\mathbf{x} - \mathbf{h}\| \\ &= \|V\mathbf{y} - \mathbf{h}\|. \end{aligned}$$

Let $\tilde{\mathbf{y}}_1$ *be the unique solution to*

$$V_{11}\mathbf{y}_1 = \mathbf{h}_1.$$

Then all solutions minimizing $\|A\mathbf{x} - \mathbf{b}\|$ *are of the form*

$$\hat{\mathbf{x}} = W \begin{bmatrix} \tilde{\mathbf{y}}_1 \\ \mathbf{y}_2 \end{bmatrix}$$

where $\mathbf{y}_2$ *is arbitrary.*

Therefore, the minimum length solution is

$$\hat{\mathbf{x}} = W \begin{bmatrix} \hat{\mathbf{y}}_1 \\ 0 \end{bmatrix}.$$

With the orthogonal decomposition of matrix A, least squares problem can be solved. Singular value decomposition (Press et al., 1992; Lawson & Hanson, 1974) is the method of choice for orthogonal transformations.

4.3.2. Least Squares with Constraints

Our original problem has two constraints: equality and inequality constraints. The problem can be restated as follows:

LSIE *Given an* $m_1 \times n$ *matrix* D*, an* m_1*-vector* $\mathbf{d}$*, an* $m_2 \times n$ *matrix* E*, an* m_2*-vector* $\mathbf{e}$*, an* $m \times n$ *matrix* G*, and an* m*-vector* $\mathbf{g}$*, find one that minimizes*

$$\|D\mathbf{x} - \mathbf{d}\|$$

among all n*-vectors* $\mathbf{x}$ *that satisfy*

$$E\mathbf{x} = \mathbf{e}$$

and

$$G\mathbf{x} \geq \mathbf{g}.$$

Let W be an $n \times n$ matrix that transforms E to a lower triangular matrix when multiplied from the right:

$$\begin{bmatrix} E \\ D \\ G \end{bmatrix} W = \begin{bmatrix} \tilde{E}_1 & 0 \\ \tilde{D}_1 & \tilde{D}_2 \\ \tilde{G}_1 & \tilde{G}_2 \end{bmatrix}.$$

Then $\hat{y_1}$ is the solution of the lower triangular system $\tilde{E}_1 y_1 = \mathbf{e}$. We get the solution $\hat{y_2}$ of the following problem using the LDP algorithm (Lawson & Hanson, 1974):

$$\text{Minimize } \|\tilde{D}_2 y_2 - (\mathbf{d} - \tilde{D}_1 \hat{y_1})\| \text{ subject to } \tilde{G}_2 y_2 \geq \mathbf{g} - \tilde{G}_1 y_1.$$

After solving $\hat{y_2}$ the solution $\hat{\mathbf{x}}$ can be computed

$$\hat{\mathbf{x}} = W \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

4.4. EXPERIMENTS ON OVERLAPPING CLASSES

We can test the accuracy of posterior probability estimates by generating overlapping class-conditional probability density functions $p(\mathbf{x}|c_i)$'s. To generate a 6-class problem, we constructed six different Gaussian distributions in 2-dimension vector space with different means and covariances. Training examples are shown in Figure 4.1.

Figure 4.2 shows the actual and estimated posterior probability for class 5 which shares decision boundaries with three other classes. The discrepancies between the actual and estimated probabilities occur mainly along the decision boundaries. Except for the decision boundaries, the figure shows that the estimated probability approximates the actual probability pretty well.

