A NEW PERSPECTIVE ON CLASSIFICATION

by

Guohua Zhao

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Mathematical Sciences

UTAH STATE UNIVERSITY
Logan, Utah

2000
ABSTRACT

A New Perspective on Classification

by

Guohua Zhao, Doctor of Philosophy
Utah State University, 2000

Major Professor: Dr. Adele Cutler
Department: Mathematics and Statistics

The idea of voting multiple decision rules was introduced into statistics by Breiman. He used bootstrap samples to build different decision rules, and then aggregated them by majority voting (bagging). In regression, bagging gives improved predictors by reducing the variance (random variation), while keeping the bias (systematic error) the same. Breiman introduced the idea of bias and variance for classification to explain how bagging works. However, Friedman showed that for the two-class situation, bias and variance influence the classification error in a very different way than they do in the regression case.

In the first part of the dissertation, we build a theoretical framework for ensemble classifiers. Ensemble classifiers are currently the best off-the-shelf classifiers available, and they are the subject of much current research in classification. Our main theoretical results are two theorems about voting iid (independently identically distributed) decision rules. The bias consistency theorem guarantees that voting will not change the Bias set, and the convergence theorem gives an explicit rate of convergence. The two theorems explain exactly how ensemble classifiers work. We also introduce the concept of weak consistency as opposed to the usual strong consistency. A boosting theorem is derived for a distribution-specific situation with iid voting.
In the second part of this dissertation, we discuss a special ensemble classifier called PERT. PERT is a voted random tree classifier for which each random tree classifies every training example correctly. PERT is shown to work surprisingly well. We discuss its consistency properties. We then compare its behavior to the NN (nearest neighbor) method and boosted c4.5. Both of the latter methods also classify every training example correctly. We call these types of classifiers “oversensitive” methods. We show that one reason PERT works is because of its “squeezing effect.”

In the third part of this dissertation, we design simulation studies to investigate why boosting methods work. The outlier effect of PERT is discussed and compared to boosted and bagged tree methods. We obtain a new criterion (Bayes deviance) that measures the efficiency of a classification method. We design simulation studies to compare the efficiency of several common classification methods, including NN, PERT, and boosted tree method.
ACKNOWLEDGMENTS

I would like to thank Dr. Adele Cutler, my major professor and thesis advisor, who introduced me into this exciting area of research in the summer of 1997. It is a real happy experience to have a nice major professor who helped me not only in research work, but also in general working ethics. Without her careful guidance and consistent encouragement, this research work would have been impossible. I would also like to thank her for her patient and careful editing effort, which resulted in improved writing.

I would also like to thank my committee members, Dr. Richard Cutler, Dr. Renate Schaaf, Dr. Juergen Symanzik, and Dr. Nicholas Flann, for their guidance and support throughout the entire process.

Finally I would like to thank our department, which provided me with the financial support while I was pursuing this research. Encouragement and help from my fellow graduate students are also gratefully acknowledged.

Guohua Zhao
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>1 INTRODUCTION AND LITERATURE REVIEW</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Basic Setting for Classification</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Some Statistical Classification Methods</td>
<td>6</td>
</tr>
<tr>
<td>1.2.1 Nearest Neighbor Method</td>
<td>6</td>
</tr>
<tr>
<td>1.2.2 Linear Discriminant Analysis</td>
<td>7</td>
</tr>
<tr>
<td>1.2.3 Logistic Regression Classifiers</td>
<td>7</td>
</tr>
<tr>
<td>1.2.4 Tree Methods</td>
<td>8</td>
</tr>
<tr>
<td>1.2.5 Neural Networks</td>
<td>8</td>
</tr>
<tr>
<td>1.3 Bagging, Boosting, and Bias-Variance Decomposition</td>
<td>9</td>
</tr>
<tr>
<td>1.3.1 Classification Bias and Variance Interpretation</td>
<td>12</td>
</tr>
<tr>
<td>1.3.2 Theoretical Interpretation: The Convergence Property</td>
<td>14</td>
</tr>
<tr>
<td>1.3.3 Empirical Interpretation: Margin and Diversity</td>
<td>15</td>
</tr>
<tr>
<td>1.4 Random iid Ensembles and Perfect Random Trees</td>
<td>16</td>
</tr>
<tr>
<td>1.4.1 Independent Ensemble Methods</td>
<td>16</td>
</tr>
<tr>
<td>1.4.2 Perfect Random Tree</td>
<td>18</td>
</tr>
<tr>
<td>1.5 Summary and Our Work</td>
<td>19</td>
</tr>
<tr>
<td>2 A LEARNING SYSTEM</td>
<td>22</td>
</tr>
<tr>
<td>2.1 A Posterior Measure on Rule Spaces</td>
<td>22</td>
</tr>
<tr>
<td>2.2 The Learning System</td>
<td>25</td>
</tr>
<tr>
<td>2.2.1 $q$ Surfaces and Learning System</td>
<td>27</td>
</tr>
<tr>
<td>2.2.2 Instability and Diversity</td>
<td>32</td>
</tr>
<tr>
<td>2.3 The Effects of Bias and Variance on Generalization Error</td>
<td>35</td>
</tr>
<tr>
<td>2.3.1 Bayes Deviance for Classification</td>
<td>36</td>
</tr>
<tr>
<td>2.3.2 Extension of Friedman’s Idea</td>
<td>37</td>
</tr>
<tr>
<td>2.3.3 Conclusion</td>
<td>41</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table                                              Page
4.1 Simulation 1. A Comparison Between PERT and NN   86
4.2 Simulation 2. Misclassification Error as Sample Size Increases  87
4.3 Comparison of Two Variants of PERT and Other Methods  100
5.1 Error Rates With and Without 5% Outliers.           106
5.2 Percentage of Error Increase When 5% Outliers Are Added  106
5.3 Convergence Rate for Different Methods              109
5.4 Relative Efficiency for PERT                        111
5.5 Relative Efficiency for Boosted CART                111
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A learning process.</td>
<td>16</td>
</tr>
<tr>
<td>2.1 The training data $\rightarrow$ Q surface $\rightarrow$ decision rule</td>
<td>24</td>
</tr>
<tr>
<td>2.2 The estimated $q$ surface and margin surface for tree methods, XOR data</td>
<td>28</td>
</tr>
<tr>
<td>2.3 An explanation plot for relationship between variance and the classification error, theoretical $p &gt; 0.5$ case</td>
<td>39</td>
</tr>
<tr>
<td>3.1 The $q_n(1</td>
<td>x)$ versus $n$</td>
</tr>
<tr>
<td>3.2 The lower limit for $q$ so that adding the two rules will help</td>
<td>65</td>
</tr>
<tr>
<td>3.3 Big K value is favored for k-NN method, the case of nonzero Bayes error</td>
<td>67</td>
</tr>
<tr>
<td>3.4 Voting three lines can separate four points</td>
<td>70</td>
</tr>
<tr>
<td>3.5 The data and the Q surface for voting random hyperplanes</td>
<td>73</td>
</tr>
<tr>
<td>3.6 The decision boundary and learning curve for voting random hyperplanes</td>
<td>74</td>
</tr>
<tr>
<td>3.7 The decision boundaries of PERT and NN method</td>
<td>75</td>
</tr>
<tr>
<td>4.1 An explanation plot for the calculation of the probability that a vertical split falls in the 1-th vertical interval</td>
<td>80</td>
</tr>
<tr>
<td>4.2 Influence of one outlier for NN and PERT</td>
<td>90</td>
</tr>
<tr>
<td>4.3 Bayes deviance versus Bayes error for PERT</td>
<td>91</td>
</tr>
<tr>
<td>4.4 2-dimensional training data from the overlapping normal</td>
<td>93</td>
</tr>
<tr>
<td>4.5 Different behavior as dimensionality of feature space increases</td>
<td>94</td>
</tr>
<tr>
<td>4.6 Three fundamental waveforms for the waveform data</td>
<td>99</td>
</tr>
<tr>
<td>5.1 The relationship between Bayes deviance $D_B$ and Bayes error $E_B$ for NN, PERT, and boosted CART dimension=2</td>
<td>103</td>
</tr>
<tr>
<td>5.2 The relationship between Bayes deviance $D_B$ and Bayes error $E_B$ for NN, PERT, and boosted CART dimension=8</td>
<td>103</td>
</tr>
</tbody>
</table>
5.3 The log-transformed Bayes deviance versus log-sample size, for NN, PERT, and boosted CART dimension=4 and 8 . 110
CHAPTER 1
INTRODUCTION AND LITERATURE REVIEW

1.1 Basic Setting for Classification

Classification is an interesting and important branch in statistics. It is closely related to pattern recognition, machine learning, and data mining. One major goal for classification is to find a prediction rule. Lots of examples fall into this category of research. Examples include: machine identification of zip codes, recognition of human pronunciation, identification of patients with diabetes, credit card approval, etc.

For example, when a machine tries to learn whether a written digit is 1 or 2, it needs to know the characteristics of these numbers. This information is usually provided with a collection of data whose class labels are already known. In a digit recognition example, each digit can be represented by a $16 \times 16$ matrix of 1’s or 2’s, representing whether or not there are black spots on the pixel. The matrix is the “feature vector” and has 256 elements. The “class label” is the correct digit. When the machine is presented with examples whose feature matrices and class label are both known, it finds a rule to assign class labels for new examples. In statistics, the collection of data used to develop a decision rule is called the training data. A classification method is one that generates a decision rule for any given training data. Detailed notation is summarized below.

Let training data \( T = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \} \), where each \( x_i \) is a feature vector, and each \( x_i \) is considered to be one realization of a random vector \( X \). Feature space \( \mathcal{X} \) is the set of all feature vectors. Let \( k \) be the number of different classes. Each \( y_i \in \mathcal{Y} = \{0, 1, \ldots, k - 1\} \) is the class label. Again each \( y_i \) is a realization of a random variable \( Y \). Furthermore, \( X \) is assumed to come from a mixture distribution, i.e., the probability density of \( X \) is

\[
 f_X(x) = \sum_{i=0}^{k-1} \pi_i f_i(x) \tag{1.1}
\]
with \( \pi_i, i = 0, 1, \ldots, k - 1 \), the prior for each of the classes, \( f_i(x) \) the probability density for class "i", and \( F_X(x) \) the cumulative density of \( X \).

**Definition 1.1.1 (Data structure)**

A data structure is a triplet \( \{\mathcal{X}, \pi, f\} \), where \( \mathcal{X} \) is a feature space, \( \pi = (\pi_0, \pi_1, \ldots, \pi_{k-1})^T \) are the priors for the class label \( Y \), \( f = (f_0, f_1, \ldots, f_{k-1})^T \), \( k \) is the total number of classes, and \( f_X(x) \) is given by (1.1).

Notice when a data structure is given, the information about the data is complete. The joint density \( f_{X,Y}(x,i) \) of \( X \) and \( Y \) can be calculated. Further the posterior probability \( P(Y = i|x) \) at any point \( x \) can be derived by using Bayes Theorem:

1. \( f_{X,Y}(x,i) = \pi_i f_i(x) \).
2. \( P(Y = i|x) = \frac{\pi_i f_i(x)}{\sum_{i=0}^{k-1} \pi_i f_i(x)} \).

For simplicity, \( p(i|x) \) is used to denote \( P(Y = i|x) \). The definition of Bayes rule (Definition 1.1.3) depends on this posterior probability.

When training data are given, the examples in the training data are assumed to be drawn iid (independently identically distributed) from a supposedly fixed but unknown data structure/data distribution. A decision rule \( r \) is a mapping from the feature space \( \mathcal{X} \) to the set of class labels \( \mathcal{Y} \). The goal in classification is to use the training data to develop a decision rule that can make accurate prediction for new examples. The definition of misclassification is given below.

**Definition 1.1.2 (Misclassification error)**

Assume the data structure \( \mathcal{D} = \{\mathcal{X}, \pi, f\} \) is given, the misclassification error (also known as generalization error) of a decision rule \( r \) is defined as:

\[
E(r) = P_{X,Y}(r(X) \neq Y).
\]

The prediction accuracy is:

\[
P_{X,Y}(r(X) = Y) = 1 - E(r).
\]
In order to compare two classification methods, a good estimate of the generalization errors are necessary. Some classification methods may work better than others for one data structure, while not working as well for other data structures. This is clearly shown by the Statlog Project (Michie et al. [1994]), which compared 20 different classification methods on 20 datasets. Therefore when we compare two methods, we need to compare two methods on the same data structure.

For given training data $T = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \}$, a classification method $M$ produces a decision rule, denoted by $r(., T, M)$ to explicitly show the dependence on $T$ and $M$. The so-called training error is a handy estimate of the generalization error. It is defined as:

$$\text{training error} = \frac{1}{n} \# \{ i : r(x_i, T, M) \neq y_i, (x_i, y_i) \in T \}. \tag{1.4}$$

It is well known that the training error is a biased estimate of the generalization error. We build a decision rule from training data and then evaluate the performance of the rule on the same training data, so our estimate is optimistic.

A better strategy is to evaluate the decision rule on a set of samples that are independent from the training data. Such an independent dataset is called a testset. The resulting error estimate is accordingly called testset error. It is clear that the testset error is an unbiased estimate of the generalization error.

In most applications, only one data set is given. In order to estimate the testset error, we can partition the data into two groups. One group is used as training data, the other group used as the test data. Breiman [1998a] used this method. Another method is called $v$-fold cross-validation (Breiman & Peters [1992]). In $v$-fold cross-validation, the data is partitioned into $v$ groups. The classification method is fit using data from $v - 1$ of the groups and the testset error is obtained for the remaining group. The generalization error is estimated by the average of the $v$ testset errors. Freund and Schapire [1996] and Quinlan [1996] used this CV method to compare different classification methods.

The Bayes rule $C_B(x)$ is a heuristic decision rule. It is obtained under the assumption
that the true probability density is already known. Assuming that the data structure $\mathcal{D} = \{\mathcal{X}, \pi, f\}$ is given, the Bayes rule $C_B(x)$ and its generalization $E_B$ are defined as follows.

Definition 1.1.3 (Bayes rule, Bayes error)

Given a data structure $\mathcal{D} = \{\mathcal{X}, \pi, f\}$, the Bayes rule $C_B$ and its misclassification error $E_B$ are defined as:

$$C_B(x) = \arg\max_i p(i|x) = \arg\max_i P(Y = i|x),$$

$$E_B = P_{X,Y}(C_B(X) \neq Y) = \int_{\mathcal{X}} P_Y(C_B(x) \neq Y) p(dx).$$

where $\arg\max_i f(i) = i_0$ if $f(i_0) > f(i)$ for any $i \neq i_0$.

The Bayes rule predicts the class which has the largest posterior probability. The Bayes error represents the intrinsic difficulty in discriminating different classes. When the classes are perfectly separable, in other words the supports of the densities do not overlap, the Bayes error is zero. Otherwise, the Bayes error will be positive. To further explore the idea, we define the pointwise error, i.e., the error $e_b(x) = P(C_B(x) \neq Y)$, for any $x$. Notice that this is the integrand of formula (1.6). Suppose at some point $x$, $p(0|x) = 0.6, p(1|x) = 0.4$. The Bayes rule predicts $x$ to be of class “0”. However, it may actually come from class “1”. When this happens, the Bayes rule makes an incorrect prediction, the probability of which is 0.4. The Bayes error $E_B$ is the total of these chances where $x$ comes from the less likely class.

Both Bayes rule and Bayes error are important in the theoretical analysis of classification methods. It is well known that the Bayes rule has the smallest misclassification error among all decision rules. Any classifier that has the same classification error will also be optimal. Usually a method will not produce a decision rule that is as good as the Bayes rule, but it is desirable to find decision rules that will asymptotically achieve the Bayes error. This is known as the consistency problem. One consistency definition from Devroye et al. [1996] is given below. Discussions on other forms of consistency will be given in Chapter 3.
**Definition 1.1.4 (Strong consistency)**

Given a data structure $\mathcal{D}$, let $T_n$ be training data of size $n$, drawn from this data distribution, and denote $E_n = P_{X,Y}(r(X,T_n,M) \neq Y)$, the generalization error of the decision rule obtained by applying $M$ to $T_n$. A classification method $M$ is called (strongly) consistent if

$$E_n \rightarrow E_B, \text{ with probability 1.}$$

A classification method that is consistent is also called asymptotically efficient.

In real life, the true posterior density $p(i|x)$ or the class conditional probability $f_i(x)$ or the prior $\pi_i$ for each class is not known. However, they can be estimated from the given training data. Each $\pi_i$ can be estimated by the frequency of the class, and each $f_i(x)$ can be estimated from the data of class $i$. The density estimation methods can be parametric methods or nonparametric methods such as histograms, kernel methods, or k-nearest neighbor methods. Let $\hat{\pi}_i$ and $\hat{f}_i(x|T)$ be the estimates of $\pi_i$ and $f_i(x)$, respectively, $\hat{p}(i|x)$ can be calculated and a decision rule can be obtained by substituting the theoretical quantities by their estimates in the formula in Definition 1.1.3. This is called the plug-in Bayes rule.

**Definition 1.1.5 (Plug-in Bayes rule)**

Given data structure $\mathcal{D} = \{X, \pi, f\}$ and training data $T$, a plug-in Bayes rule is defined as:

$$C_{plug-in}(x) = \arg\max_i \hat{p}(i|x,T), \text{ where } \hat{p}(i|x) = \frac{\hat{\pi}_i \hat{f}_i(x|T)}{\sum_{i=0}^{k-1} \hat{\pi}_i \hat{f}_i(x|T)}.$$  

The Plug-in Bayes rule is a very general method for finding a classification rule. For 2-class problems with classes $\{0, 1\}$, the plug-in rule is reduced to $C_{plug-in} = I(\hat{p}(1|x) > 0.5)$, where $I(x) = 1$, if $x$ is TRUE, 0 if $x$ is FALSE. Friedman [1997] provides an insightful investigation of how plug-in methods work for two class problems. We will further discuss their behavior in Chapter 2.
1.2 Some Statistical Classification Methods

In this section, classification methods that will be used in this dissertation are reviewed. As all these methods are fairly common, we only give quite short introductions. For in-depth discussions, the readers are referred to Michie et al. [1994], Ripley [1996], and Devroye et al. [1996]. The latter reference contains theoretical discussions of many classification methods.

When presented with training data, the goal for a classification method is to output a decision rule. When new data are provided, the decision rule then predicts class labels for them. On many occasions, the most important factor for a classification method is its prediction accuracy. However, there are also other concerns such as time efficiency, storage efficiency, and comprehensibility, etc. Current applications of classification (e.g., data mining) routinely involves large datasets. In this case, it is very important that a classification method should be fast and memory-saving in addition to being accurate. It may also be important for the method to be robust to a small proportion of incorrect labels in the training data.

1.2.1 Nearest Neighbor Method

Given data $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the nearest neighbor (NN) method predicts a new data point $x$ as being from the class of the training point which is closest to $x$, i.e.,

$$NN(x, T) = y_i, \quad \text{where } i = \text{argmin}_i \text{ Dist}(x, x_i).$$

Here Dist is some metric in the feature space. NN is simple, and the decision boundary is piecewise linear. NN suffers from the curse-of-dimensionality (Bellman [1961], Friedman [1997]). When the dimension of the feature space is high, a large amount of training data is needed to achieve some given prediction accuracy, while for lots of other methods, less data is needed to achieve the same prediction accuracy. Another disadvantage about NN is that it does not summarize well. We have to keep all the training points to predict for a new point.
The k-NN method uses k neighbors in predicting for a new point. For a given point \( x \), the \( k \) nearest neighbors to \( x \) are picked, and then \( x \) is predicted to be from the majority class of these \( k \) points. When there are ties, a random break-up is used. Fix and Hodges [1951] showed that if

\[
k(n) \to \infty, \quad \frac{k(n)}{n} \to 0,
\]

then k-NN is strongly consistent as defined in section 1.1.

1.2.2 Linear Discriminant Analysis

Assume that for each class \( i \), the data follows a Normal distribution \( N(\mu_i, \Sigma) \). The linear discriminant classifier predicts \( x \) to be from the class \( i \), which gives the largest \( LDA_i \):

\[
LDA_i = \log(\pi_i) + \mu_i^T \Sigma^{-1} (x - \frac{1}{2} \mu_i)
\]

where \( \pi_i \) and \( \mu_i \) are the prior probability and mean for class \( i \), and \( \Sigma \) is the common covariance matrix, all of which need to be estimated from the training data. For the 2-class question, it suffices to know whether \( LDA_1 - LDA_0 > 0 \). LDA predicts \( x \) to be of class “1” if:

\[
d_{1,0} = (\mu_1 - \mu_0) \Sigma^{-1} (x - \frac{\mu_0 + \mu_1}{2}) - \log \frac{\pi_1}{\pi_0} > 0.
\]

When the assumption of normality is true, and the variance-covariance matrices of the different classes are the same, LDA is known to work very well.

1.2.3 Logistic Regression Classifiers

Logistic model has the same assumption as linear discriminant analysis. The \( \mu \)'s and \( \Sigma \) are not estimated by their sample versions. Instead, the linearity of the following model is assumed.

\[
\log \frac{p(i|x)}{p(1|x)} = \alpha_i + \beta_i^T x.
\]

Then maximum likelihood estimation can be used to estimate the parameters. After the parameters are estimated, the prediction at a new point \( x \) is the class “i” which has
the largest $\hat{\alpha}_i + \hat{\beta}_i^T x$. Efron [1975] showed that logistic classification usually has better prediction accuracy than LDA.