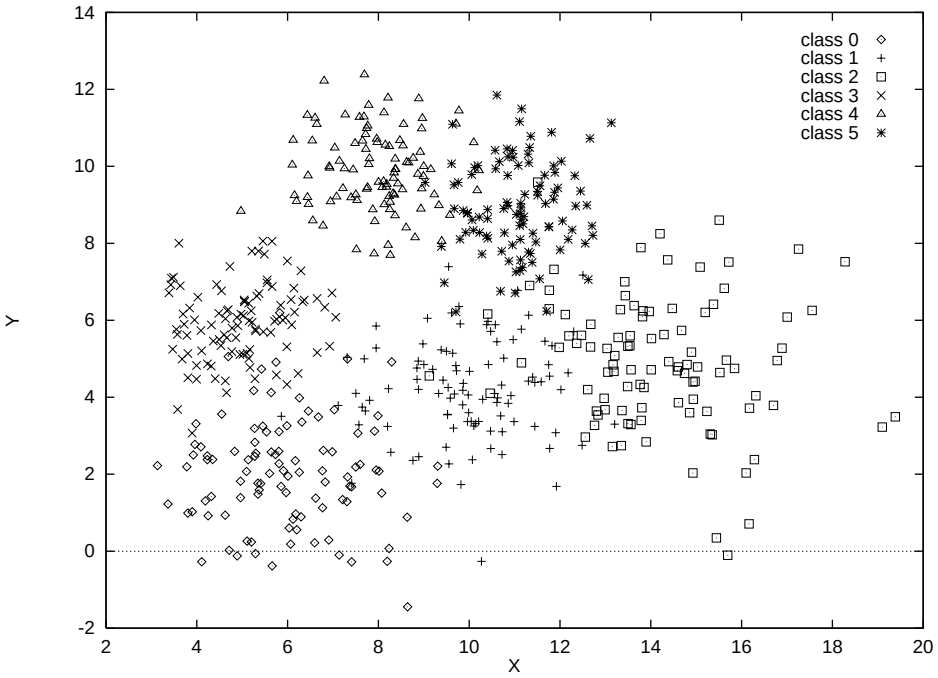


Figure 4.1. 600 training examples for six overlapping classes.

We also compared the estimated posterior probability values from ECOC and those from multiclass C4.5 (See Figure 4.3). The upper diagram compares the true probabilities with the estimated probabilities within the ECOC framework, while the lower diagram compares the true probabilities with the estimated probabilities within the multiclass C4.5 framework. As the figure shows, the quality of estimated probabilities is much better in ECOC than in C4.5. It appears that the ECOC method can also correct errors in the probability estimates of the underlying binary learning algorithm.

The confusion matrix of C4.5 shows how the misclassifications are distributed. When the true class is known, the estimated probabilities show how the misclassifications would distribute. Another way of testing how well the estimated

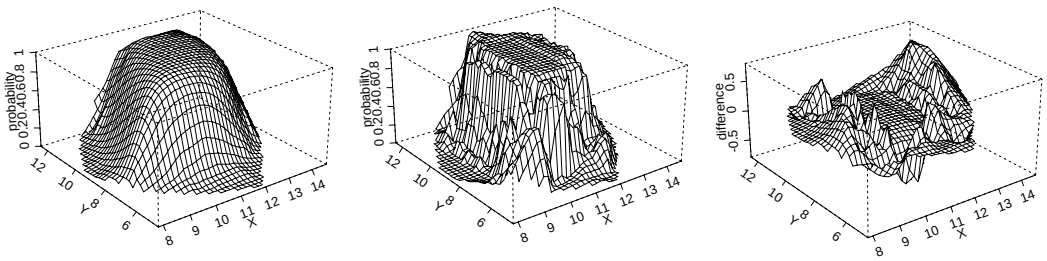


Figure 4.2. (a) True posterior probability (b) Estimated posterior probability (c) difference between (a) and (b) of class 5.

probability approximates the actual one is to see how well the estimated probability could predict the confusion matrix. Diagonal elements in a confusion matrix indicate the number of correct classifications, while off-diagonal elements indicate the number of misclassifications. Figure ?? compares the number of elements in the corresponding positions of the actual confusion matrix from ECOC and the confusion matrix predicted with estimated probabilities.. The upper diagram compares the number of the diagonal elements between two confusion matrices. It shows that the diagonal elements are consistently underpredicted. Pessimistic estimate of the C4.5 may be the cause for this underprediction. The lower diagram compares the number of off-diagonal elements. As a whole, the confusion matrix is predicted reasonably well.

The precision of estimation of probability in ECOC will depend on the size of training samples and the accuracy of the underlying estimation of each bit position.

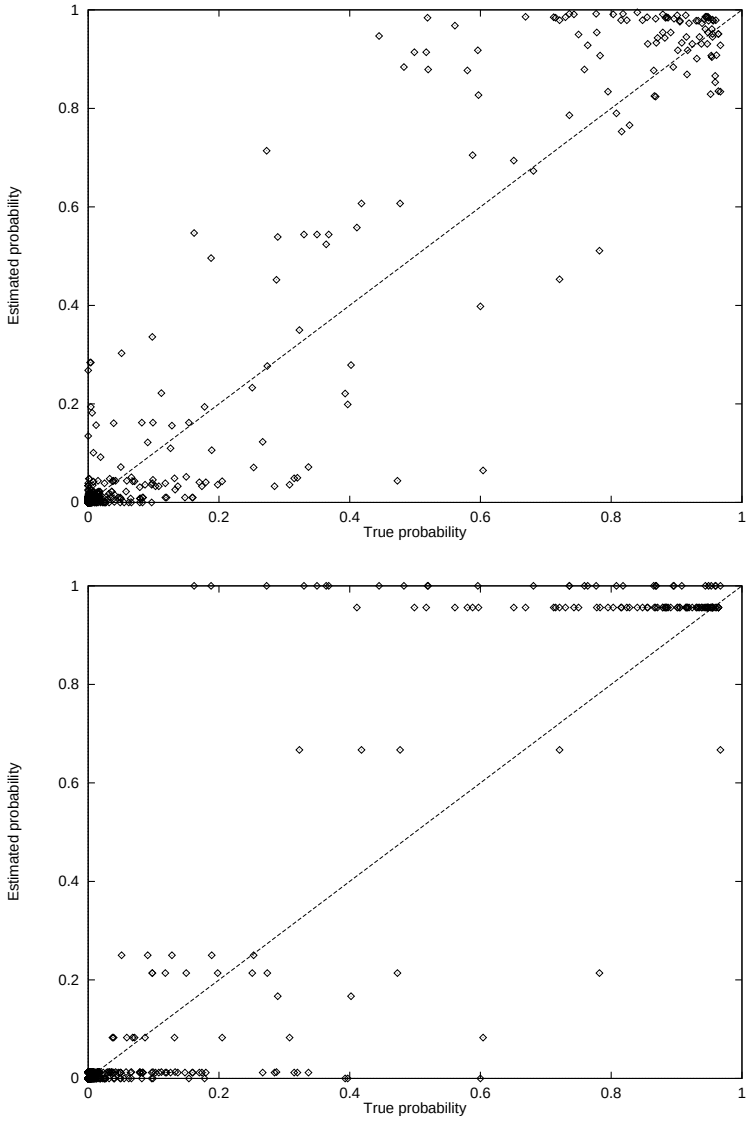


Figure 4.3. True versus estimated posterior probabilities of ECOC and C4.5.