### 1.2.4 Tree Methods

Tree methods aim to partition the feature space into hypercubes. In each hypercube, the predicted class will be the majority class of those training points that fall in the hypercube. Two immediate questions about tree methods are:

1. **Splitting criteria**: How to choose a variable to split and where to split?

2. **Stopping rule**: When to stop splitting?

CART (Breiman et al. [1984]) and c4.5 (Quinlan [1993]) are the two most commonly used commercial methods. The Gini Criterion (Breiman et al. [1984]) is one commonly used criterion for splitting the nodes. It is a kind of “impurity measure.” Given a data structure $D = \{X, \pi, f\}$, the Gini impurity at the current node $t$ is defined as:

$$Gini(t) = \sum_{i \neq j} p(i|t)p(j|t).$$

where $p(i|t)$ is the proportion of class “i” examples in the node. Notice that $Gini(t) \geq 0$. It equals to 0 if all examples from one same class, and the maximum is achieved when all classes have same proportions, which is the most “chaotic” case.

When splitting a tree, we choose from all possible variables and all possible splits to find the one that has the smallest impurity. The current node is then partitioned into two nodes. The same procedure is carried for each of them until a certain stopping criteria is achieved. Normally, some of the nodes can be combined by pruning. Details about the tree methodology can be found in the references given above.

### 1.2.5 Neural Networks

Neural networks are also called artificial neural networks (ANN). They are motivated by the desire to imitate the behavior of human brain. It is believed that the human brain
processes information in a massively parallel manner. When the brain receives information, it processes it through several layers of nodes. A single layer feed-forward neural network for the L-class classification problem has the following form:

\[ y_l = \phi \left( a_{0l} + \sum_{j=1}^{n} a_{jl} \phi (b_{0j} + \sum_{k=1}^{m} b_{kj} x_k) \right), l = 0, 1, \ldots, L - 1, \]

where the \( a_{0l} \) and \( b_{0j} \)'s are called the bias terms, and the \( a_{jl} \) and \( b_{kj} \)'s are called the weights. \( \phi \) is called the “activation function,” which often takes the form:

\[ \phi(x) = \frac{e^x}{1 + e^x}. \]

For given training data \( T = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \} \), \( y_i \) are coded as follows: \( y_i = (y_{i1}, y_{i2}, \ldots, y_{iL}) \), with \( y_{ij} = 1 \), if the \( i \)-th data point is from class “\( j \)”, and \( y_{il} = 0 \) for all other classes. The parameters are then fitted by minimizing the the residual sum of squares (RSS):

\[ RSS = \sum_{l=1}^{L} \sum_{i=1}^{n} \left( y_{il} - \phi \left( a_{0l} + \sum_{j=1}^{n} a_{jl} \phi (b_{0j} + \sum_{k=1}^{m} b_{kj} x_{ik}) \right) \right)^2. \]

Typical fitting methods for neural networks include back-propagation (Rumelhart et al. [1986]), Quickprop (Fahlman [1989]) or some general purpose optimization methods such as Quasi-Newton methods, conjugate gradient methods etc. The Splus function net (Venables and Ripley [1997]) uses the Quasi-Newton method for optimization.

1.3 Bagging, Boosting, and Bias-Variance Decomposition

The methods introduced in the previous section output one single rule for a given training set. Another methodology is to generate multiple rules and then combine them to give a final decision rule. The collection of rules is called an “ensemble.” Accordingly, the method is called an ensemble method. Recent work on Adaboost (Freund and Schapire [1996], [1997]) ignited a considerable enthusiasm in ensemble methods. Adaboost is a general scheme that can be combined with an existing classification method (called a base
learner in this context). It repeatedly applies the base learner to sequentially perturbed training data to obtain decision rules, and then finally combines these rules by weighted voting. It has been shown (Freund and Schapire [1996], Drucker and Cortes [1996], Quinlan [1996], Breiman [1998a]) that Adaboost usually improves the prediction accuracy of a base learner significantly.

Adaboost was initially proposed to solve a theoretical question raised in Kearns and Valiant [1988]. It adaptively resamples to feed to the base learner in such a way that the examples that are hard to classify will have big chances to be chosen for the next classifier. In focusing on learning these difficult examples, it then can obtain a better decision rule. Theoretical results on Adaboost will be reviewed later in this section.

Breiman [1996a] invented another scheme to improve an existing classification method. His bagging algorithm generates different bootstrap samples (Efron and Tibshirani [1993]) from the original data, then the base classifier is applied to each of these samples, and finally these individual rules are aggregated by majority voting. Bagging is very simple, and it is clear how it works. On the other hand, Adaboost is quite a complicated algorithm. The fact that Adaboost works so well has prompted people to investigate ensemble methods in a broader framework. We first introduce both the bagging and Adaboost algorithms, then review the recent investigations on them.

Given a base classifier $M$ and training data $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, for a pre-chosen iteration number $N$, bagging and boosting algorithms are defined as follows:

1. **Bagging:**
   
   Repeat step 1) and 2) $N$ times:
   
   - Generate a bootstrap version of the data.
   - Apply the base classifier $M$, and obtain the decision rule $r_i$.

   Vote these $N$ rules to give the final decision $r_{final}$.

2. **Boosting** (Adaboost.M1, Freund and Schapire [1996]):
   
   Initialize $t=0$, and the resampling probability $p_i^0 = \frac{1}{n}, i = 1, 2, \ldots, n$. 
Repeat the following steps N times:

- Resample from $p^t = \{p^t_i, i = 1, 2, \ldots, n\}$.
- Apply the base classifier $M$, and obtain the decision rule $r_t$.
- Calculate the weighted error over the training data, $\epsilon_t = \frac{1}{n} \sum_{i} r_t(x_i) \neq y_i p^t_i$.

If $\epsilon_t > 0.5$, abort the loop.

- Calculate the $\beta_t = \log(\frac{1-\epsilon_t}{\epsilon_t})$, calculate $p^{t+1}_i = p^t_i \beta_t d^t_i$, where $d^t_i = 1$, if $r_t(x_i) \neq y_i$,

  0, otherwise. Normalize $p^{(t+1)}$ by dividing each element by the sum of $p^{(t+1)}$.

The final rule is the weighted vote of the N rules, using weight $w_t = \log(\beta_t)$ for rule $r_t$.

Breiman [1996b] discussed the instability issue of classification methods. A classification method is considered to be unstable if a small change in the training data causes big changes in the resulting decision. By voting different rules, bagging will stabilize an unstable classification method. Thus bagging is considered to be a variance reduction method. This intuition is empirically shown to be true by Breiman [1998a].

Adaboost is a much more complicated algorithm. The $(t+1)$-th rule $r_{t+1}$ depends on the prediction behavior of $r_t$. It is actually shown that $r_{t+1}$ depends on all previous rules (Schapire et al. [1998]). Adaboost was derived as a special solution of the boosting problem posed by Kearns and Valiant [1994] within the PAC framework. Its exceptionally good performance ignited immediate studies. Breiman [1998a] claimed that Adaboost works because it reduces the variance part of the classification error. Breiman also claimed that Adaboost will not be able to improve stable classifiers such as LDA. At the same time, Schapire et al. [1998] showed it will also reduce the bias part of the error if Adaboost is combined with a weak classifier that does not have much fitting power. Actually from the PAC learning, it is much clearer that Adaboost will reduce the bias as Adaboost will increase the fitting power (i.e., VC-dimension) of the base learner. However it is not clear why Adaboost will reduce the variance part. It is shown (Schapire et al. [1998]) that Adaboost may even achieve a reduction in error at the price of increasing the variance. The behavior of Adaboost is very mysterious.
1.3.1 Classification Bias and Variance

Interpretation

The idea of decomposing the classification error into the bias and variance comes from the regression setup. There are several definitions for the classification bias and variance. However, none of them are as natural as their regression counterparts. We review the major definitions, and point out the problems.

In order to have a better idea of the classification bias and variance, we first look at the regression situation. In regression, we have a predictor random variable \( X \), and response variable \( Y \), and we assume the relationship between \( X \) and \( Y \) is

\[
Y = f(X) + \epsilon,
\]

where \( \epsilon \) is assumed to be \( N(0, \sigma^2) \) with \( \sigma \) unknown. The random variable \( \epsilon \) is also assumed to be independent from \( X \), and \( f \) is assumed to be in some space of functions.

Now for given training data \( T \), suppose we obtain an estimate \( \hat{f} \) of \( f \), and from this we can estimate \( Y \) by \( \hat{Y} = \hat{f}(x) \) at any point \( x \). We usually measure the prediction error by the squared error loss:

\[
P E(\hat{Y}, Y) = E(\hat{Y} - Y)^2.
\]

The independence between \( \epsilon \) and \( X \) renders the following decomposition:

\[
E(\hat{Y} - Y)^2 = E((\hat{Y} - E\hat{Y})^2 + (E\hat{Y} - EY)^2 + \sigma^2.
\]

The first term measures the random variation in \( \hat{Y} \), which is not related to the systematic part of the estimation. This is the usual variance definition. The second term measures the systematic difference between the expected \( Y \) and the expected \( \hat{Y} \), it is the squared bias. The third term is the irreducible error caused by the noise, and is not affected by the predictors. Now suppose we have multiple predictions \( \hat{Y} \), all following the same distribution. Then averaging of these to give a new predictor will give a smaller variance term, while keeping the bias term the same. For regression, bagging introduces iid predictors and therefore reduces the prediction errors.
For the classification case, the most commonly used loss function is the so-called 0/1 loss function, i.e., the loss is 1 if $\hat{Y}(x) \neq Y$, 0 otherwise. Therefore, the prediction error at point $x$ is

$$PE(\hat{Y}, Y) = EI(\hat{Y}(x) \neq Y).$$

The goal is to obtain a similar three-part decomposition. At the same time, we still want the variance to represent the random variation and the bias to represent the systematic discrepancy between the prediction and the Bayes rule. Here the systematic discrepancy is the discrepancy between the aggregated rule \cite{Breiman1998} and the Bayes rule. A careful inspection shows that none of the definitions to date satisfies the additive decomposition while still keeping the natural meaning mentioned above.

Kong and Dietterich \cite{Kong1995} have a meaningful definition for bias, but with their variance can be negative. Kohavi and Wolpert’s \cite{Kohavi1996} definition for variance does represent the variations among the decision rules, however, their bias does not represent the systematic discrepancy between the aggregated rule and the Bayes prediction. Breiman \cite{Breiman1998a} defined the Bias set as the points where the aggregated prediction will be the same as the Bayes rule. His definition for both bias and variance is defined on the whole distribution, not pointwise. This makes the definition unintuitive. His definition for variance is related to the systematic part of the distribution, and bias may be increased through voting. Tibshirani \cite{Tibshirani1996} had an insightful thought that in theory, bagging will not necessarily reduce the classification error because the 0/1 loss function is not a convex function.

Friedman \cite{Friedman1997} stepped away from the bias and variance decomposition. Instead he focused on the 2-class question, where the decision is obtained by a plug-in method, i.e., the method where we get an estimate $\hat{p}(1|x)$, then plug in to the formula $r(x) = I(\hat{p}(1|x) > 0.5)$ to form a decision rule. He investigated how the bias and variance in estimating $p(i|x)$ may influence the classification error. The result is interesting and quite unintuitive (details in section 2.3). The limitation of Friedman \cite{Friedman1997} is that the tools used there cannot deal with the multiclass situation. In section 2.3, we will extend his results to the multiclass case.
1.3.2 Theoretical Interpretation: The Convergence Property

There has been lots of other work that aims to understand boosting better. Freund and Schapire [1996], Breiman [1999b], Grove and Schuurmans [1998], Friedman et al. [1998], Schapire et al. [1998], Ho [1998] and Dietterich [1998b], Breiman [1999a] are among those that gave interesting points of view.

There are two major questions concerning Adaboost:
1. Is Adaboost trying to minimize some criteria?
2. Is the algorithm convergent?

The first four papers aimed to answer Question 1. Freund and Schapire [1996], and Breiman [1999b] viewed Adaboost as a matrix game and a prediction game respectively. They both concluded that Adaboost is fitting a minimax rule. Grove and Schuurmans [1998] used linear programming (this is implied by Breiman [1997]) to solve the minimax problem. These three methods assume that $N$ decision rules are already obtained, and they view adaboost as a way to generate the best weighting strategy. Friedman et al. [1998] viewed Adaboost as a logistic additive model. This view is closer to the original adaboost algorithm, as it does not assume that the decision rules are already there.

Breiman discussed the reason why Adaboost is resistant to overfitting, ie, why the testset error will not deteriorate as training continues. He conjectured that the stochastic process given by the resampling probability $\{p(t), t = 1, 2, \ldots, \}$ has a stationary distribution. This is a hard question and it is even difficult to empirically check this conjecture. The convergence of the voted rule is a more precise way to address this question. The graph below gives some intuition to the convergence problem. We apply boosting with the single-layer feed-forward neural network. The data used is like a horseshoe. The Bayes error for this data distribution is 0. This experiment is carried out in Splus 5.2, and nnet (Venables and Ripley [1997]) is used for the neural network. We carry out the boosting 10 times. For each run and each iteration $k$, a voted rule is obtained by voting the rules up to this iteration. Each curve is the decision boundary for a decision rule. We took iteration= $1, 10, 20, 50$, and we have
10 decision boundaries for each of these iterations. These decision boundaries are shown in Figure 1.1.

In the beginning, there is lots of variation among the 10 different runs. However, as the learning procedure moves on, there is less and less variation among the rules. Although it is not clear that the decision boundaries will be the same in the end, it is clear that they are becoming more and more consistent with each other. This consistency phenomenon is also an important indication that the variation among the decision rules decreases as the learning process moves on.

1.3.3 Empirical Interpretation: Margin and Diversity

Schapire et al. [1998] defined the margin of a point as the difference between the weight assigned to the correct label and the maximal weight assigned to any single incorrect label. Suppose we carry out boosting for $N$ steps, and obtain $N$ decision rules $r_1, r_2, \ldots, r_N$, with the weights $w_1, w_2, \ldots, w_N$ respectively, then at the training point $(x, y)$, the margin is:

\begin{equation}
\text{margin}(x) = \sum_{i: r_i(x) = y} w_i - \max_{i \neq y} \sum_{j: r_j(x) = i} w_j.
\end{equation}

Notice that if the margin is positive, then the voted rule predicts the observed class; otherwise, the voted rule makes an incorrect prediction. A bigger positive margin also indicates that the members of the ensemble of rules are consistent with each other. On the other hand, a close-to-zero margin means the ensemble is not sure about its prediction. Schapire et al. [1998] showed that Adaboost tends to increase the minimum margin, and to decrease the maximum margin, i.e., it tends to distribute the correct votes uniformly for all training points.

Dietterich [1998b] and Ho [1998] observed that in order to make the ensemble method work, the decision rules in the ensemble must be diverse. Given two decision rules $r_1, r_2$, Dietterich defined $\kappa$-statistics, which are similar to $\chi^2$ statistics. His observation was that Adaboost tends to produce ensembles that have large diversity, while bagging tends to
generate decision rules of similar diversity. Ho defined diversity as the disagreement of two
rules, i.e., diversity of $r_1, r_2 = E_X I(r_1(X) \neq r_2(X))$. Ho used a c4.5 tree that is fully split so
that it classifies every training example correctly. It was observed that if the base learner
is very flexible as was the case in Ho, bagging and Adaboost are very similar, and there is
no significant difference between the diversities.

Breiman [1999b] extended the diversity idea to correlation between two random vari­
ables, and obtained a bound on the generalization error in terms of the diversity and strength
of the ensemble.

1.4 Random iid Ensembles and Perfect
Random Trees

1.4.1 Independent Ensemble Methods

An ensemble method needs a base classifier/learner. It repeatedly applies the base
classifier to perturbed training data. The different rules obtained in different runs are then
combined by weighted or unweighted voting. The way of generating different rules can be
different. Dietterich ([1998a] (discussion paper on Breiman [1998a]) listed four typical ways.

1. Manipulating the training set.
2. Manipulating the input features.
3. Manipulating the output targets.
4. Randomizing the fitting procedure.

The first strategy carries the most statistical flavor. Breiman’s bagging is of this type. Given training data of size $n$, a bootstrap sample consists of $n$ data points, each drawn from the original data with probability $\frac{1}{n}$. For given training data, we can generate many bootstrap samples. We can build one rule for each of these bootstrap samples, thus giving an ensemble of different rules. Breiman points out that a critical factor for bagging to work is the instability of the classification method. Dietterich [1998b] and Breiman [1999a] further point out that it is the amount of diversity of the collection of rules that influences how much bagging may improve the performance of a base classifier.

The second strategy is discussed in Ho [1998] and Amit and Geman [1997]. Ho’s method builds an ensemble of rules by considering all possible subsets of the variables (features). For each subset, a fully-split C4.5 tree is built on the whole data with chosen features. The $2^p - 1$ trees comprise the ensemble. The empirical experiments show that this method works well for many big datasets. One limitation of Ho’s strategy is that the number of trees is limited to $2^p - 1$.

Amit and Geman apply ensemble methods to optical character recognition and face detection. They propose generating a large set of new features from the original features, and building trees using different random subsets of these new features. Taking advantage of this large set of new features, Amit and Geman only build very shallow trees, so their ensembles are usually very large.

The third strategy is explored by error-correcting output coding (Dietterich and Bakiri [1995]). ECOC deals with questions of multiple classes. Each time, they cleverly separate the class labels into two groups, relabeling one group as class “0”, the other as class “1”.
Then the base classifier is applied to the relabeled training data. Different relabeling thus provides an ensemble of different rules.

Breiman [1998b] also experiments with perturbing the output of a classification problem. Output flipping tries to change some proportion (flipping rate) of output to other classes. The ensembles are obtained by applying the base classifier on different perturbed training data. His experiments show that this strategy also improves the prediction accuracy of a base learner with the improvement being comparable to that of bagging.

The last strategy is much more complicated. There can be lots of ways of adding randomness in the fitting procedures (Dietterich [1998b] has more discussion on this). One method Dietterich [1998b] experimented with is adding randomness in c4.5 tree methods. Instead of the usual way of splitting at the feature that gives largest reduction in Gini impurity, it randomly chooses one among the 20 best splits. It is shown that voting these different c4.5 trees gives results comparable to bagging c4.5 trees.

One common characteristic of the ensemble methods is that they use a common base classifier. The building of the base learner typically requires an optimization method. PERT (Cutler [1999]) and Stochastic Discrimination (Kleinberg [1990], Kleinberg and Ho [1993], Kleinberg [1996]) adopt a different strategy. For given training data, both methods sample random decision rules from a given space of decision rules. Then these rules are combined to give the final decision rule. A theoretical study on PERT is one major topic of this dissertation. We introduce the definition below.

1.4.2 Perfect Random Tree

A perfect random tree (PERT) is a voted decision rule for which each tree is split until all the nodes are pure. At each node, the split is purely random so there is no optimization involved.

Given the training data \( T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \), with each \( x_i \) a d-dimensional vector, one PERT is built as follows:
Initialize the stack so it contains one node comprising $T$ itself. Repeat until there are no more nodes in the stack.

1. If the stack is empty, stop. Otherwise take a node from the stack.
2. If all $y_i = y_1$, the node is a terminal node. $\hat{y} = y_1$ for this node. Go to step 1.
3. Randomly pick two sample indices $i$ and $j \in \{1, 2, \ldots, n\}$ until $y_i \neq y_j$.
4. Randomly pick an attribute index $l \in \{1, 2, \ldots, d\}$.
5. Randomly pick a number $s$ between $x_{il}$ and $x_{jl}$.
6. Divide the parent node into two child nodes,
   
   left_node = all examples with $x_{il} > s$, put this node on the stack.

   right_node = all examples with $x_{il} \leq s$, put this node on the stack.
7. Go to step 1.

The above procedure is repeated $N$ times and the random trees are voted to give the final decision rule.

One characteristic of PERT is that it classifies every training example correctly. This is like the NN method and Adaboost. We call a method that classifies all training examples correctly an "oversensitive" method. For convenience, we give the following definition.

**Definition 1.4.1 (Oversensitive classification method)**

A classification method $M$ is called oversensitive if $r(., T, M)$ correctly classifies every point in $T$.

Oversensitive methods will not work well for the regression situation. For classification, ensemble methods that are oversensitive work very well. PERT and boosted c4.5 are two such methods. We will discuss their properties in Chapter 4 and Chapter 5.

**1.5 Summary and Our Work**

As reviewed in the previous section, there are many random methods using iid ensembles. However, the theoretical studies on how they work are rare. Breiman [1999b] focused on how strength and diversity may provided a bound on the generalization error. Kleinberg
[1998] studied how the spread and richness determines the generalization error. This dissertation focuses on the convergence of the voting procedure. It shows how voting works in multiclass situations and clarifies the confusion raised by the bias and variance definitions for classification.

This dissertation has seeds in Breiman [1998a] and Friedman [1997]. Breiman used \( Q(j|x) \) to represent the expected percentage of decision rules that predict \( x \) as class "\( j \)". This dissertation explicitly defines the space of decision rules, which are generated by applying the base method \( M \) to all possible training sets of fixed size. The \( Q(j|x) \) (our \( q(j|x) \)) is a measure on the space of decision rules. Therefore \( M \) can be viewed as a mapping from the data space \((\mathcal{X}, p)\) to the rule space \((\mathcal{R}, q)\) (details in Chapter 2). Such a triplet \( \mathcal{L} = \{(\mathcal{X}, p), M, (\mathcal{R}, q)\} \) is called a learning system. Bias, variance, diversity and instability are all characterized within the learning system. Also, Friedman’s idea on how bias and variance of regression estimates combine to influence the classification error is automatically extended to the multiclass situation.

In Chapter 3, voting processes are defined, and two basic theorems, the bias consistency theorem and convergence theorem, are proven. The two results give theoretical justification of ensemble methods discussed in the previous section. They make the voting behavior crystal clear. The theorems also give a simple formula that tells when voting will improve a base learner. A new kind of consistency which we call weak consistency is proposed and discussed. The voting behavior for NN and some random methods are also discussed.

Chapter 4 is devoted to discussing the properties of PERT classifiers. A recursive equation for \( Q(j|x) \) of PERT is derived. It is also shown that PERT is not strongly consistent, and the reason why PERT works well lies in the squeezing effect, which is indicated by the recursive equation. Two variants of PERT are invented to deal with more complicated questions.

In Chapter 5, the robustness properties of boosted CART are discussed. A new perspective is obtained by viewing boosted CART as an oversensitive method. Simulations
are carried out to verify this explanation. From the framework, a new quantity (Bayes deviance) is also obtained to measure the efficiency of a classification method. Simulations are carried out to compare the efficiency of different classification methods.
CHAPTER 2
A LEARNING SYSTEM

In this chapter, a framework is built that provides a clearer idea about classification bias and variance. Some concepts such as instability, which is heuristically discussed by Breiman [1996a], are characterized. A posterior measure $Q$ is introduced in section 2.1 to describe ensemble methods. In section 2.2, a learning system is defined and bias, variance and instability are then defined in this framework. It is shown that the same instability defined here is actually the expected diversity discussed by Dietterich [1998b] and Ho [1998]. A bound on generalization error is also obtained. In section 2.3, the relationship between the misclassification error and classification bias and variance is discussed. Friedman’s idea on how bias and variance for regression combine to influence the classification error is extended into the multiclass situation.

2.1 A Posterior Measure on Rule Spaces

An ensemble method such as those discussed in section 1.4 generates an ensemble of decision rules. Typically a random vector is involved in the procedure of building the rule, and usually the random vectors for building different rules are independent. This implies that the decision rules obtained will also be independent. Notice for Adaboost, the decision rules are generated sequentially, and each decision rule is dependent on the previous rule. Therefore, Adaboost does not fit in this framework. However, it can be accommodated in the framework given in the next section.

Assume a data structure $D$ and an ensemble method $M$ are given. Let $k$ be the total number of classes, and $T$ be training data of size $n$, with each observation drawn iid from $D$. Denote the random vector involved in the rule-building procedure by $\theta$, and its distribution by $F_\theta$. Denote by $r(\cdot, \theta, T, M)$ the decision rule obtained by applying the method $M$ to
training data $T$ using the random vector $\theta$. Then we have the following result.

**Theorem 2.1.1 (Independence of rules)**

Assume $\theta_1, \theta_2, \ldots, \theta_n$ are iid $F_\theta$. Let $M$ be a random method which involves random vectors. Denote by $r(., \theta_i, T, M)$ the decision rule built by using the vector $\theta_i$. Assume that for each point $x$, $r(x, ., T, M)$ is measurable. Then the decision rules $r(., \theta_i, T, M)$ are independent and have identical distributions at every point $x$ in the feature space.

**Proof:** The proof is direct. The independence can be obtained by Theorem 3.3.3 of Chung [1974].

Now let $\mathcal{R}_T$ be the set of all possible decision rules obtained by using different $\theta$, i.e.,

$$\mathcal{R}_T = \{r(., \theta, T, M)|\theta \sim F_\theta\}.$$

Notice the measure on $\theta$ induces a measure on $\mathcal{R}_T$. But it is usually very complicated because a decision rule is a complicated function on the data space $\mathcal{X}$. However, discussion on the behavior of the decision rule can be made at each point. $Q(i, x|T)$ is defined as follows:

$$Q(i, x|T) = P\{r(x, \theta, T, M) = i\}$$

$$Q(i, x|T) = P_\theta\{\theta|r(x, \theta, T, M) = i\}.\tag{2.1}$$

With this definition of $Q(i, x|T)$, it is easy to see that at each point $x$, $\{Q(i, x|T), i = 0, 1, \ldots, k - 1\}$ is a measure on $\mathcal{R}$. Each random method can be viewed as a mapping from training data $T$ to $k$ surfaces $Q(0, x|T), Q(1, x|T), \ldots, Q(k - 1, x|T)$, each representing the posterior probability for one class. Voting infinitely many iid rules is equivalent to finding $\arg\max_i Q(i, x|T)$. When only finite numbers of decision rules are voted, it is like estimating $\arg\max_i Q(i, x|T)$ by $\arg\max_i Q(i, x|T)$. There might be some discrepancy between the sample version and its theoretical counterpart. The distribution of $\hat{Q}(i, x|T)$ will be discussed later on.
Figure 2.1. The training data $\rightarrow$ Q surface $\rightarrow$ decision rule

For the 2-class case, it will be enough to discuss $Q(1, x|T)$, as $Q(0, x|T) = 1 - Q(1, x|T)$. Figure 2.1 gives a set of training data, the Q surface plot, and finally the decision rule. The training data are 2-dimensional, 2-class data with seven sample points as plotted in the left graph. The classification method we use is the PERT method introduced in section 1.4. For PERT, we can actually calculate the theoretical $Q(1|x)$ using a recursive equation (details in Chapter 4). The Q surface is given in the top graph. The classification rule is: predict $x$ to be from class “1” if $Q(1, x|T) > 0.5$; otherwise predict $x$ to be from class “0”. The right graph gives the decision boundary obtained through this Q surface.

Notice on most occasions, it is impossible to get a theoretical Q surface. We usually generate a finite ensemble of decision rules $r_1, r_2, \ldots, r_N$. At each new point $x$, we can calculate the relative frequency of the decision rules that predict $x$ to be of class $i$. This is an estimate of $Q(i, x|T)$.

\[
\hat{Q}(i, x|T) = \frac{1}{N} \# \{ j : r_j(x) = i \}.
\]

At point $x$, the class of $x$ is predicted as the $i$ which has the largest $\hat{Q}(i, x|T)$.

For the PERT method, it is possible to obtain the theoretical Q values for a training
set of any size. We use a simpler example to explain how to calculate the $Q$ surface.

**Example 2.1.2 (Simple $Q$-surface for PERT)**

This is a 2-class, 2-dimensional problem with training set consisting of 2 points $T = \{(x_1, y_1), (x_2, y_2)\}$, with $x_1 = (a_1, b_1), y_1 = 0; x_2 = (a_2, b_2), y_2 = 1, a_1 \neq a_2, b_1 \neq b_2$.

In order to build one random tree, either a vertical split or a horizontal split is chosen. PERT stops after one step as either choice will separate the two points. So the space of the decision boundaries is simple. It is the set of all vertical lines $x = x_{cut}$ with $x_{cut}$ between $a_1$ and $a_2$ and horizontal lines $y = y_{cut}$ with $y_{cut}$ between $b_1$ and $b_2$. The random vector involved here can be represented by one random vector $U = \{u_1, u_2\}$, where $u_1$ is used to determine whether to do a horizontal split or a vertical split, and $u_2$ is used to determine where to put the split line.

Now at any point $x = (a, b)$, some rules will predict $x$ to be of class “0”, others will predict $x$ to be of class “1”. Whether $x$ is predicted as class “0” or “1” depends on where $x$ is located. Following the way PERT is carried out, the probability $Q(1, x|T)$ of $x$ being predicted as “1” can be calculated by:

\begin{equation}
Q(1, x|T) = \frac{1}{2} \delta \left( \frac{a - a_1}{a_2 - a_1} \right) + \frac{1}{2} \delta \left( \frac{b - b_1}{b_2 - b_1} \right),
\end{equation}

where

\begin{equation}
\delta(u) = \begin{cases} 
0, & u \leq 0 \\
 u, & 0 < u < 1 \\
 1, & u \geq 1.
\end{cases}
\end{equation}

Notice that for a deterministic method, $Q(i, x|T)$ is either 0 or 1. Therefore, $Q(i, x|T)$ is usually discontinuous. In contrast, most of the random methods such as PERT and Random Hyperplane generate (introduced in section 3.4) a continuous $Q$ surface. Whether this makes a difference is an interesting question, and will be discussed in the future.

### 2.2 The Learning System

Suppose a data structure $D$ is given. In order to see how a classification method $M$ works on $D$, a sample is drawn from $D$, the method $M$ is applied to it, and then an independent
test set is used to evaluate its performance. Since a larger set of training data usually results in a better decision rule, the training size is usually fixed in the beginning, say at \( n \). Let \( T \) be a random sample of training data of size \( n \). The misclassification error of \( M \) is

\[
e_T = \mathbb{E}_D I(r(X, T, M) \neq Y).
\]

Here \( \mathbb{E}_D \) refers to the expectation with respect to the data structure \( D \), i.e., it refers to \( X \) and \( Y \). Notice \( e_T \) can be considered to be a random variable. If \( d \) is the dimension of the feature space, then \( T \) can be seen as a random vector of size \( n \times (d + 1) \). Let \( F_T \) be its cumulative density function, and \( p(dT) = P(T \in dT) \) be its “infinitesimal” change, in the same sense \( p(dx) \) is used.

Now let \( E_M = E_T(e_T) \), the mean generalization error of \( M \). It is desirable that both \( E_M \) and the variance of \( e_T \) should be small. A small mean generalization error means that method \( M \) works generally well, while a small variance of \( e_T \) indicates that a random \( e_T \) won’t be much different from the expected generalization error. This means the classification method is stable in generating similarly accurate decision rules.

Our goal in this section is to describe everything in terms of the posterior measure on the space of decision rules. In this section \( M \) is assumed to be a deterministic method or a voted random method (i.e., the randomness has already been eliminated by voting). We define the space \( \mathcal{R} \) as:

\[
\mathcal{R} = \{r(. , T, M) | T \overset{\text{iid}}{\sim} F_T \}.
\]

Now the posterior measure \( q(i|x) \) is defined (Breiman [1998a] has already proposed this) by:

\[
q(i|x) = E_T I(r(x, T, M) = i).
\]

Imagine that many independent training sets \( T_1, T_2, \ldots, T_N \) are drawn, and \( M \) is applied to them, then \( q(i|x) \) represents the expected relative frequency that these rules predict \( x \) as class “i”.
For the 2-class situation, if $q(1|x)$ is close to 1, then lots of rules will classify $x$ as class “1”. So the variation among the rules will be small. In another way, it means the set of the rules will be quite stable in predicting the class for $x$. If $q(1|x)$ is close to 0, then lots of rules will predict class “0” at $x$, and again the variation among the rules will not be large. When $q(1|x)$ is close to 0.5, then the prediction will be quite varied. In another way, this means the rules will not be stable in predicting the class of $x$, so the class of $x$ will be very hard for the rules to predict. One way to measure this variation or hardness to predict is to use the margin function $m(x) = |q(1|x) - q(0|x)|$ (general definition is given in Definition 2.2.4).

The example below aims to give some intuition about the prediction variation. The tree classifier on the continuous XOR data used in Leisch and Hornik [1997]. XOR is a standard benchmark question for machine learning methods. It is defined as follows.

**Example 2.2.1 (XOR distribution)** The data are distributed uniformly in $[-1,1] \times [-1,1]$. The data in the first and third quadrants are assigned class 1; others are assigned class “0”.

In order to estimate the $q(1|x)$, 1000 training datasets are drawn, each consisting of 300 points. For each training set, the tree method is applied to give a decision rule. All trees are evaluated on a $41 \times 41$ grid of points. The $q(1|x)$ is estimated by the relative frequency of rules predicting $x$ as of class “1”. $m(x)$ is also calculated. Figure 2.2 gives the surface plots of both $q(1|x)$ and $m(x)$.

From Figure 2.2, we can see that the tree methods are fairly stable for XOR data, with most of the $q(1|x)$ either close to 0 or close to 1. Thus these points are easy for tree methods. For those points around the boundary, i.e., the axes, $q(1|x)$ is close to 0.5. They are hard points, where the prediction varies as the training sample differs.

### 2.2.1 $q$ Surfaces and Learning System

Assume the data distribution is known. At each point $x$, the posterior probability $p = \{p(i|x), i = 0, 1, \ldots, k-1\}$ can be calculated. For any classification method $M$, together
with the data structure $\mathcal{D}$, it produces a vector measure $q = \{q(i|x), i = 0, 1, \ldots, k - 1\}$. Therefore, a machine learning method $M$ is a mapping from a data space $(\mathcal{D}, p)$ to a rule space $(\mathcal{R}, q)$.

(2.9) \hspace{1cm} (\mathcal{D}, p) \xrightarrow{M} (\mathcal{R}, q).

and

$$
\begin{pmatrix}
    p_0 \\
    p_1 \\
    p_2 \\
    \vdots \\
    p_{k-1}
\end{pmatrix}
\xrightarrow{M}
\begin{pmatrix}
    q_0 \\
    q_1 \\
    q_2 \\
    \vdots \\
    q_{k-1}
\end{pmatrix}
$$

It is easy to see that the measure $q$, together with $p$ determines properties of the learning system such as prediction accuracy and variability of the collection of rules $\mathcal{R}$. We define these quantities later on.

**Definition 2.2.1 (Learning system)**

A learning system is a triplet consisting of a data structure, a learning method and the space of decision rules. It is denoted by $\mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\}$. 
When a learning system $\mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\}$ is given, and $C_B(x)$ is the Bayes classifier as defined in Chapter 1, the bias set can be defined as follows.

**Definition 2.2.2** (Bias set and unbiased set, Breiman [1998a])

$$B = \{ x | \text{argmax}_i q(i|x) \neq \text{argmax}_i p(i|x) \} = \{ x | \text{argmax}_i q(i|x) \neq C_B(x) \}.$$

The complement of $B$ is called the Unbiased set, and is denoted by $U$.

**Definition 2.2.3** (Pointwise bias and variance)

\begin{align*}
\text{bias}(x) &= I(\text{argmax}_i q(i|x) \neq C_B(x)) \\
\text{var}(x) &= \sum_{i=0}^{k-1} q(i|x)(1 - q(i|x)).
\end{align*}

**Remark 2.2.2** As reviewed in the Introduction, there are already several definitions of bias and variance for classification. The definition of bias given above is the same as Kong and Dietterich [1995]. The definition of variance is developed independently by the current author. It is more general than the one derived by Breiman [1998a] in that it is not related to the data distribution. We list the properties of our variance definition as follows:

1. It is natural. It represents the total variance of a multinomial distribution, as indicated by Proposition 1 below.

2. $0 \leq \text{var}(x) \leq 1$ and $\text{var}(x) = 0$ iff $q(i|x) = 1$ for some $i$.

3. $\text{var}(x)$ is maximized when all $q(0|x), q(1|x), \ldots, q(k-1|x)$ are the same.

4. $\text{var}(x)$ represents the variations among decision rules. It is directly related to the convergence of the voting process (Chapter 3).

One disadvantage is that our definition does not admit an additive decomposition. However, our framework provides an easy and accurate way to understand how voting methods
work. The multinomial nature of classification methods actually shows that the systematic error and the random error are not independent, so any additive decomposition is misleading in explaining how voting works.

Consider the learning system $\mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\}$. $\mathcal{R}$ are $q$ are defined as in formulas 2.7 and 2.8. $n$ rules $\{r_1, r_2, \ldots, r_n\}$ are drawn independently from $\mathcal{R}$. Define $Z(x) = (Z_0(x), Z_1(x), \ldots, Z_{k-1}(x))^T$, with $Z_i(x) = \#\{r_j | r_j(x) = i, j = 1, 2, \ldots, n\}, i = 0, 1, \ldots, k-1$. Then the following results are true.

**Proposition 2.2.3 (Distributions of votes vector)**

Let $Z(x)$ be as defined above. Then $Z(x)$ follows a $k$-class Multinomial$(n, q)$ distribution, where $q = (q(0|x), q(1|x), \ldots, q(k-1|x))^T$. Specifically, $EZ(x) = nq(x)$, and $Var(Z(x)) = (\sigma_{ij})_{k \times k}$, with $\sigma_{ii} = nq(i|x)(1 - q(i|x))$, and $\sigma_{ij} = -nq(i|x)q(j|x)$, for any $i \neq j$.

For a multinomial distribution, a natural definition of total variance is the sum of the diagonal elements of the variance-covariance matrix $\Sigma$. It will be $\sum_{i=0}^{k-1} q(i|x)(1 - q(i|x))$, which is our variance for the collection of decision rules. The covariance between each of the two different components is negative. The proof can be obtained by considering each $Z_i(x)$ as the sum of iid Bernoulli distributions.

Notice when the variance at point $x$ is big, then the learning system is not not confident about its prediction at this point. Therefore, the variance can be considered to be a hardness-to-predict measure. We give another quantity that may also tell how hard the point $x$ is for the learning system.

**Definition 2.2.4 (Decision margin)**

Given the learning system $\mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\}$, the margin $m(x)$ of the system is defined as the difference between the maximum of $q(i|x)$ and the second largest $q(i|x)$:

$$m(x) = \max_i q(i|x) - \max_{j \neq i} q(j|x),$$

with $i_q = \arg \max_i q(i|x)$. 


Remark 2.2.4 For a 2-class question, the relationship between the variance is: \( \text{var}(x) = \frac{1}{4}(1 - m^2(x)) \). In this case, a big margin indicates a small variance. For the general multiclass case, when \( m(x) \) is large, the largest \( q(i|x) \) is away from other \( q(i|x) \), this usually indicates a small variance. The decision margin plays an important role in the convergence theorem discussed in the next chapter.

When the learning system is given, we can obtain the expected classification error at each point in terms of \( p \) and \( q \):

\[
P_{Y,T}(r(x,T,M) \neq Y) = 1 - P_{Y,T}(r(x,T,M) = Y) = 1 - \sum_{i=0}^{k-1} P_T(r(x,T,M) = i) P(Y = i|x)
= 1 - \sum_{i=0}^{k-1} q(i|x)p(i|x).
\]

The total expected error is the integral of the pointwise error, and thus we obtain our definition for generalization error of a learning system.

Definition 2.2.5 (Generalization error of a learning system)

Given a learning system \( \mathcal{L} = \{(D,p), M, (R,q)\} \), we define the pointwise error \( e(x) \) and the overall generalization error \( E_{\mathcal{L}} \) as follows:

\[
e(x) = P_{Y,T}(r(x,T,M) \neq Y) = 1 - \sum_{i=0}^{k-1} q(i|x)p(i|x)
\]

(2.13)

\[
E_{\mathcal{L}} = \int_X e(x)p_X(dx) = 1 - \int_X \sum_{i=0}^{k-1} q(i|x)p(i|x)P_X(dx).
\]

(2.14)

Notice the above definition of \( E_{\mathcal{L}} \) is the same as \( E_M = E_T(e_T) \), the expectation of \( e_T \) in 2.6. Both are double integrals, with a difference in integration order. The Tonelli Theorem (Royden [1988]) guarantees the exchangeability of the integration order.

\[
E_T(E_{X,Y}P(r(X,T,M) \neq Y)) = E_X(E_{Y,T}P(r(X,T,M) \neq Y))
= E_X e(X)
= E_{\mathcal{L}}.
\]
Remark 2.2.5 If we rewrite

\begin{equation}
(2.15) \quad e(x) = [1 - p(C_B(x)|x)] + [p(C_B(x)|x) - \sum_{i=0}^{k-1} q(i|x)p(i|x)],
\end{equation}

then the first part is the error suffered by the Bayes rule, the second part is the extra error suffered because we are not using the Bayes rule. Breiman defined the bias and variance on the overall distribution by integrating and partitioning the second part.

\[ E_e = \int_X (1 - p(C_B(x)))P_X(dx) \]

\[ + \int_U (p(C_B(x)|x) - \sum_{i=0}^{k-1} q(i|x)p(i|x))P_X(dx) \]

\[ + \int_B (p(C_B(x)|x) - \sum_{i=0}^{k-1} q(i|x)p(i|x))P_X(dx). \]

From formula 2.15, we have the following observations:

1. The minimum of \( P_{Y:T}(r(x,T,M) \neq Y) \) is \( 1 - \max_i p(i|x) = 1 - p(C_B(x)|x) \).

2. The error is small if \( q(C_B(x)|x) \) is large and \( q(i|x) \) is small for all \( i \neq C_B(x) \).

3. When voting or arcing is used, it can be viewed as a way to change \( q(i|x) \) towards a better direction.

2.2.2 Instability and Diversity

Now we continue to characterize the instability idea.

Definition 2.2.6 (Instability of a learning system)

Given the learning system \( \mathcal{L} = \{(\mathcal{D},p), M, (\mathcal{R},q)\} \), the instability of the learning system, also called total variance is defined as:

\begin{equation}
(2.16) \quad \text{Ins}(\mathcal{L}) = \int_X \text{var}(x)P_X(dx).
\end{equation}

The instability definition here quantifies the instability idea of Breiman [1996a]. Breiman described unstable classifiers as those for which small changes in the training data will introduce drastic changes in the resulting decision rules. Breiman did not give a formal definition. The definition here is possible owing to the \( q \) measure and our definition of
pointwise variance. It shown that our $Ins(\mathcal{L})$ is actually the expected disagreement between two randomly chosen decision rules. Thus it characterizes the diversity (Dietterich [1998b]) and dissimilarity measure (Ho [1998]). Both of these authors showed that diversity is important in making ensemble methods work. This in turn verifies Breiman’s claim that “the vital element is the instability of the prediction method” (Breiman [1996a]).