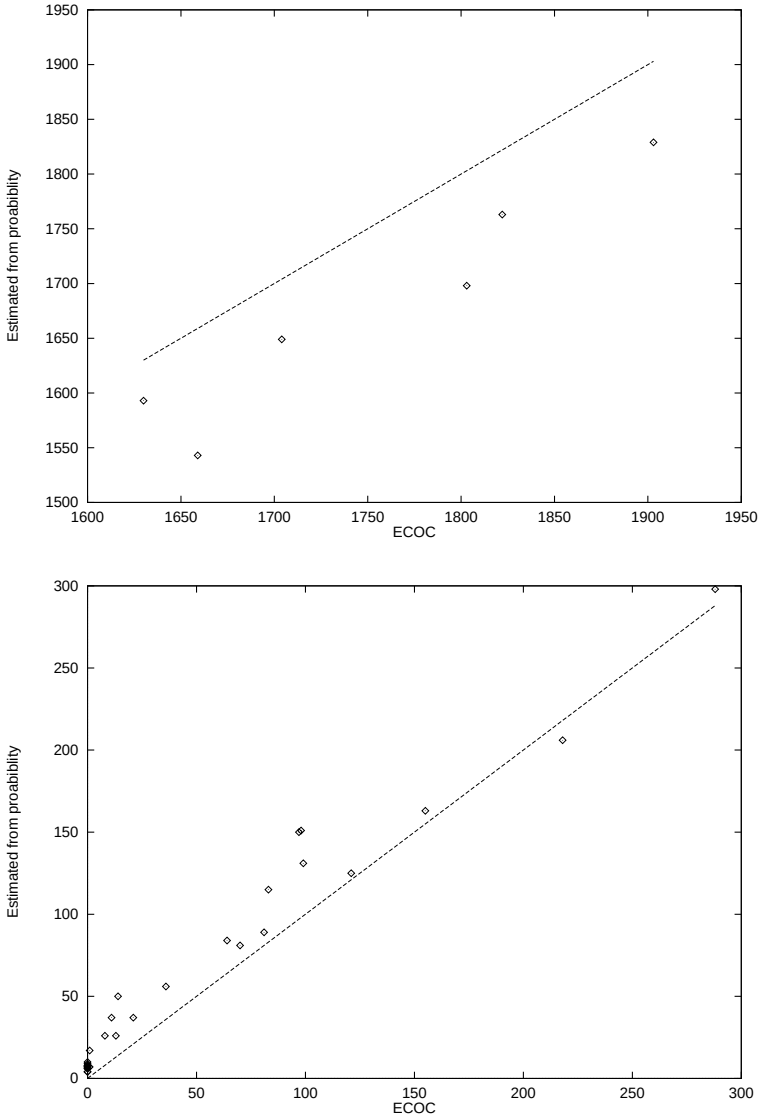


Figure 4.4. Diagonal and off-diagonal elements from confusion matrix.

4.5. REJECT OPTION

It is the doubtful classifications which contribute most to the error rate. It may be more costly to misclassify an example rather than not to classify it. In recognizing zip codes, for example, it is more economical to reject an input if doubtful and ask people to do the job, rather than to deliver a letter to the wrong place. Hence, a reject option has significant practical importance.

A rejection rule is based on the "degree of uncertainty" that a classifier has about its own classification. The posterior probability provides a concise way of expressing the uncertainty. If the two largest posterior probabilities $P(c_i|\mathbf{x})$ and $P(c_j|\mathbf{x})$ are very close, we feel less confident about the classification of $\mathbf{x}$ and consider rejecting $\mathbf{x}$.

The reject option can be formalized in terms of Bayes decision theory (Chow, 1970). Bayes decision theory (Berger, 1985) is concerned with finding the decision rule that would minimize the expected loss. Let $C = \{c_1, \ldots, c_k\}$ be the finite set of classes and $A = \{\alpha_0, \alpha_1, \ldots, \alpha_a\}$ be the finite set of possible actions. Let $\lambda(\alpha_i|c_j)$ be the loss function which denotes the cost for taking action α_i when the true class is c_j. When an input pattern $\mathbf{x}$ is observed, we can compute the *a posteriori* probability $p(c_i|\mathbf{x})$ by using the LSP method. If we take action α_j when the true class is c_i, the loss will be $\lambda(\alpha_j|c_i)$. If the input pattern is $\mathbf{x}$ and the action α_j is considered, the expected loss is

$$EL(\alpha_j|\mathbf{x}) = \sum_{i=1}^{k} \lambda(\alpha_j|c_i)p(c_i|\mathbf{x})$$

since $p(c_i|\mathbf{x})$ is the probability that the true class is c_i. Then we will select the action that minimizes the expected loss. This is the *Bayes decision rule.*

Let α_0 be the "reject" action and $\alpha_i, i = 1, \ldots, k$ be the action "classify $\mathbf{x}$ as c_i". When the loss function is defined as follows:

$$\lambda(\alpha_0|c_i) = r \quad 0 \leq r \leq 1$$

and for non-reject actions,

$$\lambda(\alpha_j|c_i) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases}$$

a reject action incurs r cost, a correct classification incurs no cost, and any error incurs a unit cost. With this loss function, the expected loss is

$$\begin{aligned} EL(\alpha_0|\mathbf{x}) &= \sum_{i=1}^{k} \lambda(\alpha_0|c_i) p(c_i|\mathbf{x}) \\ &= r \end{aligned}$$

and for $j > 0$,

$$\begin{aligned} EL(\alpha_j|\mathbf{x}) &= \sum_{i=1}^{k} \lambda(\alpha_j|c_i) p(c_i|\mathbf{x}) \\ &= \sum_{i \neq j} p(c_i|\mathbf{x}) \\ &= 1 - p(c_j|\mathbf{x}). \end{aligned}$$

So, the rule becomes to choose α_j to minimize $EL(\alpha_j|\mathbf{x})$. The reject rule can be stated as

$$\text{if } 1 - \max p(c_i|\mathbf{x}) \begin{cases} < r, \text{ then classify } \mathbf{x} \\ > r, \text{ then reject } \mathbf{x} \end{cases}$$

where r is a threshold. The larger r is, the smaller will be the rejection region and the more input patterns will be classified.

The LSP estimate of probabilities in ECOC is tested in rejection experiments. Figure 4.5 shows the rejection curve for the problem of Figure 4.1. The figure

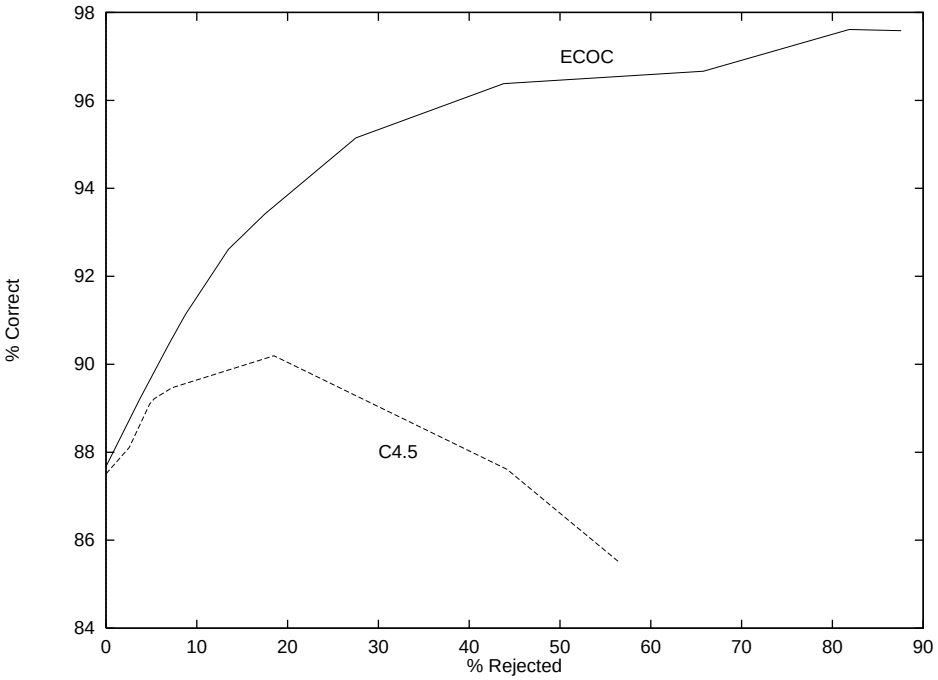


Figure 4.5. Rejection curve for the problem of Figure 4.1

shows the performance improvement by employing reject option with the probability estimate. The figure also plots the rejection curve of C4.5. This figure shows that probability estimate in ECOC is much more reliable than that of C4.5.

5. CONCLUSION

In this thesis, we have investigated why the ECOC technique works by employing the statistical notions of bias and variance. We have also presented a new technique for computing the *a posteriori* probabilities.

5.1. WHY ECOC WORKS

The primary contribution of the dissertation was to explain why the ECOC technique works.