Here is the proof: Let $r_1, r_2$ be two independent decision rules drawn from $\mathcal{R}$, then the expected disagreement between $r_1$ and $r_2$ is

$$
E_X I(r_1(X) \neq r_2(X)) = 1 - E_X I(r_1(X) = r_2(X))
$$

$$
= 1 - \int_X I(r_1(x) = r_2(x))p(dx)
$$

$$
= 1 - \int_X \sum_{i=0}^{k-1} I(r_1(x) = i, r_2(x) = i)p(dx)
$$

$$
= 1 - \int_X \sum_{i=0}^{k-1} I(r_1(x) = i) \times I(r_2(x) = i)p(dx).
$$

Taking expectations with respect to $r_1, r_2$, we have:

$$
E_{r_1, r_2} E_X I(r_1(X) \neq r_2(X)) = 1 - \int_X \sum_{i=0}^{k-1} q(i|x)^2p(dx)
$$

$$
= \int_X \sum_{i=0}^{k-1} q(i|x)(1 - q(i|x))p(dx)
$$

$$
= Ins(\mathcal{L}).
$$

Intuitively, we would expect that when both the size of the bias set and the variance are small, the generalization error will be small. We obtain such a bound on the generalization error in terms of the size of the bias set, the instability of the system.

**Theorem 2.2.6 (Bound on the generalization error)**

Assume a learning system $\mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\}$ is given. Let $k$ be the number of classes. Let $s_B = P_X(B)$ be the size of the bias set, and $V = Ins(\mathcal{L})$. Let $A = \sup_x DM(x)$, where $DM(x) = \max_i (p(C_B(x)|x) - p(i|x))$, Then the generalization error $E_\mathcal{L}$ can be bounded:

$$
E_\mathcal{L} \leq E_B + A(kV + s_B).
$$
Proof: For convenience, we denote $C_B(x)$ by $i_B$. The Decomposition formula 2.27 gives:

$$e(x) = P_Y(C_B(x) \neq Y) + \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q(i|x)$$

Integrating both sides, we have:

$$E_C \leq E_B + \int_X \sum_{i \neq i_B} (p(i_B|x) - p(i|x))q(i|x)P_X(dx).$$

We partition the second term on right hand side into two parts:

(2.18) \[\int_B \sum_{i \neq i_B} (p(i_B|x) - p(i|x))q(i|x)P_X(dx)\]

and

(2.19) \[\int_U \sum_{i \neq i_B} (p(i_B|x) - p(i|x))q(i|x)P_X(dx).\]

Now 2.18 is controlled by:

$$\int_B \sum_{i \neq i_B} Aq(i|x)P_X(dx) = A \int_B (1 - q(i_B|x))P_X(dx) \leq A s_B.$$

2.19 is controlled by:

$$\int_U A(1 - q(i_B|x))P_X(dx) \leq A \int_U kvar(x)P_X(dx) = A kV.$$

The second to last equation holds because

$$var(x) = \sum_i q(i|x)(1 - q(i|x)) > q(i_B|x)(1 - q(i_B|x))$$

Since $x$ is in the Unbiased Set, $q(i_B|x)$ is the largest among all $q(i|x)$; therefore, $q(i_B|x) > \frac{1}{k}$.

This means:

$$var(x) > \frac{1}{k}(1 - q(i_B|x)),$$

or

$$(1 - q(i_B|x)) < kvar(x).$$

The two inequalities render the needed bound. \qed
Remark 2.2.7 The bound obtained here is not tight. However, the goal here is simply to show that the misclassification error can be bounded by the size of the Bias set and the variance. The formula says if the size of the bias set and the total variance goes to 0, then the misclassification error will approach the Bayes error.

Breiman [1996a] concluded that bagging can only improve unstable classifiers. He listed CART and neural networks as unstable classifiers, and k-NN and LDA as stable classifiers. However, Zhao and Cutler [1999] showed that bagging LDA can improve the prediction accuracy for XOR data. Actually LDA is not stable for XOR data, and that is why bagging LDA can still work for a revised version of XOR data. Thus we argue that whether a classifier is stable or not is data related, i.e., it is more of a learning system characteristic. It is more appropriate to say that a learning system is stable or not. Here is another example that supports this argument.

Example 2.2.8 \( L=(D=2\text{-class}, \pi_0 = \pi_1, M=k\text{-NN}, k=N, R=\text{general}) \)

Since \( k=N \), at each training example, k-NN predicts all data points as of the same class, the majority class in the current training data, i.e., \( Q(i|x|T) = 0 \) or \( 1 \). Now since \( \pi_0 = \pi_1 \), the chance is the same that the majority class is 0 or 1. This means again \( q(i|x) = 0.5 \), a very unstable case.

Friedman [1997] discussed the performance of k-NN method. He applied the k-NN method with different \( k \) values to the same training data \( T \). He got a bowl-shaped error curve, which means the optimal \( k \) value for k-NN methods is in the middle. A very large \( k \) will make the k-NN unstable and therefore make it impossible to work well. We will further discuss consistency and instability of the k-NN method in Chapter 4.

2.3 The Effects of Bias and Variance on Generalization Error

Friedman [1997] provided a thoughtful analysis of classification error. For the 2-class situation, he considered the decision rules obtained through the plug-in methods (Definition
It is shown that the bias and variance of \( \hat{p}(i|x) \) influence the classification error in a different way than for the regression case. It is not clear how his analysis can be carried over to the multiclass situation. The framework given here provides a way to extend Friedman's idea to the multiclass case.

For convenience, \( r(x) \) or \( r(x|T) \) is used to represent \( r(x,T,M) \), i.e., the decision rule obtained by applying \( M \) to the training data \( T \).

2.3.1 Bayes Deviance for Classification

A key formula in Friedman [1997] is Formula (5.1), which decomposes the misclassification error into the Bayes error and the remaining part. The formula is valid only for the 2-class case, and in our notation, it is:

\[
(2.20) \quad P_{Y,T}(r(x|T) \neq Y) = P_Y(C_B(x) \neq Y) + |1 - 2p(1|x)|P_T(r(x|T) \neq C_B(x))).
\]

From the above formula, it is easy to see that the misclassification error is a linearly increasing function of the part \( P_T(r(x|T) \neq C_B(x))) \). This probability measures the average amount of disagreement between the collection of rules and the Bayes rule. The smaller the disagreement is, the smaller the average prediction error will be. This quantity also plays an important role for the consistency analysis and the bias and variance relationship. We call it Bayes deviance and it will be formally defined below. One virtue of Bayes deviance is that it is a discrepancy from the deterministic Bayes rule. When we discuss the classification error, it includes variation in \( Y \) itself, whereas Bayes deviance does not.

Definition 2.3.1 (Bayes deviance)

Given the learning system \( \mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\} \), we define pointwise Bayes deviance \( d_B(x) \) and the overall Bayes deviance \( D_B(\mathcal{L}) \):

\[
(2.21) \quad d_B(x) = P_T(r(x) \neq C_B(x)) = 1 - q(C_B(x)|x), \]

\[
(2.22) \quad D_B(\mathcal{L}) = \int_X d_B(x)P_X(dx).
\]
In order to extend Friedman’s Formula (5.1) to the multiclass situation, we note that, for a fixed decision rule \( r \),

\[
(2.23) \quad P_Y(r(x) \neq Y) = (p(C_B(x)|x) - p(r(x)|x))I(r(x) \neq C_B(x)) + P_Y(C_B(x) \neq Y).
\]

Notice that for the two class situation, we have \((p(C_B(x)|x) - p(r(x)|x)) = |1 - 2p(1|x)|\). Therefore, formula 2.23 becomes:

\[
(2.24) \quad P_Y(r(x) \neq Y) = P_Y(C_B(x) \neq Y) + |1 - 2p(1|x)|I(r(x) \neq C_B(x))
\]

and 2.20 can be obtained by taking the expectation over the distribution of \( T \). Now for the multiclass case, \( p(C_B(x)|x) - p(r(x)|x) \) may take different values, so we have to partition the first term in 2.23 into different parts, i.e.,

\[
(2.25) \quad \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))I(r(x) = i).
\]

Notice the expectation of 2.25 over the distribution of \( T \) is

\[
(2.26) \quad \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q(i|x).
\]

Finally a counterpart of equation 2.20 is derived for the multiclass case by taking the expectation of equation 2.23 over \( T \):

\[
(2.27) \quad P_{Y,T}(r(x|T) \neq Y) = P_Y(C_B(x) \neq Y) + \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q(i|x).
\]

This formula is interesting in that it gives us the extra error we suffer as the result of not having the Bayes rule. At each point \( x \), for those rules that predict \( x \) as class “\( i \)”, the extra loss is \( p(C_B(x)|x) - p(i|x) \). In order to reduce the error, we will need to reduce \( q(i|x) \) for all \( i \neq C_B(x) \). And the only way to achieve the Bayes error is to have \( q(C_B(x)|x) = 1 \) and \( q(i|x) = 0 \) for all other classes.

### 2.3.2 Extension of Friedman’s Idea

Friedman’s idea is to investigate how the bias and variance of estimating \( p(i|x, T) \) may influence the classification error. As formula 2.20 indicates, the error is monotonically
increasing in the Bayes deviance, thus the analysis is actually on how the bias and variance of $\hat{p}(i|x, T)$ will influence this Bayes deviance.

Before we move on, it’s worthwhile that we look at an important feature of classification methods. We take the 2-class problem as an example.

For the 2-class situation, assume at some point $x$, $C_B(x) = 1$, i.e., $p(1|x) > 0.5$. If the estimated $\hat{p}(1|x) > 0.5$, then the “plug-in” method will predict $x$ as 1; otherwise, it will predict $x$ as 0. More importantly, the predicted class of the plug-in method will be 1 as long as $\hat{p}(1|x) > 0.5$. It makes no difference whether $\hat{p}(1|x) = 0.51$ or $\hat{p}(1|x) = 0.99$. In general, if $\hat{p}(1|x) - 0.5$ has the same sign as $p(1|x) - 0.5$, then the “plug-in” predictor will be the same as the Bayes predictor. Therefore, a classification method converts the density estimate into a discrete situation.

Assume again that the training data $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ are iid $F_T$. Then the estimated posterior density also forms a distribution $\hat{p}(1, x|T)$. Friedman considers how the bias and variance of $\hat{p}(1, x|T)$ influence the classification error. Let $\bar{p}(1|x) = E_T \hat{p}(1, x|T) = \int \hat{p}(1, x|T)p(dT)$, and $\text{var}(\hat{p}(1|x)) = E_T(\hat{p}(1, x|T))^2 - (\bar{p}(1|x))^2$. Then Friedman’s boundary bias is defined as:

\begin{equation}
(2.28) \quad b(p(1|x), \bar{p}(1|x)) = \text{sign}(0.5 - p(1|x)) \times (\bar{p}(1|x) - 0.5)
\end{equation}

where $\text{sign}(x) = 1$, if $x > 0$ and $\text{sign}(x) = -1$, if $x < 0$. For simplicity, we rewrite the equation as:

\begin{equation}
(2.29) \quad b(p, \bar{p}) = \text{sign}(0.5 - p) \times (\bar{p} - 0.5).
\end{equation}

Under the assumption of symmetry in the distribution of $\hat{p}(1, x|T)$, Friedman concluded that a point has a negative boundary bias if and only if $x$ is in the Unbiased set. Furthermore, Friedman assumed the asymptotic normality of $\hat{p}(1, x|T)$. Then he obtained the following distribution for $\hat{p}$

\begin{equation}
(2.30) \quad \text{pdf}(\hat{p}) = \frac{1}{\sqrt{2\pi \text{var}(\hat{p})}} exp \left(- \frac{1}{2} \frac{(\hat{p} - \bar{p})^2}{\text{var}(\hat{p})}\right).
\end{equation}

Notice here we suppress “$x$” from each probability for ease of reading.
Then the discrepancy between the Bayes rule and an individual rule is

\begin{align}
(2.31) \quad d_B(x) &= P_N(r(x) \neq C_B(x)) = 1 - \Phi \left( \text{sign}(p - 0.5) \frac{(p - 0.5)}{\sqrt{\text{var}(p)}} \right) \\
(2.32) &\quad = 1 - \Phi \left( -\frac{b(p, \bar{p})}{\sqrt{\text{var}(\bar{p})}} \right)
\end{align}

where $\Phi(x)$ is the cdf of the standard normal distribution. For the case of $p(1|x) > 0.5$, $d_B(x)$ is actually:

\begin{equation}
(2.33) \quad d_B(x) = P_{\bar{p}}(\bar{p} > 0.5)
\end{equation}

Assume that $\bar{p} - 0.5$ is fixed. Then it is easy to see the following:

1. If the boundary bias is negative, then $\bar{p}$ is on the same side of 0.5 with $p(1|x)$. Therefore the smaller the variance is, the smaller $d_B(x)$ will be, and the less error $r(x)$ will suffer (left-hand side of Figure 2.3).

2. If the boundary bias is positive, then $\bar{p}$ is on the different side of 0.5 with $p(1|x)$. A smaller variance will make $d_B(x)$ bigger, thus increasing the misclassification error (right-hand side of Figure 2.3).
The same conclusions can be obtained in our framework through the relationship between \(q(i|x)\) and \(\hat{p}(i|x)\):

\[
q(1|x) = P_T(\hat{p}(1|x) > 0.5).
\]

Now for the multiclass situation, the corresponding relation is

\[
q(i|x) = P_T(\hat{p}(i|x) > \max_{j \neq i} \hat{p}(j|x)).
\]

Assume that \(C_B(x) = i_0\). Now if \(E\hat{p}(i_0|x) > E\hat{p}(i|x)\) for any \(i \neq i_0\), i.e., \(i_0 = \arg\max_{i} E\hat{p}(i|x)\), when the variance of \(\hat{p}(i_0|x)\) is small enough, we have \(q(i_0|x) > q(i|x)\). In this case \(\arg\max_{i} q(i|x) = i_0\), and \(x\) will be in the Unbiased set with negative boundary bias. In this case, a reduction in variance in each \(\hat{p}(i|x)\) will cause each \(\hat{p}(i|x)\) to become more concentrated towards its expectation, and formula 2.35 means that \(q(i_0|x)\) will increase and the other \(q(i|x)\) will decrease. Then formula 2.27 indicates that the prediction error will also decrease. When \(\arg\max_{i} E\hat{p}(i|x) = j_0 \neq i_0\), it is not clear whether reducing the variance of \(\hat{p}(i|x)\) will reduce the misclassification error. We have the following observation:

1. When \(p(j_0|x) = \max_{i \neq C_B(x)} p(i|x)\), a reduction in the variances of \(\hat{p}(i|x)\) will decrease the misclassification error.

2. When \(p(j_0|x) = \min_i p(i|x)\), reducing the variance will increase the misclassification error.

Therefore, a reduction of the variance in \(\hat{p}(i|x)\) may not necessarily introduce a reduction in classification error. However, the situation for the multiclass case is more complicated than 2-class situation Friedman discussed.

In the next chapter, voting effect of iid decision rules will be discussed. We will show exactly how voting works, and that the “bias” influences the classification error only through whether \(x\) is in the bias set or not. Voting the rules will eliminate the randomness. But it can be either stabilized in the right direction, or maybe on the wrong side.
2.3.3 Conclusion

Decomposing the classification error into noise+bias+variance does not explain exactly how voting works. Formula 2.27 does a better job. It points out that the only way to reduce the error is to change the \( q(i|x) \). Bias influences the misclassification error in whether the point \( x \) is in the bias set or Unbiased set. This means that the idea of a bias set makes more sense than a definition for bias in the usual sense. The variance idea is more practical in that it tells how much variation the space of decision rules has. The way the bias and variance combine to influence the classification error was already obtained by Friedman [1997] for the 2-class question in the special case of plug-in rules. This is obvious in the framework provided here for the multiclass case. The effort in trying to do bias+variance decomposition is somewhat misleading in that there are no perfect definitions for both bias and variance, still carrying their natural meanings. This is easily seen by looking at the multinomial distribution, where the mean and variance are not independent.

In the next chapter, a theory on voting iid decision rules will be given. Theorems obtained there explain how voting works.
CHAPTER 3

THEORETICAL RESULTS FOR VOTING

In this chapter, the behavior of voting iid decision rules is discussed. The space of decision rules can be arbitrary. It can be generated by applying bagging or randomization methods to the same training data, or it can be a heuristic space of rules such as those for PERT or stochastic discrimination. However, the rules should be selected independently from the same distribution. In section 3.1, voting processes \( \{R_n(x), n = 1, 2, 3, \ldots, \} \) are defined. Two theorems about voting processes are proved. Voting effects on the generalization error are also obtained. In section 3.2, weak and strong consistency are introduced. Section 3.3 is devoted to analyzing the voting effect for the nearest neighbor method. In section 3.4, voting in some confined space is investigated, e.g., the effect of voting rules that have training error less than 0.5 – \( \delta \).

3.1 The Voting Processes

Given the learning system \( \mathcal{L} = \{ (\mathcal{D}, p), M, (\mathcal{R}, \nu) \} \), let \( k \) be the number of classes for the classification question, and assume \( n \) rules \( r_1, r_2, \ldots, r_n \) are independently and identically drawn from the rule space \( \mathcal{R} \) with probability measure \( \nu \). Let \( \epsilon_0, \epsilon_1, \ldots, \epsilon_{k-1} \) be iid \( \text{Uniform}(0, 1) \). Define the following:

1. \( q(i|x) = \mathbb{E}_\nu I(r(x) = i) \).
2. \( Z_n(i|x) = \sum_{j=1}^{n} I(r_j(x) = i) \).
3. \( R_n(x) = \text{argmax}_i \{ Z_n(i|x) + \epsilon_i \} \).
4. \( q_n(i|x) = \mathbb{E}_\nu I(R_n(x) = i) \).
5. \( m(x) = \max_i q(i|x) - \max_{j \neq i_q} q(j|x) \) where \( i_q = \text{argmax}_i q(i|x) \).
According to the definition, $q(x) = (q(0|x), q(1|x), \ldots, q(k-1|x))^T$ is a frequency measure on $\mathcal{R}$ at each point $x$. While $\nu$ can be an abstract measure on $\mathcal{R}$, only $q(x)$, the $\nu$-induced measure at each point $x$, is necessary. Therefore, we will discuss the questions in terms of $q(x)$.

$Z_n = (Z_n(0|x), Z_n(1|x), \ldots, Z_n(k-1|x))^T$ is the same in Chapter 2. $Z_n(i|x)$ represents the votes of the $n$ rules for class $i$. Proposition 2.2.3 in Chapter 2 shows that $Z_n$ follows the multinomial distribution with parameters $n$ and $q(x)$.

$R_n$ is the rule resulting from voting $n$ rules, which is conveniently called the $n$-voted rule. The $\epsilon_i$'s are added to deal with ties. As each $\epsilon_i$ is less than 1, they won't change the order of the $Z_n(i)$'s if none of them are equal. When there are ties between two or more classes, it is easy to see that adding iid $\epsilon_i$ will break ties evenly. $q_n(i|x)$ is the measure induced by $R_n(x)$. It is the relative frequency of $R_n(x)$ being class $i$, the same as $q(i|x)$ for one single rule $R_1$.

$m(x)$ is the margin for the collection of decision rules. It is already defined in Definition 2.2.4. It is nonnegative and is not related to the observed class given in the training sample. The usual margin definition (Schapire et al. [1998], Breiman [1999a]) describes the difference between the relative frequency of being correctly classified and the maximum of the relative frequency of being classified as class $i \neq Y$. It is related to the observed or “true” class $Y$.

For example, Breiman [1999a] defines the margin as:

$$m_r(x, Y) = P_\Theta(r(x, \Theta) = Y) - \max_{j \neq Y} P_\Theta(r(x, \Theta) = j).$$

The definition given here has virtues in that it depends only on the collection of rules, and it measures how difficult it is for the collection of rules to discriminate between the most likely class and the next most likely class. It gives control over the rate of convergence as shown in Theorem 3.1.5.
3.1.1 Theoretical Results for Voting Processes

This section contains two basic voting theorems that illustrate how voting works. The bias set consistency theorem guarantees that voting will not change the bias set. The convergence theorem gives a bound on the rate of convergence in terms of the margin function.

**Proposition 3.1.1** (Consistency of bias set, 2-class case)

Let $U_1, U_2, \ldots, U_n$ be iid Bernoulli $(1, p)$ with $p > 0.5$. Let

$$q_n(1|x) = P\left(\sum_{i=1}^{n} U_i > \frac{n}{2} + \epsilon\right)$$

where $\epsilon \sim \text{Uniform}(-0.5, 0.5)$. The following statements are true:

1. If $n$ is an even number, then $q_{n+1}(1|x) > q_n(1|x)$.

2. If $n$ is an odd number, then $q_{n+1}(1|x) = q_n(1|x)$.

**Proof:** Let $X = \sum_{i=1}^{n} U_i$, and $Y = U_{n+1}$, then

$$q_{n+1}(1|x) = P(X + Y > \frac{n + 1}{2} + \epsilon).$$

When $n = 2k$ is even,

$$q_n(1|x) = P(X > k) + \frac{1}{2}P(X = k)$$

and

$$q_{n+1}(1|x) = P(X + Y \geq k + 1)$$

$$= pP(X + 1 \geq k + 1) + (1 - p)P(X \geq K + 1)$$

$$= pP(x = k) + pP(X > k) + (1 - p)P(X > K)$$

$$= pP(X = k) + P(X > k)$$

$$> \frac{1}{2}P(X = k) + P(X > k).$$
This shows \( q_{n+1}(1|x) > q_n(1|x) \) for \( n \) even. The last inequality follows because the assumption that \( p > 0.5 \). When \( n = 2k - 1 \), an odd integer,

\[
q_n(1|x) = P(X \geq k) = P(X > k) + P(X = k)
\]

and

\[
q_{n+1}(1|x) = P(X + Y > k) + \frac{1}{2} P(X + Y = k)
\]

\[
= pP(X + 1 > k) + (1 - p)P(X > k) + \frac{1}{2} (pP(X + 1 = k) + (1 - p)P(X = k))
\]

\[
= P(X > k) + pP(X = k) + \frac{1}{2} (pP(X = k - 1) + (1 - p)P(X = k)).
\]

When \( X \) follows \( \text{Binomial}(2k-1,p) \), \( pP(X = k-1) = (1-p)P(X = k) = \frac{(2k-1)!}{k!(k-1)!} p^k (1-p)^k \).

Therefore:

\[
q_{n+1}(1|x) = P(X > k) + pP(X = k) + (1 - p)P(X = k)
\]

\[
= P(X > k) + P(X = k).
\]

This proves \( q_{n+1}(1|x) = q_n(1|x) \) for the case of odd \( n \).

Note: If we consider \( U_i \) to be the prediction of a decision rule \( r_i \) at some point \( x \), then for the 2-class situation where \( \mathcal{Y} = \{0, 1\} \), \( q_n(1|x) \) defines the probability that the \( n \) voted rules will predict \( x \) as of class “1”. Proposition 3.1.1 says if each rule has the same chance \( p > 0.5 \) of predicting \( x \) as of class “1”, then the more rules voted, the bigger the chance \( x \) will be predicted as “1”. Consider a data point \( (x, y) \) with \( C_B(x) = 1 \). If \( x \) is in the unbiased set \( U \), then \( q(1|x) > 0.5 \). In this case voting more rules will increase \( q_n(1|x) \). This means \( q_n(1|x) > 0.5 \), so \( x \) will be in the unbiased set of the \( n \) voted rules. On the other hand, if \( x \) is in the bias set \( B \), then \( q(1|x) < 0.5 \). Voting \( n \) rules will have \( q_n(1|x) < 0.5 \). Therefore \( x \) will be in the bias set of the \( n \) voted rule. Combining both arguments, it can be concluded that for 2-class case, voting will not change the bias set or the unbiased set.

For the multiclass case, the following result shows that voting will not change the bias set. However, the monotonicity property has not been established.
Theorem 3.1.2 (Consistency of bias set, general case)

Assume the space of decision rules \((\mathcal{R}, q)\) is given. Let \(q_n(i|x)\) be the probability that voting \(n\) rules predicts for class \(i\). For any two classes \(i\) and \(j\), \(q(i|x) > q(j|x)\) implies \(q_n(i|x) > q_n(j|x)\).

Proof: Without loss of generality, we prove \(q_n(0|x) > q_n(1|x)\) when \(q(0|x) > q(1|x)\). Since the discussion is for fixed \(x\), \(x\) is dropped in our notation. Let \(q_i = q(i|x)\), and \(Z^n_i = Z_n(i|x)\), the number of votes for class \(i\) among the \(n\) rules, and \(P(i_0, i_1, \ldots, i_{k-1}) = P(Z^n_0 = i_0, Z^n_1 = i_1, \ldots, Z^n_{k-1} = i_{k-1})\). Then

\[
q_n(0|x) = P(Z^n_0 > \max_{i \neq 0}(Z^n_i + \epsilon_i)) = \sum_{i_0 > \max_i i_j} P(Z^n_0 = i_0, Z^n_1 = i_1, \ldots, Z^n_{k-1} = i_{k-1}) + \sum_{l=2}^n \frac{1}{l} P(l \text{ of } Z^n_i \text{'s achieve the maximum votes including } Z^n_0). 
\]

Notice the mapping that switches the votes for class \(0\) and \(1\) is one-to-one and onto, so it suffices to show that if \(Z^n = (Z^n_0, Z^n_1, \ldots, Z^n_{k-1}) = (i_0, i_1, \ldots, i_{k-1})\), with \(i_0\) among one of the maximum votes, then \(P(i_0, i_1, \ldots, i_{k-1}) \geq P(i_1, i_0, \ldots, i_{k-1})\). The later inequality is true because \(\frac{P(i_0, i_1, \ldots, i_{k-1})}{P(i_1, i_0, \ldots, i_{k-1})} = \left(\frac{q_0}{q_1}\right)^{i_0 - i_1}\), which is greater than 1 if \(i_0 > i_1\) or equal to 1 if \(i_0 = i_1\). This indicates each term in the equation above for \(q_n(0|x)\) is greater than or equal corresponding terms in a similar equation for \(q_n(1|x)\). Thus the theorem is proved.

Remark 3.1.3 It is possible to obtain a monotonicity result like the following: for the class \(i_{\text{max}}\), which has the largest \(q(i_{\text{max}}|x)\), \(q_n(i_{\text{max}}|x)\) increases as \(n\) increases, while for all other classes \(i \neq i_{\text{max}}\), \(q_n(i|x)\) decreases. I have shown that the first part of the claim is true, but failed in proving the second part. Theorem 3.1.5 below gives a convergence result that says \(q_n(i_{\text{max}}|x)\) converges to 1, while all other \(q_n(i|x)\) will converge to 0. It is also possible that the monotonicity holds asymptotically.

We begin the convergence results with lemma for 2-class case.
Lemma 3.1.4 (Convergence of voting process, 2-class case)

Assume the space of rules \((\mathcal{R}, q)\) is given, \(\mathcal{Y} = \{0, 1\}\). If \(m(x) = q(0|x) - q(1|x) > 0\), with \(e \sim Unif(-1, 1)\), then

\[
P(Z_n(0|x) = 1|x) \leq (4q(0|x)q(1|x))^\frac{\alpha}{2} = (1 - m^2(x))^\frac{\alpha}{2}
\]

Proof: We only prove for odd case \(n = 2k + 1\). The even case is similar and is skipped.

For simplicity of the proof, denote \(q(0|x)\) by \(q\). Notice for the 2-class case, \(q(0|x) - q(1|x) > 0\) implies \(q = q(0|x) > 0.5\), or \(\frac{1-q}{q} < 1\). Now

\[
P(Z_{2k+1}(0|x) \leq Z_{2k+1}(1|x)) = \sum_{i=k+1}^{2k+1} \frac{(2k + 1)!}{i!(2k + 1 - i)!} (1 - q)^i q^{2k+1-i}
\]

\[
= q^k (1-q)^{k+1} \sum_{i=k+1}^{2k+1} \frac{(2k + 1)!}{i!(2k + 1 - i)!} \left(\frac{1-q}{q}\right)^{i-k-1}
\]

\[
< q^k (1-q)^{k+1} \sum_{i=k+1}^{2k+1} \frac{(2k + 1)!}{i!(2k + 1 - i)!}
\]

The last inequality holds because \(\frac{1-q}{q} < 1\) and \(i \geq k + 1\). From Binomial distribution, we have

\[
\sum_{i=k+1}^{2k+1} \frac{(2k + 1)!}{i!(2k + 1 - i)!} = \frac{1}{2} 2^{2k+1}.
\]

Using \(\sqrt{\frac{1-q}{q}}\) again, we finally obtain the needed inequality:

\[
P(Z_{2k+1}(0|x) \leq Z_{2k+1}(1|x)) < (4q(1-q))^{\frac{\alpha}{2}}.
\]

Since \(m(x) > 0\), the right-hand side of 3.2 tends to zero as \(n \to 0\). Consider a point \(x\) with \(C_B(x) = 1\). If \(x\) is from the unbiased set, then \(q(1|x) - q(0|x) > 0\). In this case, Lemma 3.1.4 says that the chance the \(n\) voted rule makes an incorrect prediction (i.e., predicts \(x\) to be from class “0”) will go to 0 as \(n\) (the number of rules getting voted) goes to \(\infty\). It also gives a bound for this convergence. In order to guarantee a misclassification error less than
\( \epsilon \), a sample size of \( \frac{2 \log(\epsilon)}{\log(1-m^2(x))} \) will be enough. This lemma also indicates that the smaller the margin, the bigger the sample size needed to obtain satisfactory misclassification error.

**Theorem 3.1.5** (Convergence of voting process, general case)

Assume the space of decision rules \((R, q)\) is given. If \( i_0 = \arg \max_q q(i|x) \), and \( m(x) = q(i_0|x) - \max_{i \neq i_0} q(i|x) > 0 \), then \( E Z_n(i|x) = nq(i|x) \). We have the following bound of convergence:

\[
P(Z_n(i_0|x) \leq \max_{i \neq i_0} Z_n(i|x)) \leq (k-1)(1 - \frac{1}{2} m^2(x))^n. \tag{3.3}
\]

**Proof:** First of all, we notice:

\[
P(Z_n(i_0|x) < \max_{i \neq i_0} Z_n(i|x)) \leq \sum_{i=1}^{k-1} P(Z_n(i_0|x) < Z_n(i|x)). \tag{3.4}
\]

It suffices to show:

\[
P(Z_n(i_0|x) < Z_n(i|x)) < (1 - \frac{1}{2} m^2(x))^n. \tag{3.5}
\]

Without loss of generality, we assume \( i_0 = 0 \), and we want to show:

\[
P(Z_n(0|x) \leq Z_n(1|x)) < (1 - \frac{1}{2} m^2(x))^n
\]

Now

\[
P(Z_n(0|x) \leq Z_n(1|x)) = \sum_{i_0 \leq t_1} \frac{n!}{i_0! i_1! \ldots i_{k-1}!} q_0^{i_0} q_1^{i_1} \ldots q_k^{i_k-1}
\]

\[
= \sum_{l=0}^{n} \frac{n!}{l!(n-l)!} \sum_{i_0+i_1+\ldots+i_{k-1}=l} \frac{l!}{i_0! i_1! \ldots i_{k-1}!} q_0^{i_0} q_1^{i_1} \ldots q_k^{i_k-1}
\]

\[
= \sum_{l=0}^{n} \frac{n!}{l!(n-l)!} \sum_{i_2+i_3+\ldots+i_{k-1}=n-l} (n-l)! q_2^{i_2} q_3^{i_3} \ldots q_{k-1}^{i_{k-1}}
\]

\[
< \sum_{l=0}^{n} \frac{n!}{l!(n-l)!} \left(4q_0q_1\right)^{\frac{1}{2}(1-q_0-q_1)^{n-l}}
\]

\[
= (1 - q_0 - q_1 + 2\sqrt{q_0q_1})^n.
\]
Notice the last inequality holds due to Lemma 3.1.4. When conditioned on $i_0 + i_1 = l$, the probability $P(Z_n(0|x) \leq Z_n(1|x)|Z_n(0|x) + Z_n(1|x)) < (4q_0q_1)^{1/2}$. We also have:

\[
1 - q_0 - q_1 + 2\sqrt{q_0q_1} = 1 - (\sqrt{q_0} - \sqrt{q_1})^2
\]

\[
= 1 - \frac{(q_0 - q_1)^2}{(\sqrt{q_0} + \sqrt{q_1})^2}
\]

\[
\leq 1 - \frac{m^2(x)}{q_0 + q_1 + 2\sqrt{q_0q_1}}
\]

\[
\leq 1 - \frac{1}{2}m^2(x).
\]

Finally Equations 3.4 and 3.5 complete the proof.

\[\square\]

**Remark 3.1.6** This theorem gives the convergence rate of voting processes. For the multiclass situation, the bound is on the margin $m(x)$. Notice that at different $x$, the margin $m(x)$ may be different, and the rate of convergence will differ accordingly.

This is particularly clear for the 2-class situation, where $m(x) = |q(1|x) - q(0|x)|$. If $m(x)$ is close to 0, then it will be hard to discriminate between the two classes, and the convergence will be slow, so lots of rules will be needed in order to achieve good accuracy at $x$. Alternatively, it means iid voting is not efficient in reducing errors for hard points, where $m(x)$ is close to 0. Adaboost (Freund and Schapire [1997]) focuses on these hard points in the later stages. Therefore, these hard points are correctly classified more often. This raises the margin of these points, and this may be a reason why Adaboost performs so well for many datasets.

Figure 3.1 shows the $q_n(1|x)$ for four different points, each with a different $q_1(1|x)$ value. For a point with $q_1(1|x) = 0.9$, $m(x) = 0.8$, the convergence occurs quickly. Within 10 rules, $q_n(1|x)$ has reached 1.0. However, for a point with $q_1(1|x) = 0.53$, the convergence still has not set in even after 500 rules are voted. In real data, there might be quite a few points with $q_1(1|x)$ close to 0.5, and the low prediction accuracy for these points may hinder the overall prediction accuracy.
The rate of convergence for different margin

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.1}
\caption{The $q_n(1|x)$ versus $n$}
\end{figure}

Corollary 3.1.7 (Zero-One law of the voting process)

Assume the space of rules $(R, q)$ is given, $i_0 = \arg \max_i q(i|x)$. If $m(x) = q(i_0|x) - \max_{i \neq i_0} q(i|x) > 0$, then

1. $q_n(i_0|x) \to 1$, for $n \to \infty$.
2. $q_n(i|x) \to 0$, for $n \to \infty$, for $i \neq i_0$.

Proof: Notice that, for $i \neq i_0$,

$$q_n(i|x) = P(Z_n(i|x) > \max_{j \neq i} (Z_n(j|x) + \epsilon_j)) \leq P(Z_n(i|x) > Z_n(i_0|x) + \epsilon_0) \leq P(Z_n(i|x) \geq Z_n(i_0|x)).$$

Then Theorem 3.1.5 completes the proof for $i \neq i_0$. The $i = i_0$ case then follows because $q_n(i_0|x) = 1 - \sum_{i \neq i_0} q_n(i|x)$. \qed

This Corollary gives a theoretical justification that voting will stabilize the classification. It says that voting infinitely many decision rules will finally predict $x$ to be from the class $i_0$ which has the largest $q(i|x)$. Now if $\arg \max p(i|x) = i_0$, i.e., $C_B(x) = i_0$, then $x$ is in the Unbiased set. In this case, voting will predict $x$ to be from class $i_0$ with probability 1. However, it could happen that $\arg \max p(i|x) \neq i_0$, then voting will predict $x$ to be
from a class that is not the same as Bayes rule predicts. Thus it actually increases the misclassification error at \( x \). Whether voting will improve the overall prediction accuracy or not depends on the probability measure of the bias set \( B \) and the margin \( m(x) \) at the points in \( B \). This will be further discussed at the end of this section.

Note that Corollary 3.1.7 can be viewed as a Zero-One law for voting processes. If we let \( A_n = \{ R_n | R_n(x) = i_0 \} \), then \( A_n = \cap i \neq i_0 \{ Z_n(i_0|x) > (Z_n(i|x) + \epsilon_i) \} \). Further let \( A = \bigcup_{k=1}^{\infty} \cap_{n=k}^{\infty} A_n \), the set of \( x \) for which is contained by all \( A_n \) for \( n \) big enough. Now all \( r_j \)'s are iid, and all \( Z_n(i|x) \) are symmetric in the \( r_j \)'s. The Zero-One law of Hewitt-Savage (Billingsley [1995], Chow and Teicher [1997]) then says either \( P(A) = 0 \), or \( P(A) = 1 \). This means either all \( n \) voted rules will predict \( x \) to be of class “\( i_0 \)”, or none of these rules will predict \( x \) to be of class “\( i_0 \)”.

### 3.1.2 Analyzing the Voting Effect for Generalization Error

Assuming that the learning system \( \mathcal{L} = \{ (\mathcal{D}, p), M, (\mathcal{R}, \nu) \} \) is given, we can partition \( \mathcal{X} \) into \( k \) regions, \( A_l = \{ x | \text{argmax}_i q(i|x) = l \}, l = 0, 1, \ldots, k - 1 \). Then the generalization error \( E_\mathcal{L} \) of the learning system, i.e., the classification error for one decision rule is

\[
E_{\text{one}} = E_\mathcal{L} = \int_U e(x)P_X(dx) + \int_B e(x)P_X(dx)
\]

\[
= \int_U e(x)P_X(dx) + \sum_{l=0}^{k-1} \int_{B \cap A_l} e(x)P_X(dx)
\]

where \( e(x) = 1 - \sum_{i=0}^{k-1} p(i|x)q(i|x) \) as given in definition 2.2.5.

The error for voting \( n \) rules, is similar:

\[
E_n = \int_U e_n(x)P_X(dx) + \sum_{l=0}^{k-1} \int_{B \cap A_l} e_n(x)P_X(dx)
\]

with \( e_n(x) = 1 - \sum_{i=0}^{k-1} p(i|x)q_n(i|x) \).

Now for any \( x \in A_l \), Corollary 3.1.7 says \( q_n(l|x) \rightarrow 1 \), and \( q_n(i|x) \rightarrow 0 \), for any \( i \neq l \). Therefore, \( e_n(x) \rightarrow 1 - p(l|x) \). Now if we vote infinitely many rules, the corresponding
misclassification error will be

\[ E_{voted} = \lim_{n \to \infty} E_n = \int_U (1 - p(C_B(x)\mid x))P_X(dx) + \sum_{l=0}^{k-1} \int_{B \cap A_l} (1 - p(l\mid x))P_X(dx). \]

Whether voting will reduce the generalization error depends on the reduction of error in the unbiased set and the possible increase in error on the bias set.

For the 2-class question, a similar calculation gives:

\[ E_{one} - E_{voted} = \int_U [1 - 2p(0\mid x)]q(C_{NB}(x)\mid x)P_X(dx) - \int_B [1 - 2p(0\mid x)]q(C_B(x)\mid x)P_X(dx) \]

where \( C_{NB}(x) = 1 - C_B(x) \), representing the class that is not the class the Bayes rule predicts.

Therefore, whether voting will improve the generalization errors depends on whether the improvement on the unbiased set is greater than the increased error in the bias set. For most classifiers, the unbiased set is bigger, so voting will usually improve the classification error. The formula also indicates that we can easily construct an ensemble of rules for which voting will increase the generalization error.

### 3.2 Weak and Strong Consistencies

The idea of consistency of a statistic concerns its behavior for a large sample. If a statistic is shown to be consistent, then it will work well if we have infinitely many data. How many data we need to get a desired accuracy depends on the variance of the statistic. Maximum likelihood estimate and method of moments estimates are two general principles that will produce consistent estimators. They are considered to be universally consistent under some regularity conditions (Lehman [1983]). For other methods, whether they are consistent or not may be related to the distributions under consideration.

In classification, it is equally important that our classifiers should work well when we have enough data. CART, KNN and artificial neural networks (ANN) are universally consistent (Breiman et al. [1984], Fix and Hodges [1951], Rumelhart et al. [1986]). We introduce
a new kind of consistency idea, which we call weak consistency. The relationship between weak consistency (defined below) and strong consistency (Definition 1.1.4) is discussed. We also define the efficiency of a classification method to be the Bayes deviance defined in Chapter 2. In classification, it’s usually hard to obtain an analytical formula for classification error. We plan to use simulation to study the efficiency problem.

3.2.1 Definition of Weak Consistency

Definition 3.2.1 (Weak consistency)

Assume the learning system \( \mathcal{L} = \{(D, p), M, (R, q)\} \) is given, \( C_B \) is the Bayes rule for \( D \), \( T_n \) is a random training set of size \( n \). Let \( q^{(n)}(i|x) = P_{T_n}(r(x, T_n, M) = i), i = 0, 1, \ldots, k - 1. \)

Then a method \( M \) is said to be weakly consistent if

\[
P_X(\arg\max_i q^{(n)}(i|x) \neq C_B(x)) \rightarrow 0, \text{ as } n \rightarrow \infty.
\]

The definition of strong consistency (Chapter 1) is given by Devroye et al. [1996]. They also give a definition for weak consistency, but it is immediately shown that the two definitions are the same. In our notation, Devroye et al. [1996] defined the error of a decision rule obtained through \( T_n \):

\[
L(T_n) = P_{X,Y}(r(X, T_n, M) \neq Y).
\]

Then strong consistency means:

\[
L(T_n) \rightarrow E_B, \text{ with probability } 1.
\]

Weak consistency means:

\[
E_{T_n}L(T_n) \rightarrow E_B.
\]

Notice here the strong consistency is convergence in probability, and weak consistency is equivalent to convergence in measure. Since \( L(T_n) \) is bounded between \( E_B \) and 1, the two types of convergence are equivalent.
Our definition of weak consistency is a weaker condition than the strong consistency. The set \( B_n = \{ x | \arg \max_i q^n(i|x) \neq C_B(x) \} \) is the bias set of applying method \( M \) on all training sets of size \( n \). For any \( x \notin B_n \), the collection of rules picks the prediction given by the Bayes rule more often than any other class. Weak consistency requires that the probability measure of \( B_n \) tends to 0 as \( n \) goes to \( \infty \) so that at every point, \( x \) will be predicted right on average. Therefore, it is quite a weak condition. The proposition below shows that strong consistency implies weak consistency. We will also show that the NN method and PERT will be weakly consistent. However, for some parametric methods, if the assumed data structure has a discrepancy with the true structure, it is possible that the method will not be weakly consistent. We discuss the relationship between the two types of consistency. We also give an example for which LDA does not satisfy the weak consistency condition. We give an application of the voting theorem with respect to consistency, then finally prove some results for the NN method.