We demonstrated from both an information-theory perspective and a decision-boundary perspective that the decoding step of ECOC is a form of voting.

Voting can reduce errors only if the errors committed by the voters are somewhat uncorrelated. 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 the work on the "boosting" method and the work 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. From both an information-theory and a decision-boundary perspectives, it was demonstrated that in ECOC the error correlation between pairs of binary functions is quite low.

Homogeneous voting (i.e., applying the same algorithm to replicates of the same problem created through bootstrapping or injecting randomness) only reduces the variance of the algorithm. We showed this by relating it to our definition of variance.

Each binary learning problem defined during ECOC causes C4.5 to make different bias errors, because a different subset of the decision boundaries is being learned in each bit. The different bias errors of C4.5 result from interactions between distant training examples (and distant decision boundaries). Unlike homogeneous voting, ECOC voting therefore is able to correct bias errors as well as variance errors.

In our experiment, the performance improvement of ECOC was comparable to that of the bootstrap aggregating procedure. The variance reduction for C4.5 using ECOC was smaller than that achieved by using Breiman's 200-replicate bootstrap aggregating, since many fewer hypotheses were participating in voting in ECOC. This means that there is room for further performance improvement by reducing the variance. By combining the ECOC method with bootstrap voting, we introduced a new technique that achieved the best performance of all.

From our analysis, we can predict that other algorithms with non-local behavior (such as backpropagation with feed-forward sigmoidal networks or the perceptron algorithm) will also benefit from error-correcting output coding. However, algorithms such as the nearest neighbor method 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 & Dietterich (1992), where

ECOC gave only a very small improvement with the generalized radial basis function (GRBF) algorithm (and this could have been the result of variance reduction rather than bias reduction).

5.2. COMPUTING POSTERIOR PROBABILITIES FROM ECOC

The second contribution of the dissertation was to show how class probability values (posterior probabilities) can be computed with ECOC.

We assume that the underlying algorithm, such as C4.5 and backpropagation with mean-square error objective function, can provide probability estimates. Using these underlying probability estimates for the binary functions, we set up an overdetermined system of linear equations relating these probabilities to the desired class probabilities. We solve this system by constrained least squares methods.

In many real-world problems, providing accurate and useful probabilistic information is a fundamental task. For example, in clinical medicine the uncertainty over which treatment decisions to make is conveniently formulated in terms of the posterior probabilities of disease diagnoses. By providing the posterior probabilities of classes, we expect that the ECOC technique will become more useful and widely applied.

5.3. FUTURE WORK

To further enhance our understanding of the capabilities of ECOC, we need to test the method on more synthetic data to verify the results under a wide range of conditions (e.g., noise, higher dimensional data, different training set sizes).

We introduced a new technique by combining the ECOC method with bootstrap voting. This new method appears promising. To validate the method, more experiments on real-world data sets are needed.

Another important open problem is to develop methods for designing good codes. Good codes for ECOC are ones that make individual functions less correlated and Hamming distances between codewords as long as possible. Many codes, including BCH error-correcting codes and random codes, were used in the previous research. More research is needed to develop automatic methods for designing optimal codes for ECOC.

Large scale voting and error-correcting codes require huge amounts of memory and CPU time to evaluate hypotheses. An important open problem for future research is to find ways of converting ECOC (and other collections of "voted" hypotheses) into representations that are smaller and more efficient to evaluate.

We tested the computed posterior probabilities with overlapping classes and a simple reject option task. We need to test the method on a broader range of problems. We also need more study on the assessment of the reliability of the posterior probabilities of classes.

Another interesting open problem is to determine whether the method can correct for errors in the probability estimates of the underlying binary algorithm.

BIBLIOGRAPHY

Agrawala, A. K. (Ed.). (1977). *Machine Recognition of Patterns*. IEEE Press, New York.

Bakiri, G. (1991). *Converting English Text to Speech: A Machine Learning Approach*. Ph.D. thesis, Department of Computer Science, Oregon State University.

Battiti, R., & Colla, A. M. (1994). Democracy in neural nets: Voting schemes for classification. *Neural Networks*, *7*(4), 691–708.