**Proposition 3.2.1 (Relationship between weak and strong consistency)**

Given the Learning system \( \mathcal{L} = \{(\mathcal{D}, p), M, (\mathcal{R}, q)\} \), if \( A_t = \{ x | p(C_B(x)|x) = p(i|x), \text{ for some } i \neq C_B(x) \} \) satisfies \( P_X(A_t) = 0 \), then \( M \) being strongly consistent implies \( M \) will be weakly consistent.

**Proof:** A stronger result for \( q(i|x) \) is proved, i.e., if \( M \) is strongly consistent, then with probability 1, \( q(C_B(x)|x) = 1 \) and \( q(i|x) = 0 \), for any \( i \neq C_B(x) \).

At each point \( x \), Formula 2.27 renders:

\[
\begin{align*}
e_n(x) &= P_{X,Y}(r(x, T_n, M) \neq Y) \\
&= P_{X,Y}(C_B(x) \neq Y) + \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q(i|x) \\
&\geq P_{X,Y}(C_B(x) \neq Y).
\end{align*}
\]

Now since

\[
E_n = \int_X e_n(x)P_X(dx),
\]
and
\[ E_B = \int_{X,Y} P(C_B(x) \neq Y)P_X(dx). \]

Therefore, \( E_n \to E_B \), together with 3.6 implies that except for a set of zero probability,
\[ \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q(i|x) \to 0. \]

Now the set \( \{x|p(C_B(x)|x) - p(i|x) > 0 \text{ for some } i\} \) has probability 0, therefore \( q(i|x) = 0 \) with probability 1 for all \( i \neq C_B(x) \). This means that \( q(C_B(x)|x) = 1 \). Therefore, \( \arg\max_i q(i|x) = C_B(x) \) with probability 1. \( \square \)

**Example 3.2.2** *(LDA is not weakly consistent when the normality assumption fails)*

Let \( \mathcal{L} = \{(D, p), M, (R, q)\} \) be given. Let the two classes have the same prior probability \( \pi_0 = \pi_1 = 0.5 \) with densities:
- Class “\( 0 \)”: \( f_0(x) = \text{pnorm}(x, -1, 1) \),
- Class “\( 1 \)”: \( f_1(x) = \text{pnorm}(x, 1, 1)I(x < 1) + \text{pnorm}(x, 1, 2)I(x \geq 1) \).

where \( \text{pnorm}(x, \mu, \sigma) \) represents the density function of \( N(\mu, \sigma^2) \) at \( x \). Then the Bayes rule \( C_B(x) = “0”, \text{ if } x < 0, \text{ and } C_B(x) = “1” \text{ otherwise.} \) However, the asymptotic LDA line is:
\[ \text{LDA}_\text{cut} = \frac{1}{\sqrt{2\pi}}. \text{ Therefore, the bias set is } \{x|0 < x < \frac{1}{\sqrt{2\pi}} \}. \]

### 3.2.2 The iBoost Algorithm

Weak consistency addresses questions about the average behavior as \( n \) becomes big. Intuitively it is expected that voting independent rules obtained through repetitively applying a weakly consistent method would produce the Bayes rule. One such algorithm is invented that turns a weak classifier \( M \) into a strong classifier \( M_{\text{Boost}} \). We call this algorithm iBoost (i.e., iid boosting) as compared to the distribution-free boosting algorithm of Schapire [1990] and Freund [1995] (to be discussed next).

**The iBoost algorithm**

1. Choose a sequence \( k(n) \) such that: \( k(n) \to \infty \), and \( \frac{k(n)}{n} \to 0 \).
2. For given training data \( T_n \) of size \( n \), randomly partition \( T_n \) into \( k = k(n) \) sub-groups, with each group containing \( \frac{n}{k} \) sample data.

3. Apply \( M \) to each subgroup and obtain \( k \) decision rules.

4. Let \( r(\cdot, T_n, M_{Boost}) \) be the voted rule.

**Theorem 3.2.3** (Boosting theorem for iid voting)

Assume the learning system \( \mathcal{L} = \{(\mathcal{D}, p), M, (R, q)\} \) is given. If \( M \) is weakly consistent, the derived method \( M_{Boost} \) is strongly consistent.

**Proof:** Notice that \( M \) being weakly consistent implies that the size of the bias set will go to 0. For any \( \epsilon > 0 \), there exists \( N_{\epsilon} \) such that \( A_{\epsilon} = \{x| \text{argmax}_iq(i|x) \neq C_B(x)\} \) satisfies \( P_X(A_{\epsilon}) < \epsilon \). Now for any \( x \notin A_{\epsilon} \), when \( \frac{n}{k(n)} > N_{\epsilon} \), \( \text{argmax}_iq(i|x) = C_B(x) \). As \( k(n) \) also goes to \( \infty \), Corollary 3.1.7 guarantees that \( r(\cdot, T_n, M_{Boost}) \) goes to \( C_B(x) \). This proves Theorem 3.2.3.

**Remark 3.2.4** Here \( k(n) \) is the number of independent rules. When \( k(n) \) tends to \( \infty \), then variations will be reduced to 0. This guarantees the convergence of the voting process. Now \( \frac{n}{k(n)} \) is the size of each subgroup, and \( \frac{n}{k(n)} \) goes to \( \infty \), so the weak consistency of \( M \) guarantees that the probability measure of the Bias set of \( M \) will go to 0. This guarantees that the convergence occurs towards the right class at almost every point in the feature space.

A natural question for this methodology is: why should we want to partition the data at all? Might it be better if we just feed the data all at once to the original method \( M \)?

The NN method is a nontrivial example for our boosting theorem. Since NN fits every training sample perfectly, it will suffer extra loss at and around those training points that come from a class different from the Bayes prediction at those points. In this case, no matter how much data we have, the error of the NN method will be bigger than \( e_B \). However, the error of the voted NN rule will go to \( e_B \). The proposition below actually shows that for the 2-class case, the approximate error of NN will be greater than \( e_B \) (unless \( e_B = 0 \)).
Therefore, NN is not strong consistent, and the voting methodology helps to improve its prediction accuracy.

**Proposition 3.2.5 (NN is not strongly consistent)**

Assume that a data structure $D = \{X, \pi, f\}$ is given, with all $f_i(x)$ continuous. Then the asymptotic error of NN is

\[
E_{NN} = E_B + \int_X \sum_{i \in C_B(x)} (p(C_B(x)|x) - p(i|x))p(i|x) P_X(dx)
\]

Therefore, $E_{NN}$ will be greater than $E_B$ unless $E_B = 0$.

**Proof:** At any fixed point $x$, and with a training data of size $n$, Formula 2.27 gives:

\[
e_n(x) = P_{X,Y}(NN(x, T_n, M) \neq Y)
= P_{X,Y}(C_B(x) \neq Y) + \sum_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))q_n(i|x).
\]

The Strong Law of Large Numbers renders $q_n(i|x) \rightarrow p(i|x)$ as sample size goes to $\infty$. An integral over the data space gives the Equation 3.6.

The proposition above can be used to obtain a bound on the asymptotic error for NN method, which is also given in Hand [1981]. When $k = 2$, our result reduces to $E_{NN} < 2E_B(1 - E_B)$, which is already obtained by Cover and Hart [1967] and Stone [1977].

**Proposition 3.2.6** For the $k$-class question, the asymptotic error $E_{NN}$ for NN method can be bounded by:

\[
E_{NN} < 2E_B - \frac{k}{k-1}E_B^2.
\]

**Proof:** First of all, The Bayes error $E_B = P_{X,Y}(Y \neq C_B(X))$. At given point $x$, $P(Y \neq C_B(x)) = 1 - p(C_B(x)|x)$. Therefore, we have:

\[
E_B = \int_X 1 - p(C_B(x)|x) P_X(dx).
\]
Now according to 3.6, we have:

\[ E_{NN} = E_B + \int_X \sum_{i \notin C_B(x)} p(C_B(x) | x)p(i|x)P_X(dx) - \int_X \sum_{i \notin C_B(x)} p^2(i|x)P_X(dx) \]

\[ = 2E_B - \int_X (1 - p(C_B(x) | x))^2P_X(dx) - \int_X \sum_{i \notin C_B(x)} p^2(i|x)P_X(dx). \]

Applying the Cauchy-Schwarz Inequality, we have:

\[ \sum_{i \notin C_B(x)} p^2(i|x)P_X(dx) \geq \frac{1}{k-1} \left( \sum_{i \notin C_B(x)} p(i|x) \right)^2 \]

\[ = \frac{1}{k-1} (1 - p(C_B(x)|x))^2. \]

Combining them, we have:

\[ E_{NN} \leq 2E_B - \frac{k}{k-1} \int_X (1 - p(C_B(x)|x))^2P_X(dx). \]

Another application of the Cauchy-Schwarz Inequality (Integral case) gives the inequality:

\[ \int_X (1 - p(C_B(x)|x))^2P_X(dx) \geq \left( \int_X 1 - p(C_B(x)|x)P_X(dx) \right)^2 \]

\[ \geq E_B^2. \]

Combine this inequality with the previous one, we prove the proposition.

\[ \square \]

**Theorem 3.2.7** NN is weakly consistent, and the voted NN will be strongly consistent.

*Note: The weak consistency of NN will be proved in the next section. The strong consistency of voted NN can be obtained by using Theorem 3.2.3.*

### 3.2.3 Bayes Deviance as a New Efficiency Criterion

In section 2.3, we have defined the Bayes deviance. The equation (2.27) is also obtained that says the misclassification error is positively proportional to Bayes deviance. Bayes
deviance is a very important concept in understanding the voting behavior as seen in section 2.3 and section 3.1. We will further discuss its role as an efficiency measure for classification method.

For convenience, the definitions of pointwise Bayes deviance $d_B(x)$ and the total Bayes deviance are restated below:

\begin{align}
(3.9) \quad d_B(x) &= P_T(r(x,T,M) \neq C_B(x)), \\
(3.10) \quad D_B &= \int_X d_B(x)p(dx).
\end{align}

At each point $x$, $d_B(x)$ represents the average discrepancy of the decision rules from the Bayes rule. Notice the Bayes prediction $C_B(x)$ is really the “right” class we want to pin down; therefore, the $d_B(x)$ represents the amount of deviation of predictions made by the collection of rules from the “right” class. Therefore Bayes deviance can be used as an efficiency measure of a classification method. The rate of convergence of the method is represented by the order of convergence of $D_B$ as a function of training size $n$.

Notice the usual misclassification error is also a criterion to determine how efficient a classification method is. Friedman [1997] proposed to model the efficiency of method $M$ by $e(d,N)$ with the formula:

\begin{equation}
(3.11) \quad e(d,n) = c(d) * n^{-\frac{1}{r(d)}}
\end{equation}

where $d$ is the dimension of the feature space, $N$ is the sample size, $c(d)$ is a scale factor, and $r(d)$ represents the rate of convergence. Notice that both $c(d)$ and $r(d)$ depend on the classification method. An efficient classifier will have small $r(d)$.

This is shown to give quite sensible results for his examples. However, the examples he used have Bayes error equal to 0. When the Bayes error is nonzero, then $e(d,n)$ will be confounded with Bayes error, and will not go to 0 as the sample size $n$ goes to $\infty$. This makes use of $e(d,n)$ as an efficiency measure less attractive.

The Bayes deviance does not have this problem. It has the following advantages:
1. It does NOT contain the Bayes error. For a strong-consistent method, it will go to 0 when sample size $n$ goes to $\infty$, even though the Bayes error is not 0.

2. It represents the variance idea for classification while the usual classification error does not.

3. For oversensitive method, it has the meaning of "influence area." Each "bad point" controls an area in its neighborhood that will introduce extra error.

The Bayes deviance also represents an interesting variation idea when the classification method $M$ is weakly consistent. Notice that weak consistency implies that the probability measure of the bias set goes to 0 as the training size $n$ goes to $\infty$. So that as $n$ is big, for almost all $x$ in the feature space, $\arg\max_q q(i|x) = C_B(x)$, i.e., the aggregated rule will predict $x$ to be the same class the Bayes rule predicts $x$ to be. In this case, we have:

$$d_B(x) = P_T(r(x,T,M) \neq \text{aggregated rule at } x).$$

If we consider the aggregated rule as the counterpart of the sample mean, then the Bayes deviance simply represents the variance idea. From formula 2.27, we also get the following bound on the misclassification error:

$$P_{Y,T}(r(x|T) \neq Y) \leq P_Y(C_B(x) \neq Y) + d_B(x)M(x),$$

where $M(x) = \max_{i \neq C_B(x)} (p(C_B(x)|x) - p(i|x))$, the distributional margin. It is also clear that the classification method is strongly consistent if the Bayes deviance goes to 0 as the training size $n$ goes to $\infty$.

One disadvantage of Bayes deviance is it is hard to estimate. On most occasions, the Bayes rule is not known. Therefore, an empirical estimation of Bayes deviance is not possible. A study on how to obtain an unbiased estimate for Bayes deviance would be interesting.

However, to compare the efficiency of different classification methods, we can use artificial data distributions for which the Bayes rule can be obtained. In Chapter 5, we will design
large simulation studies to compare the efficiency of classification methods introduced in section 1.2.

3.2.4 Relation to Boosting Theory in the PAC Framework

Our boosting theorem is given in a setting that is quite different from the PAC (possibly always correct) framework. For the iBoost algorithm, the decision rules are obtained from independent and identically distributed training datasets. Therefore, each decision rule is considered to be independent and from the same underlying rule space. The PAC framework was introduced by Kearns and Valiant [1988]. Breiman [1998a] also gave a clear description in statistical language. PAC learning is considered to be distribution-free learning. A boosting algorithm aims to convert a weak classifier into a strong classifier. A weak learner in the PAC setting means that, for any small numbers $\epsilon$ and $\delta$, and any data distribution, the learner (classification method) has a chance of at least $\delta$ of finding a decision rule that has error less than $0.5 - \epsilon$. A strong learner in the PAC framework requires that with a chance of at least $1.0 - \delta$, the learner can find a decision rule that has error less than $\epsilon$. So a strong learner is very desirable as it generates accurate decisions for almost all versions of the training data. A boosting algorithm in PAC learning works by repetitively finding “better-than-random-guess” rules, then combining these weak rules into an arbitrarily good classifier. So their weak classifier is different from ours in that our measure is on the rule space $P_T(r(x,T,M) \neq C_B)$, but for PAC learning, the error measure is on the data space. There seems to be a conjugacy between boosting in the PAC framework and the iid voting. It might be an interesting question to explore.

The voting theorem (Theorem 3.1.5) presented here is more like the sample mean in point estimation. Assume that at each point $x$, the rules are on average right, i.e., the bias set is empty. Then Theorem 3.1.5 says that voting many rules will finally reproduce the Bayes rule. Notice that some non-parametric methods such as KNN and CART achieve consistency because they actually do “local voting.” In Chapter 4, we revise the PERT to
give PERT2, which is also consistent as a result of “local voting” (see section 3.3 below). The way we choose how to partition the feature space only influences the estimation efficiency.

3.3 Voting with NN Method

In this section, further discussion are made for voting behavior and its application to NN methods. An interesting result is proved that shows the bias and variance trade-off involved in the voting behavior. For convenience, we restrict our discussion to the 2-class case. The results can easily be generalized to the multiclass case.

3.3.1 k-NN Viewed as Voting k NNs

Suppose the data structure $D = \{\mathcal{X}, \pi, f\}$ is given. We want to predict the class of a given point $x \in \mathcal{X}$. Suppose the training set $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is also given. The NN method predicts the class of $x$ to be the class of the nearest neighbor of $x$ in the training data $T$. The k-NN method picks the $k$ nearest neighbors $\{x^{(1)}, x^{(2)}, \ldots, x^{(k)}\}$ of $x$ from $T$. Then it predicts $x$ to be the majority class of these $k$ neighbors. Notice this voting can be viewed as voting independent rules. Here each neighbor corresponds to a decision rule, and the prediction of the resulting rule is actually the label of that neighbor.

Now if the training set is very big so that all $dk$ neighbors of $x$ have $p(1|x^{(i)}) > p(1|x)$, then Lemma 3.1.1 says that voting these rules will improve the prediction accuracy. In practice, some of the $k$ nearest neighbors may have $p(1|x^{(i)}) < p(1|x)$. The question is: Under what conditions will it be still better if we vote them?

3.3.2 A Heuristic Discussion on Voting
    Two More Neighbors

Lemma 3.1.1 discusses the question of whether adding one more decision rule may help in the case when all rules are independent and have the same $q(1|x)$. It was observed there that the accuracy is improved only when $n$ is an even number. It means that if each time we add two rules, then an improvement will be achieved. To avoid the even/odd difference,
we discuss the case of voting rules of odd totals only. There is no loss of generality here as far as the ideas are concerned.

Suppose there are two independent random variables $X$ and $Y$. $X \sim \text{Bin}(2n-1, p)$ has $p > 0.5$, representing the votes for $2n - 1$ rules, and $Y \sim \text{Bin}(2, q)$, representing the votes for two new rules to be added. We want to find a condition under which voting $X$ and $Y$ will do better than just using $X$. We obtain the following result.

**Proposition 3.3.1** Suppose $X \sim \text{Bin}(2n-1, p)$ with $p > 0.5$, $Y \sim \text{Bin}(2, q)$, and $X$ and $Y$ are independent. Then $P(X + Y \geq n + 1) > P(X \geq n)$ if and only if $q > \frac{\sqrt{p}}{\sqrt{p} + \sqrt{1 - p}}$ or $q > \frac{1}{p + \sqrt{p(1 - p)}}$.

*Proof:*

\[
P(X + Y \geq n + 1) = P(X + Y \geq n + 1, Y = 0 \text{ or } Y = 1 \text{ or } Y = 2)
\]

\[
= (1 - q)^2 P(X \geq n + 1) + 2q(1 - q)P(X \geq n)
\]

\[
+q^2P(X \geq n - 1)
\]

\[
= (1 - q)^2 [P(X \geq n) - P(X = n)] + 2q(1 - q)P(X \geq n)
\]

\[
+q^2 [P(X \geq n) + P(X = n - 1)]
\]

\[
= P(X \geq n) + q^2 P(X = n - 1) - (1 - q)^2 P(X = n).
\]

Therefore,

\[
P(X + Y \geq n + 1) - P(X \geq n) = q^2 P(X = n - 1) - (1 - q)^2 P(X = n)
\]

\[
= q^2 \frac{(2n - 1)!}{(n - 1)! n!} p^{n-1} (1 - p)^n
\]

\[
- (1 - q)^2 \frac{(2n - 1)!}{n!(n - 1)!} p^n (1 - p)^{n-1},
\]

which is positive iff

\[q^2 (1 - p) - (1 - q)^2 p > 0\]

i.e., iff

\[q > \frac{\sqrt{p}}{\sqrt{p} + \sqrt{1 - p}} \text{ or } \frac{q}{p} > \frac{1}{p + \sqrt{p(1 - p)}}.\]
Figure 3.2 gives the lower limit for the quotient $\frac{q}{p}$ such that voting two more rules may help. The minimum of the quotient is about 0.83. This means that if we want to vote more rules and still want to get an improvement in prediction accuracy, we have to have $q$ at least as big as $0.83p$.

In other words, this means that voting more rules may not help to improve the accuracy if $q$ is too small, even though it is still greater than 0.5. The reason lies in the bias and variance trade-off of the variables $Z_1 = \frac{X}{2n-1}$ and $Z_2 = \frac{Y}{2n+1}$. The probability that $X$ gives prediction 1 is: $q_X(1|x) = P(Z_1 > 0.5)$, similarly $q_{X+Y}(1|x) = P(Z_2 > 0.5)$. Although voting more rules will stabilize the final voted rule, it will reduce the expectation of $Z_1$.

The following are the expectation and variance of $Z_1$ and $Z_2$.

1. $E_T Z_1 = p$, $E_T Z_2 = p - \frac{2(p-q)}{2n+1}$.

2. $\text{var}(Z_1) = \frac{p(1-p)}{(2n-1)}$, $\text{var}(Z_2) = \frac{p(1-p)}{(2n-1)} - \left[ \frac{8np(1-p)}{(2n+1)^2(2n-1)} - \frac{2q(1-q)}{(2n+1)^2} \right]$.

If the distributions of $Z_1$ and $Z_2$ are assumed to be normal, then $P(Z_i > 0.5) = 1 - \Phi\left( \frac{E_T Z_i - 0.5}{\sqrt{\text{var}(Z_i)}} \right)$. Therefore, it is possible that reducing the variance will not compensate for the extra error incurred by reducing the expectation.

For a real situation, the question of voting more neighbors can be even more complicated.

Assume we want to estimate the class for point $x$, the three nearest neighbors are $x_1, x_2, x_3$, respectively. Now when we try to decide whether to vote using $x_2$ and $x_3$ or not, it may happen that $q(1|x_2) > q(1|x_1)$ and $q(1|x_3) < q(1|x_1)$, i.e., only one $q$ is larger than $p$. Then the question of whether to take the extra rules into the voting can be even harder. If we let $p = q(1|x_1), \lambda_1 = q(i|x_2) > q(i|x)$ and $\lambda_2 = q(i|x_3) < q(i|x)$. Then similar reasoning gives the following condition for an improvement in voting extra rules:

$$\frac{\lambda_1 \lambda_2}{(1 - \lambda_1)(1 - \lambda_2)} > \frac{p}{1 - p}.$$

From this inequality, it is very possible that adding two extra neighbors won’t hurt much.

This gives an interpretation for what we observed in Friedman’s [1997] figure 8.2. Friedman
[1997] carried out a simulation to investigate how to choose \( k \) for the k-NN method. He observed that for k-NN, the minimum generalization error is obtained at a large \( k \) value. For a training sample of size 3200, the optimal \( k \) is about 1200. He also observed that when \( k \) is between 500 and 2000, the k-NN method produces similar generalization error. This is what our analysis indicates.

### 3.3.3 A Simulation Study for Choosing \( k \)

The data structure Friedman used has 0 Bayes error, and it is uniform on the space of two classes. In this case \( p(1|x) \) is either 0 or 1. For points away from the boundary, the neighbors will usually have equal \( q \) and \( p \). We carry out a similar simulation to check whether what Friedman observed is still true for the more general case when the Bayes error is nonzero. Our simulation gives affirmative conclusions.

**Example 3.3.2** This is a 2-class question with a 20-dimensional feature space. The class conditional distribution is:

- class 0: \( x_1 \) is \( N(-0.2, 0.16) \),
- class 1: \( x_1 \) is \( N(0.2, 0.16) \),
and all other features are independent Uniform(-1,1).

Notice the independent uniform variables serve as nuisance variables to reduce the efficiency of the NN method. To put it another way, they make a larger $k$ necessary to obtain good performance.

The size of the training data is fixed to be 300. A test set sample of size 10000 is set aside before the training set is chosen. Then we apply the k-NN method with different $k$ values. For each $k$, we run the experiment 20 times. The test set misclassification errors are averaged to give the estimate of the errors for this $k$.

Figure 3.3 is very similar to Friedman’s [1997] figure 8.2. It shows that for classification, large $k$ values are needed to give an optimal performance for the k-NN method. Also the wide valley indicates that for many intermediate values of $k$, voting extra neighbors will not help to improve the performance.

3.3.4 The Proof of Weak Consistency of NN

The following lemma says that if the support of a probability density is bounded, then the probability that at least one point will fall in any small hypercube of length $\delta$ will go to 1 as the training size $n$ goes to $\infty$.

**Lemma 3.3.3** Let $X$ be a $d$-dimensional random vector with a continuous probability density $f_X(x)$. Assume that $S$, the support of $f_X(x)$ is bounded. Let $X_1, X_2, \ldots, X_n$ be iid samples from distribution $f_X(x)$, where each $X_i = (X_{i1}, X_{i2}, \ldots, X_{id})^T$ is a $d$-dimensional vector. Let

$$D_{\text{max}}(n) = \max_{1 \leq i \leq n} \max_{1 \leq j \leq n} \{|X_{id} - X_{jd}| \mid \text{interval } (X_{id}, X_{jd}) \text{ contains no sample point}\}$$

then $D_{\text{max}}(n) \rightarrow 0$ in measure, as $n \rightarrow 0$.

**Proof:** Without loss of generality, we prove the lemma for the case of $d = 1$. Let $[a, b]$ be the support of $f_X(x)$. Now for any $\delta > 0$, choose $k$ so that $\frac{b-a}{k} < \delta$. Let $n_k = k^2$. We
The Effect of Choosing different k values for K-NN method

Figure 3.3. Big K value is favored for k-NN method, the case of nonzero Bayes error

can partition interval the \([a, b]\) into \(k\) subintervals of the same size. Denote the intervals by \(I_1, I_2, \ldots, I_k\), and their probability \(P_i = P(X \in I_i)\). As each \(P_i > 0\), there exists a big enough \(N_i\) such that when the sample size \(n > N_i\), \(P(\text{at least one sample falls in interval } I_i) > 1 - \epsilon\). Further, we can choose \(N_n\) so that \(P(\text{at least one sample falls in each interval}) > 1 - \epsilon\).

This means probability that the maximum length of an interval that does not contain one sample point is greater than \(1 - \delta\) is less than \(\epsilon\) for \(n > N_n\). This proves the lemma. \(\square\)

The proof of the weak consistency of the NN classifier can be obtained through the following steps. Denote the sample space by \(\Omega\) and the metric in \(\Omega\) by \(d\).

Proof:

1. If the support of the distribution is not bounded, then find an outer region \(B1\) such that \(\Omega - B1\) is bounded and closed, and \(P(B1) < \frac{1}{2} \times \epsilon\).

2. Choose the boundary region \(B2\) so that it satisfies the following condition:
   i) \(B = \{x| p(i|x) = p(j|x) \text{ for some } i \neq j\} \subset B2\).
   ii) \(P(B2) < \frac{1}{2} \times \epsilon\).
iii) $\delta = d(\Omega - B1 - B2, B) > 0$.

3. Let $A = \Omega - B1 - B2$. Then $P(A) > 1.0 - \epsilon$. As $D_{\text{max}}(n)$ approaches 0, we can fix $N$ such that $D_{\text{max}}(n) < \delta$ if $n > N$, with probability 1.

4. Pick any point $x_0$. There is at least one sample point in the hypercube $(x_0 - \delta, x_0 + \delta)$, otherwise $D_{\text{max}}(n)$ will be greater than $\delta$.

5. In this hypercube, $x$ is away from the intersecting boundary. This indicates that if at one point in the hypercube, class $i_0$ achieves the largest $p(i|x)$, then class $i_0$ achieves the largest $p(i|x)$ at any point in this hypercube (otherwise $A$ will contain some boundary, and $A \cup B \neq \Phi$, which is a contradiction!). Now let $i_0 = \arg\max_i p(i|x) = C_B(x)$. Since for the NN method, $q(i|x) \simeq p(i|x)$ for $n$ big enough, this proves $\arg\max_i q(i|x) = C_B(x)$.

Remark 3.3.4 The weak consistency of the NN method together with iBoosting theorem 3.2.3 proves the strong consistency of k-NN method. In order to make the k-NN method consistent, the following two conditions have to be satisfied:

1) $k \rightarrow \infty$ and 2) $\frac{n}{k} \rightarrow \infty$.

This is exactly what is needed for iBoost algorithm. Therefore, iBoost can be considered to be an extension from k-NN method.

3.4 Voting with Random Methods

In this section, we look at voting from another perspective. Assume that a data structure $\mathcal{D} = (X, \pi, f)$ is given, together with a collection of decision rules, each having generalization error of $0.5 - \gamma$. Our question is: What factors will influence the behavior of the voted rule? We look at a simple example first.
3.4.1 The Ideal Voting Theorem

To make the discussion easier, the concept of VC-dimension (Vapnik [1995]) is introduced.

**Definition 3.4.1 (VC-dimension)**

Assume the classification question is of two classes. For a given collection \( \mathcal{R} \) of decision rules, the VC-dimension of \( \mathcal{R} \) is the maximum number \( n \) so that for any \( n \) points in the feature space with all possible class assignments, there is at least one rule \( r \) from \( \mathcal{R} \) that can separate the classes.

VC-dimension represents the minimum separability of the collection of decision rules. For example, if \( \mathcal{R} \) is the set of all lines in the 2-dimensional plane, then we can always find \( r \in \mathcal{R} \) to separate any three points, but we cannot always separate four points. Therefore, the VC-dimension for lines in the plane is 3. Generally, the set of all hyperplanes in \( d \) dimensional space has VC-dimension \( d + 1 \).

**Example 3.4.1 (Discrete XOR in 2-dimensional space)** The training data consists of four points located in four different quadrants.

Class 1: \((1,1), (-1,-1)\),
Class 2: \((1,-1), (-1,1)\).

Although one line cannot separate these four points, if we vote three lines, we can separate them. The three lines shown in Figure 3.4 can carry out this task, as each point is misclassified at most once, and hence voting three lines can correctly classify all points. Thus voting increases the separating capability, or in other words voting increases the VC-dimension.

The theorem below explains why voting weak classifiers may work in a more general situation. Assume each rule misclassifies \( 100(0.5 - \gamma) \)% of the feature space, with each point having the same chance of being misclassified by one randomly selected rule. Then
Voting increases the VC-dimension

Figure 3.4. Voting three lines can separate four points

the duality between the data space and the rule space indicates that each point will have a probability of $0.5 - \gamma$ of being misclassified.

**Theorem 3.4.2 (Ideal Voting Theorem)**

Given a Learning System $\mathcal{L} = \{(D, p), M, (\mathcal{R}, \nu)\}$, if $q(i|x) = P_r(r(x) = i)$ is constant in $x$, and $E_r = 0.5 - \gamma$ for every $r \in \mathcal{R}$, and $\nu$ is uniform, then $e(x) = 0.5 - \gamma$, i.e., voting all these rules will correctly classify every point.

**Proof:** Define $\mathcal{DR} = D \times \mathcal{R}$, and hence the measure and the integral on that. Further define $Z(r, x, y) = 0$ if $r(x) = y$, $Z(r, x, y) = 1$ if $r(x) \neq y$, then $Z$ is measurable.

Now:

$$E_r = \int_{X,Y} I(Z(r, x, y) = 1) P_X dx P(y|x)$$

and

$$e(x) = \int_{\mathcal{R},Y} I(Z(r, x, y) = 1) P_r P_y (y|x).$$

Since $q(i|x)$ is constant in $x$, $e(x) = \sum_{i=0}^{k-1} (1 - p(i|x)q(i|x)$ is also constant in $x$. Therefore:

$$\int_X \int_{\mathcal{R}} P(r(x) \neq Y) P_r P_X(dx) = \int_X e(x) P_X(dx) = e(x), \text{ for any } x.$$
A similar argument gives:

$$\int_\mathcal{R} \int_X P(r(x) \neq Y) P_r P_X(dx) = E_r.$$ 

The Tonelli Theorem guarantees the exchangeability of the integrals, therefore, $e(x) = E_r$ for any $x$.

Lemma 1 of Kleinberg [1996] gives the same result. Our version is much simpler by introducing a double integral over the space $\mathcal{D} \times \mathcal{R}$ to make it mathematically precise.

Dietterich [1998b] and Breiman [1999a] discuss when an ensemble of rules will be improved through voting. An important factor is the diversity measure (or low correlation) among the classification rules. When we bag a classifier, we perturb the training sample, and usually there is strong correlation among the rules obtained. Dietterich [1998b] showed that the ensemble of rules generated by boosting usually has a big diversity measure. For random methods that do not use optimization, such as PERT and stochastic discrimination, the ensemble of rules is more diverse, as we independently sample from the rule space, with the only constraint being that the rule has to have error rate $< 0.5 - \gamma$. Notice the sacrifice here is the average strength of the collection of rules. We will use simulation to discuss the gain and the loss of using a different parameter $\gamma$.

The conditions for this theorem are quite restrictive. For a learning system, we will not expect that each point will have the same chance of being misclassified by the rules in the rule space. For those points with large margins, convergence will come faster. In other words, if we vote many iid rules (each with the same accuracy), then we may end up wasting lots of votes on the easier points, while at points with low margins, the number of rules might still not be enough. It seems that Adaboost avoids this by focusing on these hard points, improving the convergence rate at these points, and thus achieving an overall improvement.
3.4.2 Voting Random Hyperplanes, an Example

Random hyperplanes can only deal with 2-class problems. Given training data \( T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \), a random hyperplane with training error \(< 0.5 - \gamma\) can be found by choosing a random support point (usually from the training sample) and a random direction. This generates two hyperplane classifiers. One hyperplane classifies one side of the plane to be of class "0", the other side of class "1". The other does the opposite classification. If one of them has training error less than \(0.5 - \gamma\), then we keep it. Otherwise we do another round of sampling until we succeed.

We apply the random hyperplane method to our XOR data in Example 2.2.1. The sample used here is of size 300 (The upper part of Figure 3.5). We use a 41 x 41 grid as test points. Then \(Q(1|x)\), the probability that PERT predicts \(x\) to be of class "1", is estimated by voting 10000 random hyperplanes, each with training set accuracy at least 52%. The \(Q\) surface is plotted in the lower part of Figure 3.5.

We also obtain the learning curve for hyperplane learning (Figure 3.6). For these training data, the training error keeps decreasing until it reaches 23/300 and then it stays there. Overfitting will not happen because the voting process converges. From the graph, we can see that voting random hyperplanes does not work well for these data. The \(Q\) surface for the fourth quadrant is not close to 0. This happens because the training data we have are not symmetric, with fewer points in that quadrant. The decision boundary is pushed toward that quadrant.

3.4.3 A Comparison of PERT and NN

In this subsection, we apply NN method and PERT (Chapter 1) to the same training sample. We compare the \(Q\) surfaces obtained, we also compare the decision boundaries generated by the two methods.

For PERT, each of the individual trees fits the training data perfectly, so that \(Q\) will be 1 at each training point from class "1", and 0 at each training point from class "0". The
XOR Training Data

The Q surface

Figure 3.5. The data and the Q surface for voting random hyperplanes
The Learning Curve for random hyperplane

Figure 3.6. The decision boundary and learning curve for voting random hyperplanes

fact that PERT is fitting the sample perfectly does not cause a problem. The simulations in Cutler [1999] showed that PERT works well for a broad range of classification questions. This is a major difference between classification and regression. Devroye et al. [1996] said that “classification is easier than regression function estimation.” This example clearly shows this point of view.

For this example, a good classifier should make incorrect predictions only on the boundary region. In this case, whether a classifier is efficient or not depends on how well it uses data to obtain the boundary. Our experiment shows that PERT is doing better than NN. It is more stable, while the NN boundary is suffering more random changes. Figure 3.7 gives the estimated boundaries for the two methods.
Figure 3.7. The decision boundaries of PERT and NN method
CHAPTER 4

PROPERTIES OF PERFECT RANDOM TREES

In this chapter, the properties of PERT are discussed in terms of the voting framework. A recursive equation for $Q(i|x)$ is derived in section 4.1. Weak and strong consistency of PERT are discussed in section 4.2. Simulations aiming to explain why PERT is better than the NN method are carried out in section 4.3, and section 4.4 is devoted to discussing two variants of PERT, which are expected to work better than PERT.

4.1 A Recursive Equation for PERT

In this section, the training data $T$ are given. We want to calculate the posterior probability $\{Q(i,x|T), i=0,1,\ldots,k-1\}$ over a bounded region which includes all data samples. Whenever there is no confusion, $Q(i|x)$ is used to denote $Q(i,x|T)$.

4.1.1 The Recursive Equation

Notice when one component PERT tree is built as described in Section 1.4, each successful split involves four random numbers:

1. Randomly pick two sample indices $i$ and $j \in \{1,2,\ldots,n\}$ until $y_i \neq y_j$.

2. Randomly pick an attribute index $l \in \{1,2,\ldots,d\}$.

3. Randomly pick $\lambda \in (0,1)$, and split at $s = \lambda x_{il} + (1 - \lambda)x_{jl}$.

The splitting stops when all samples in each node come from one class; therefore, each PERT tree corresponds to a random sequence of maximum length $n - 1$:

$$(4.1) \quad \theta = \{(i_1,j_1,l_1,s_1),(i_2,j_2,l_2,s_2),\ldots,(i_{n-1},j_{n-1},l_{n-1},s_{n-1})\}.$$  

Assume $\lambda$ is a function that generates a random number in $(0,1)$. Then generation of an index $i \in \{0,1,\ldots,n-1\}$ is actually the result of generating a random number, e.g.,
\[ i = 1 + \text{int}(n \times \lambda) \]. The same thing is true for other indices \( j, l, \) and \( s \). So alternatively, \( \theta \) can be represented as:

\[
\theta = \{ (\lambda_1^{(i)}, \lambda_2^{(i)}, \lambda_3^{(i)}, \lambda_4^{(i)}), i = 1, 2, \ldots, n - 1 \}.
\]

Notice that for different \( i \)'s, \( \lambda_1^{(i)}, \lambda_2^{(i)}, \lambda_3^{(i)}, \lambda_4^{(i)} \) may be dependent. However, the \( \theta \)'s for different PERT trees are independent. Denote by \( \Theta \) the set of all possible \( \theta \), and \( \text{pert}(., \theta) \) the component PERT trees obtained by using \( \theta \).

Consider the space of PERT trees (here we let \( T \) be fixed):

\[
\mathcal{R} = \{ \text{pert}(., \theta) | \theta \in \Theta \}.
\]

We want to calculate

\[
Q(i|x) = P_{\theta}(\text{pert}(x, \theta) = i), \quad \text{for } i = 0, 1, \ldots, k - 1.
\]

We discuss this question for the 2-class, 2-dimensional problem. In this case, it suffices to calculate \( Q(1|x) \), as \( Q(0|x) = 1 - Q(1|x) \). Notice for the given training dataset \( T \), theoretical PERT (voting infinitely many component PERT trees) gives a \( Q \) surface over the feature space. If \( T \) is drawn from the data structure \( D \), then at any point \( x \), if \( (Q(1|x) - 0.5) \times (p(1|x) - 0.5) > 0 \), then applying PERT on this \( T \) will finally be consistent with the Bayes rule at \( x \).

For convenience, we make some changes of notation in this subsection. Let \( T = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \} \) be the training data, where \( y_i = 0 \) or \( 1 \), and \( x_i = (s_i, t_i)^T \) is a 2-dimensional feature vector. Let \( I = (s_0, s_{n+1}) \times (t_0, t_{n+1}) \) be the region in which we want to estimate \( Q(1|x) \). So \( x_0 < x_i < x_{n+1} \) and \( y_0 < y_i < y_{n+1} \), \( i = 1, 2, \ldots, n \). \( \delta \) is the same as defined in Chapter 2, i.e.,

\[
\delta(u) = \begin{cases} 
0, & u \leq 0 \\
0 < u < 1, & u < 1 \\
1, & u \geq 1.
\end{cases}
\]

Other notation is given as follows:
1. $x = (s, t)$, $s$ is the $x$-coordinate, $t$ is the $y$-coordinate.

2. $Q^{(n)}(s, t|T) = Q^{(1)}(x, T)$, the probability that one PERT tree produced by $T$ of size $n$ predicts point $(s, t)$ as of class “1”.

3. Let $s_1, s_2, \ldots, s_n$ be the reordered ascending sequence of $s_1, s_2, \ldots, s_n$, and let $t_1, t_2, \ldots, t_n$ be the reordered ascending sequence of $t_1, t_2, \ldots, t_n$. Notice after this, $s(i)$ and $t(i)$ may no longer represent coordinates of the same point.

4. Let $I^V_l$ be the $l$-th vertical interval from the left, i.e., $I^V_l = (s(l), s(l+1)), l = 1, 2, \ldots, n - 1$. Let $I^H_l$ be the $l$-th horizontal interval from below, i.e., $I^H_l = (t(l), t(l+1)), l = 1, 2, \ldots, n - 1$.

5. $T^L_l$ is the left partition, i.e., the subset of $T$ whose $s_i \leq s(l)$, $T^R_l$ is the right partition, the complement of $T^L_l$. Analogously, $T^U_l$ and $T^D_l$ are defined as the up partition and down partition formed by a horizontal split line in $l$-th horizontal interval.

6. $p(I^V_l)$ is the probability that the first vertical split falls in the interval $I^V_l$, with $p(I^H_l)$ its horizontal counterpart.

The following recursive equation is derived for calculating $Q^{(n)}(s, t|T_n)$.

**Theorem 4.1.1** (Recursive Equation for PERT, 2-class, 2-dimensional case)

Assume the classification problem is a 2-class problem and the dimension of the feature space is 2. Let $T_n$ be a training set of size $n$. With the notation given above, we have:

$$Q^{(n)}(s, t|T_n) =$$

$$\frac{1}{2} \sum_{l=1}^{n-1} p(I^V_l) \delta \left( \frac{s(l+1) - s}{s(l+1) - s(l)} \right) Q^{(l)}(s, t|T^L_l) + \frac{1}{2} \sum_{l=1}^{n-1} p(I^V_l) \delta \left( \frac{s - s(l)}{s(l+1) - s(l)} \right) Q^{(n-1)}(s, t|T^R_l)$$

$$+ \frac{1}{2} \sum_{l=1}^{n-1} p(I^H_l) \delta \left( \frac{t - t(l)}{t(l+1) - t(l)} \right) Q^{(n-1)}(s, t|T^U_l) + \frac{1}{2} \sum_{l=1}^{n-1} p(I^H_l) \delta \left( \frac{t(l+1) - t}{t(l+1) - t(l)} \right) Q^{(l)}(s, t|T^D_l),$$
with \( p(I^V) \) and \( p(I^H) \) calculated as follows:

\[
p(I^V) = \frac{1}{n_0 n_1} \sum_{s_k > s_{l+1} > s_j, y_j \neq y_k} \frac{s_{l+1} - s_l}{s_k - s_j}
\]

\[
p(I^H) = \frac{1}{n_0 n_1} \sum_{t_{l+1} > t_l, t_{j+1} > t_l} \frac{t_{l+1} - t_l}{t - t_j}
\]

with \( n_0 = \) number of “0”’s in the current node \( T_n \), and \( n_1 = \) number of “1”’s in \( T_n \).

Figure 4.1 illustrates the partitions involved and explains how we calculate \( p(I^V) \).