Baum, E. B., & Haussler, D. (1989). What size net gives valid generalization. *Neural Computation*, *1*, 151–160.

Benediktsson, J. A., & Swain, P. H. (1992). Consensus theoretic classification methods. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*(4), 688–704.

Berger, J. O. (1993). *Statistical Decision Theory and Bayesian Analysis* (Second edition). Springer-Verlag, New York.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, *24*, 377–380.

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

Breiman, L. (1993). Stacked regressions. Tech. rep., Department of Statistics, UC Berkeley.

Breiman, L. (1994). Bagging predictors. Tech. rep., Department of Statistics, UC Berkeley.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman-Soulie, F., & Herault, J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag.

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

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, *2*, 63–73.

Buntine, W., & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning*, *8*, 75–85.

Carnevali, P., & Patarnello, S. (1987). Exhaustive thermodynamical analysis of boolean learning networks. *Europhysics Letters*, *4*(10), 1199–1204.

Chou, P. A. (1991). Optimal partitioning for classification and regression trees. *IEEE Transactions on PAMI*, *13*, 340–354.

Chow, C. K. (1970). On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, *IT-16*, 41–46.

Clearwater, S., & Provost, F. (1990). Rl4: A tool for knowledge-based induction. In *Proceedings of the Second International IEEE COnfrence on Tools for Artificial Intelligence*, pp. 24–30. IEEE Computer Society Press.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*, 21–27.

Cover, T. M., & Thomas, J. A. (1991). *Elements of Information Theory*. Wiley, New York.

Crawford, S. L. (1989). Extensions to the cart algorithm. *International Journal of Man-Machine Studies*, *31*, 197–217.

Cun, Y. L., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagtion network. In *Advances in Neural Information Processing Systems 2*, pp. 396–404 San Mateo, CA. Morgan Kaufmann.

den Broeck, C. V., & Parrondo, J. M. R. (1993). Generalization error in a self-similar committee machine. *Physical Review Letters*, *71*(15), 2355–2359.

Denker, J. S., & LeCun, Y. (1991). Transforming neural-net output levels to probability distributions. In *Advances in Neural Information Processing Systems 3*, pp. 853–859 San Mateo, CA. Morgan Kaufmann.

Dietterich, T. G. (1990). Machine learning. In *Annual Review of Computer Science*, Vol. 4, pp. 255–306. Annual Reviews Inc., Palo Alto, Calif.

Dietterich, T. G., & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*.

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

Dietterich, T. G., & Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. In *Submitted to the International Conference on Machine Learning Conference*.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., & Vapnik, V. (1994a). Boosting and other machine learning algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 53–61.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., & Vapnik, V. (1994b). Boosting and other ensemble methods. *Neural Computation*, *6*, 1289–1301.

Drucker, H., Schapire, R., & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems 5*, pp. 42–49 San Mateo, CA. Morgan Kaufmann.

Duda, R., & Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York.

Efron, B., & Tibshirani, R. (1993). *An Introduction to the Bootstrap*. Chapman and Hall.

Fix, E., & Hodges, J. L. (1951). Discriminatory analysis–nonparametric discrimination: consistency properties. Tech. rep. Report N0. 4, Randolph Field, Texas: U. S. Air Force School of Aviation Medicine. Reprinted as pp. 280–322 of Agrawala, 1977.

Freund, Y. (1992). An improved boosting algorithm and its implications on learning cmplexity. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pp. 391–398 New York, NY. ACM Press.

Freund, Y. (1995). Boosting a weak learning algorithm by majorty. *Information and Computation*.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition* (Second edition). Academic Press, Boston.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

Hand, D. (1981). *Discrimination and Classification*. Wiley, Chichester.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(10), 993–1001.

Haussler, D. (1992). Decision theoretic generalization of pac model for neural net and other learning applications. *Information and Computation*, *100*, 78–150.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*(1), 79–87.

Jefferys, W. H., & Berger, J. O. (1992). Ockham's razor and bayesian analysis. *American Scientist*, *80*, 64–72.

Jordan, M. I. (1994). A statistical approach to decision tree modeling. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pp. 13–20.

Kearns, M., & Valiant, L. G. (1989). Cryptographic limitations on learning boolean formula and infinite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pp. 433–444.

Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press.

Kong, E. B., & Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Submitted to the International Conferenceon Machine Learning Conference*.

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.

Lawson, C. L., & Hanson, R. J. (1974). *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, N.J.

LeBlanc, M., & Tibshirani, R. (1993). Combining estimates in regression and classification. Tech. rep., Dep't of Preventive Medicine and Biostatistics and Dep't of Statistics, Univ. of Toronto.

Lippman, R. P. (1989). Pattern classification using neural networks. *IEEE Communications Magazine*, *11*, 47–64.

McCelland, J. L., & Rumelhart, D. E. (1986). *Explorations in Parallel Distributed Processing*. MIT Press.

Meir, R. (1994). Bias, variance and the combination of estimators; the case of linear least squares. Neuroprose archive article.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, *3*, 319–342.

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computation Geometry*. MIT Press.

Mitchell, T. M. (1980). The need for bias in learning generalizations. Tech. rep., Dept. of Computer Science, Rutgers University.

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

Natarajan, B. K. (1991). *Machine Learning - A Theoretic Approach*. Morgan Kaufmann.

Nilsson, N. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York.

Perrone, M. (1993). *Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization*. Ph.D. thesis, Brown University, Inst. for Brain and Neural Systems.

Perrone, M., & Cooper, L. (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.

Peterson, W. W., & Weldon, E. J. (1972). *Error-Correcting Codes* (Second edition). MIT Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Reciples in C*. Cambridge University Press.

Qian, N., & Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, *202*, 865–884.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1987). Decision trees as probabilistic classfiers. In *Proceedings of the Fourth International Workshop on Machine Learning*, pp. 31–37. Morgan Kaufmann.

Quinlan, J. R. (1993). *C4.5: Program for Empirical Learning*. Morgan Kaufmann, San Mateo, CA.

Richard, D. M., & Lippmann, R. P. (1991). Neural network classifiers estimate bayesian *a posteriori* probabilities. *Neural Computation*, *4*(3), 461–483.

Ripley, B. D. (1993). Statistical aspects of neural networks. In Barndorff-Nielsen, O. E., Jensen, J. L., & Kendall, W. S. (Eds.), *Networks and Chaos - Statistical and Probabilistic Aspects*, pp. 40–123. Chapman and Hall, London.

Ripley, B. D. (1994). Neural networks and related methods for classification. *Royal Statistical Society*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In *Explorations in Parallel Distributed Processing*, Vol. 1. MIT Press.

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, *10*, 153–178.

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

Schapire, R. E. (1992). *The Design and Analysis of Efficient Learning Algorithms*. MIT Press.

Schwartz, D. B., Samalam, V. K., Solla, S. A., & Denker, J. S. (1990). Exhaustive learning. *Neural Computation*, *2*(3), 374–385.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, *1*, 145–168.

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

Shavlik, J., & Dietterich, T. G. (1990). *Readings in Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In Carbonell, J. G., Michalski, R., & Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. II. Morgan Kaufmann.

Valiant, L. G. (1984). A theory of the learnable. *Communication of the ACM*, *27*(11), 1134–1142.

Wan, E. (1990). Neural network classification: A bayesian interpretation. *IEEE Transactions on Neural Networks*, *1*(4), 303–305.

Watkin, T., Rau, A., & Biehl, M. (1993). The statistical mechanics of learning a rule. *Review of Modern physics*, *65*(2), 499–556.

Wettschereck, D., & Dietterich, T. G. (1992). Improving the performance of radial basis function networks by learning center locations. In *Advances in Neural Information Processing Systems 4*, pp. 1133–1140 San Mateo, CA. Morgan Kaufmann.

Wolpert, D. (1992). Stacked generalizarion. *Neural Networks*, *5*, 241–159.

Xu, L., Krzzyzak, A., & Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*(3), 418–435.