At the beginning of the split, we have the same chance of choosing to do a vertical split or horizontal split, each with chance of \( \frac{1}{2} \). When a vertical split is selected for the next step, the probability that the split line falls in each interval can be calculated using equations 4.6 and 4.7. The recursive equation given above says the probability \( Q^{(n)}(s, t|T_n) \) is the sum of \( Q^{(l)}(s, t|T_l) \times P(T_l) \), where \( T_l \) can be all possible partition at the current node, and \( P(T_l) \) is the probability that the point \((s, t)\) stays in the partition \( P(T_l) \). This recursive equation is possible because if two split lines fall in one interval, they will generate the same partitions, regardless of their actual locations in that interval.

Now for any point \( x = (s, t) \), when the split line falls in an interval that doesn’t contain \( x \), its probability of being classified as “1” will only be related to the partition which is on the same side of the split line as \( x \) is. However, if the split falls in the interval which contains \( x \), then \( Q(1|x) \) will be related to both partitions. For example, if \( x = (s, t) \) lies in the \((l+1)\)-th vertical interval \((s(l), s_{l+1})\), the chance that \( x \) will go with the left partition is \( \frac{s_{l+1} - s}{s_{l+1} - s_l} \), hence the \( \delta \left( \frac{s_{l+1} - s}{s_{l+1} - s_l} \right) \) term in the equation. Also according to equation 4.5, \( \delta(x) \) is nonzero only if \( x > 0 \), so the equation can be simpler than it appears.

Figure 2.1 of Chapter 2 is the surface plot obtained by calculating \( Q(1|x) \) at an \( 81 \times 81 \) grid of points. The sample data are seven points with one from class “0”, the other six points from class “1”. \( Q(1|x) \) is constant on the outmost corner. The blockwise bilinearity (proved late in Proposition 4.1.2) is observed.

The recursive equation provides a way to calculate \( Q(1|x) \) mathematically. We could write programs to calculate it. When all examples in a subtree \( T^R \), or \( T^L \), \( T^U \), \( T^D \) are
The chance that a vertical split falls in the $l$-th interval is the sum of $(s_{l+1} - s_l)$ over all pairs of $(s_R, t_R)$ and $(s_L, t_L)$ from points of different classes and from two sides of the interval, all divided by $n_0 \times n_1$, which is given by formula 4.6.

Figure 4.1. An explanation plot for the calculation of the probability that a vertical split falls in the $l$-th vertical interval.

from the same class, then the corresponding $Q$ will be 0 or 1 according to the class label of the examples. When the examples in a subtree belong to different classes, the recursive equation can be used to reduce the training size $n$ until $n = 2$, for which case, we can use Formula 2.4 obtained in section 2.1, i.e.,

$$Q^{(2)}(s, t|T_2) = \frac{1}{2} \delta \left( \frac{s - s_1}{s_2 - s_1} \right) + \frac{1}{2} \delta \left( \frac{t - t_1}{t_2 - t_1} \right).$$

The computation time would be of order $(n - 1)!$ due to the combinatorial number of ways of choosing a split line at each step. Usually we do not need to calculate $Q(1|x)$. Instead we simply generate $N$ component random trees, and let them vote to get a final decision rule. However, the recursive equation tells us that for 2-dimensional feature space, $Q(1|x, T)$ will be blockwise bilinear on each block formed by the coordinates of the training data.
4.1.2 PERT as a Multilinear Smoothing Method

The following proposition gives a property of PERT. It says that the $Q$ surface for PERT is a blockwise continuous function, with the blocks formed by coordinates of sample points.

On each of these blocks, $Q$ is bilinear.

**Proposition 4.1.2 (2-class, 2-dimensional case)**

Assume $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is given, with $x_i = (s_i, t_i)^T$ and $s(1), s(2), \ldots, s(n)$ and $t(1), t(2), \ldots, t(n)$ are order statistics of the $s_i$'s and the $t_i$'s. Let $I_{ij} = (s(i), s(i+1)) \times (t(j), t(j+1))$, with $i = 0, 1, 2, \ldots, n$, and $j = 0, 1, 2, \ldots, n$. Then the following results are true:

1. $Q(s, t|T)$ is continuous on the feature space.
2. $Q(s, t|T) = ast + bs + ct + d$, on each block $I_{ij}$, with $a, b, c$ and $d$ determined by $I_{ij}$ and the training data $T$.

**Proof:** First of all, the proposition is true for the following two cases:

1. All samples are from the same class.
2. There are only two samples in the training data, from different classes.

For the general case, the recursive equation says that $Q(s, t|T)$ is the sum of terms which are products of factors of the following types: $\delta \left( \frac{s(t+1) - s}{s(t+1) - s(i)} \right)$ or $\delta \left( \frac{s - s(t)}{s(t+1) - s(i)} \right)$ or constant factors like: $\frac{s(i) - s(j)}{s(i) - s(m)}$ and similar terms in $t$. Since $\delta(x)$ is continuous, it is immediate that $Q(s, t|T)$ is also continuous in $s$ and $t$.

Consider any point $(s, t)$ in the outer region such that $s \geq s(n)$ or $s \leq s(1)$, then $Q(s, t|T)$ is constant in $s$. In other words, if two points $(a_1, b)$ and $(a_2, b)$ both satisfy $a_1 \geq s(n)$ and $a_2 \geq s(n)$, with the same $t$-coordinates, then $Q(a_1, b|T) = Q(a_2, b|T)$. This is also true for the $y$-coordinate situation.
For the more general case, at any point \((s, t)\), notice that in the recursive equation, the factor \(\delta \left( \frac{s(l+1) - s}{s(l+1), s(l)} \right)\) is non-constant in \(s\) only if the interval \((s(l), s(l+1))\) contains \(s\). Otherwise it is either 0 or 1 (definition of \(\delta\)). Now whenever the summation goes to an interval \(l\) which contains \(s\), the point \((s, t)\) is in the outer region of both-the left and right partitions generated by the split lines fall in this interval. The \(Q(1|(s, t))\) with these two partitions as training data will be constant in \(s\). This means \(Q(s, t|T)\) can have a linear term in \(s\) at most once. The same argument for \(t\) completes the proof. \(\square\)

**Remark 4.1.3** If the dimension of the feature space \(d > 2\), then in Theorem 4.1.1, the equation will consist of \(2d\) terms. Each variable contributes two terms. The constant factor will be \(\frac{1}{d}\) instead of \(\frac{1}{2}\). Therefore, \(Q(1|x)\) will be blockwise multilinear, i.e., linear in each coordinate when all other coordinates are held constant. For the more general multiclass situation, each \(Q(i|x)\) \((i = 0, 1, \ldots, k - 1)\) satisfies a recursive equation and is blockwise multilinear.

This proposition gives an interesting perspective on what PERT is actually doing on a training data set. We look at the “classification through regression methodology” given below.

Assume again the problem is a 2-class problem with \(\mathcal{Y} = \{0, 1\}\). Let \(x\) be the feature vector, \(T\) be training data. Then one way to obtain a sensible classifier is to use regression to fit a model

\[
y = f(1|x) + \epsilon.
\]

After obtaining such a regression model, the classification rule can be obtained by converting the continuous \(\hat{Y}\) to discrete class labels. The regression methods can be projection pursuit regression (Friedman and Stuetzle [1981]), smoothing spline (Wahba [1990]), kernel smoothers (Härdle [1991]), etc. Here is how this method is actually carried out:

1. Convert the class labels. Let one class be represented by \(y = 0\), the other by \(y = 1\).
2. Fit a linear or nonlinear surface to the converted data to get \(f(1|x)\), using \(T\).
3. Let $C(x|T) = I(\hat{f}(1|x) > 0.5)$.

For the converted data $T$, $y = 0$ or $y = 1$. Now consider PERT as one such method and $Q(1|x)$ as an estimate of $f(1|x)$. Then $\hat{f}(1|x_i) = Q(1|x_i) = 1$ for all training points from class “1” (i.e., $y_i = 1$), and $\hat{f}(1|x_i) = Q(1|x_i) = 0$ for all training points from class “0” (i.e., $y_i = 0$). Therefore, PERT is fitting a regression surface that passes through all the training data. Furthermore, the proposition above indicates that PERT is fitting a continuous function that is blockwise multilinear, again in the sense that it is linear in each coordinate.

The NN method gives a different smoothing surface. It also fits the training sample perfectly, but it only takes values in \{0, 1\}, so it is discontinuous. Schapire et al. [1998] showed that, when combined with c4.5, AdaBoost can usually reduce the training error to 0 after five iterations. In this way, it also fits the training samples perfectly. Therefore, they are all oversensitive methods (Definition 1.4.1). In section 4.3, we will discuss the behavior of PERT using characteristics of oversensitive methods.

## 4.2 Consistency Issues for PERT

### 4.2.1 Oversensitive Methods

An oversensitive method classifies every sample point correctly. It suffers extra error when a sample point actually comes from a class which is not the one the Bayes rule predicts. For example, consider the normal mixture $f(x) = 0.5 \times N(-1, 1) + 0.5 \times N(1, 1)$, where $N(-1, 1)$ is class “0” and $N(1, 1)$ is class “1”. The Bayes rule for this distribution is:

$$C_B(x) = \begin{cases} 
0, & \text{if } x < 0 \\
1, & \text{if } x > 0 .
\end{cases}$$

Now suppose $x_0 = -0.1$ is in the sample, and it is from class “1”. An oversensitive method classifies it as of class “1”, different from what the Bayes rule would predict. When such an oversensitive rule is applied to the theoretical distribution, it suffers a loss $p(0|x)$ at $x$, and the Bayes rule suffers a loss of $p(1|x)$ at $x$. Therefore, decision rule $r$ obtained
by an oversensitive method will suffer an extra error of $p(0|x) - p(1|x)$ at $x$. If the decision rule $r$ is "continuous", i.e., $Q(1|x)$ is continuous in $x$ (or in the more general case, when $r$ classifies a point $x$ to be of class "1", then it will also classify all points in some small neighborhood $\Delta(x)$ to be of class "1"), then $r$ will also suffer extra error in the neighboring region.

Given the training data $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, all from class "1", and $(x_0, y_0)$ from class "0", for a given classification method $M$, we define the neighborhood $\Delta(x_0)$ of $x_0$:

$$\Delta(x_0) = \{x | r(t, T, M) = y_0 \text{ for any } t \text{ on the line segment connecting } x_0 \text{ and } x\}.$$ (4.9)

Notice that an oversensitive method $M$, $\Delta(x_0)$ represents the region that is classified as $y_0$ because of the existence of the training point $(x_0, y_0)$. Therefore, it is like the region controlled by $x_0$. Imagine that $(x_0, y_0)$ is an outlier, it is clear that the smaller $\Delta(x_0)$ is, the better the method $M$ works. We then further define the probability of $\Delta(x_0)$ to be the influence area.

$$\text{IA}(x_0) = P_X(\Delta(x_0)).$$ (4.10)

More discussions will be made on the behavior of different methods in terms of influence area in section 4.3. This analysis will shed some light on the questions of why PERT generally works well and why it sometimes fails. A comparison with the NN method and boosted c4.5 will also be made with simulations.

For the 2-class question, taking expectations of two sides of Formula 2.24 gives the overall extra error for a given decision rule $r$ compared to the Bayes rule $C_B$:

$$E_{extra} = E_r - E_B = \int_{\{r(x) \neq C_B(x)\}} |p(0|x) - p(1|x)| P_X(dx).$$ (4.11)

A classification method $M$ is strongly consistent if for almost all sequences of training data $T_n$, the misclassification error $E_{r_n}$ of $\{r(., T_n, M), n = 1, 2, \ldots\}$ satisfies:

$$E_{r_n} - E_B \rightarrow 0, \text{ as } n \rightarrow \infty.$$ (4.12)
4.2.2 Strong Consistency of PERT

The following example is designed to show that PERT is not strongly consistent. The key idea here is the Bayes deviance ($D_B$) introduced in Chapter 2. Bayes deviance represents the probability of the set where the target decision rule $r$ disagrees with the Bayes rule, i.e., $D_B = P_X(r(X) \neq C_B(X))$. Example 4.2.1 is constructed so that $p(1|x) - p(0|x)$ is constant over the feature space. As a result of this, equation 4.11 becomes:

$$E_r - E_B = \text{constant} \times P_X(r(X) \neq C_B(X)) = \text{constant} \times D_B.$$

For Example 4.2.1, it is shown $D_B$ does not go to 0 as the size of the training data goes to $\infty$. Therefore, the extra error does not go to 0, and PERT is not strongly consistent for this example. Notice the same conclusion can still be drawn even if the data structure is not such an extreme one.

**Example 4.2.1** Let $X$ follow a mixture of two bivariate uniform distributions. Both components are $U(0, 1) \times U(0, 1)$, but with different priors, $\pi_0 = 0.4$ for class “0”, $\pi_1 = 0.6$ for class “1”.

Notice for this distribution, the Bayes rule is $C_B(x) = 1$, for any $x$, with Bayes error $E_B = 0.4$. Also according to the Formula 3.6 in Proposition 3.2.5, the asymptotic error for NN is $0.4 + 0.4 \times 0.2 = 0.48$. For non-oversensitive methods such as KNN and CART, it is easy to see they are consistent because of their local voting behavior. That actually makes their $r(., T_n, M)$ converge to the Bayes rule.

Now for PERT, each point of class “0” in the training sample introduces an extra error of $|p(1|x) - p(0|x)| = 0.2$. Since $C_B(x) = 1$ for any point $x$, the total extra error for PERT is

$$E_{\text{extra}} = 0.2 \times P_X(\text{PERT}(X) \neq 1) = 0.2 \times D_B.$$

Since the distributions of both classes are uniform, $D_B = P_X(\text{PERT}(X) \neq 1)$ is the area of the set $\{x | Q(1|x) < 0.5\}$. We use simulation to estimate this area.
We carry out two simulations here. The first one is to compare PERT to NN. We show that when the Bayes error is small, then PERT works much better than the NN method, but as the Bayes error increases, the two methods are more similar. This is done with a fixed small sample size. In the second simulation, we fix the Bayes error to be $E_B = 0.4$, we increase the training size, and watch out for the changes in classification error.

For Simulation 1, we choose different priors for $\pi_0 = 0.02, 0.05, 0.1, 0.2, 0.4$. Notice for the mixture of uniform distribution, $E_B = P\pi_0$. Now for each prior, a training set of size 100. Both PERT and NN are applied to the same training data, and Bayes deviance and misclassification error are obtained by applying the rules to a test set of size 10000. The result is given in Table 4.1. It shows that when $E_B$ is small, $D_B = P(\text{PERT} \neq C_B)$ can be significantly smaller than $P(\text{NN} \neq C_B)$, thus a lower classification error is achieved. However, when $E_B$ becomes bigger, this difference slowly disappears. The classification error of PERT goes towards the error of NN. The relationship between $D_B$ and $E_B$ for PERT is observed to be $D_B \sim O(E_B^3)$ when $E_B$ is small. This will be further studied in section 4.3.

The result of the second simulation is given in Table 4.2. For $E_B = 0.4$, even if the size of the training dataset increases, the generalization error does not further decrease. This means the analysis for Simulation 1 is also valid for big sample size.

An intuitive interpretation is like this: when $E_B = 0.05$, there are about 19 class "1" points around each class "0" point. Fitting a function like $f(x, y) = xy$ will squeeze the area controlled by class "0" points. In this case, a class "0" point is like an outlier in the

<table>
<thead>
<tr>
<th>Priors $\pi_0$</th>
<th>0.02</th>
<th>0.05</th>
<th>0.1</th>
<th>0.2</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\text{PERT} \neq C_B)$</td>
<td>0.0058</td>
<td>0.0081</td>
<td>0.0179</td>
<td>0.1025</td>
<td>0.3978</td>
</tr>
<tr>
<td>$P(\text{NN} \neq C_B)$</td>
<td>0.0300</td>
<td>0.0491</td>
<td>0.0688</td>
<td>0.1914</td>
<td>0.4373</td>
</tr>
<tr>
<td>$P(\text{PERT} \neq Y)$</td>
<td>0.0249</td>
<td>0.0559</td>
<td>0.1132</td>
<td>0.2636</td>
<td>0.4840</td>
</tr>
<tr>
<td>$P(\text{NN} \neq Y)$</td>
<td>0.0473</td>
<td>0.0939</td>
<td>0.1543</td>
<td>0.3163</td>
<td>0.4889</td>
</tr>
</tbody>
</table>
Table 4.2. Simulation 2. Misclassification Error as Sample Size Increases

<table>
<thead>
<tr>
<th>Sample size</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td>$P(PERT \neq C_B)$</td>
<td>0.34222</td>
<td>0.32670</td>
<td>0.32504</td>
<td>0.31870</td>
</tr>
<tr>
<td>Classification error</td>
<td>0.46941</td>
<td>0.46361</td>
<td>0.46632</td>
<td>0.46433</td>
</tr>
</tbody>
</table>

sea of class “1” points. The area $(D_B)$ controlled by such a point is approximately $O(E_B^2)$, while for NN method, $D_B = O(E_B)$. Now when $E_B = 0.4$, there are about 1.5 class “1” points around each class “0” point. This is not enough to push the blockwise bilinear curve towards to the class “0” points, and $D_B$ is more likely to be $O(E_B)$. Thus PERT cannot achieve good performance for this situation.

This nonconsistency of PERT is basically due to its oversensitive nature. It is pointed out by Friedman [1997] that the damage of oversensitivity is much less severe for classification than for regression. However, oversensitive methods will not be Bayes consistent. It is very possible that boosting methods will not be Bayes consistent for many distributions, even though boosting methods do work very well compared to other methods. Another pertinent question is whether PERT and boosting methods will be Bayes consistent if the distribution under consideration has zero Bayes error. This is probably true, but it is not proven yet.

4.2.3 The Weak Consistency of PERT, a Conjecture

As pointed out in section 3.2, the weak consistency addresses the question of whether the Bias set of a classification method will diminish as the size of the training data increases. For the NN method, this is true (Theorem 3.2.7). For any point $x$, NN uses the nearest neighbor of $x$ to predict the class of $x$. Now suppose that $p(i_B|x) > \max_{i \neq i_B} p(i|x)$ (here we let $i_B = C_B(x)$ for convenience). Then the continuity of $p(i|x)$ will guarantee that $\arg\max p(i_B|t) = i_B$, for any $t \in \Delta_r(x)$, for some neighborhood of $x$. In this case, as the size of the training data increases, there will be at least one point $x_{NN}$ in $\Delta_r(x)$. The
chance that $x_{NN}$ is of class $i_B$ is the largest among all classes. This is how we showed NN is weakly consistent in Chapter 3.

It is natural that we would expect each oversensitive method to be weakly consistent. However, the example given below shows some conditions are needed so that an oversensitive method will be weakly consistent. The example, which we call InverNN, is constructed via the NN method. It is constructed so that at many points $x$, InverNN predicts $x$ to be the opposite class of what NN predicts.

**Example 4.2.2** InverNN (two class only), an oversensitive method that is not weakly consistent.

Suppose we have training data $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Let $\text{dist}(., .)$ be a metric defined on the feature space. Let $D_{\text{min}} = \min_{i \neq j} \text{dist}(x_i, x_j)$, let $N_i$ be the set of $x$ such that $i = \arg\min_j \text{dist}(x_j, x)$. We define InverNN as follows:

\[
\text{InverNN}(x) = \begin{cases} 
    NN(x), & \text{if } x \in N_i \text{ and } \text{dist}(x, x_i) < 0.1 \ast D_{\text{min}}, \\
    1 - NN(x), & \text{else}.
\end{cases}
\]

For each set of given training data, InverNN differs from NN on over 90% of the feature space. NN is a weakly consistent classifier. So when the sample size increases, NN will correctly classify more often. This indicates that InverNN will make the wrong prediction more often. Therefore, InverNN will have the whole feature space as its Bias set. A formal proof can be obtained by the ideal voting theorem in section 3.4.

This example shows some regularity condition is needed for an oversensitive method to be guaranteed to be weakly consistent. This condition should be related to the smoothness of the fitting surface. Finding a general condition for this question is an interesting but challenging question. It is beyond the scope of this thesis.

Weak consistency is really a very weak requirement. The NN method is weakly consistent. Proposition 4.1.2 indicates that PERT can be considered as a smoothing of NN. We conjecture that PERT is also weakly consistent. This will be further studied in later research.
4.3 Resistance Study for PERT as an Oversensitive Classifier

The previous section shows that the reason why PERT works well is not in its strong consistency, as PERT is actually not strongly consistent. In this section, we consider PERT, NN method and boosted c4.5 as oversensitive methods. We show that one reason that PERT works well is in the squeezing effect discussed below.

4.3.1 The Squeezing Effect

This subsection considers how an outlier in an otherwise pure distribution may influence the construction of a decision rule and its misclassification error for an oversensitive method. Consider the special case of 2-class problem with \( Y = \{0, 1\} \) and a 2-dimensional feature space. The distribution for training data of size \( n \) is:

\[
(4.15) \quad f_{\text{contaminated}}(x) = \frac{n-1}{n} f(x) + \frac{1}{n} I_{\{x_0\}}(x)
\]

Different oversensitive methods (such as NN, PERT, and boosted c4.5) are compared on their different influence areas for the outlier \( x_0 \).

Example 4.3.1 Let \( n \) be the size of training dataset. Class “0” consists of one point located at the origin \( x_0 = (0, 0) \). All the other \( n-1 \) points are from class “1”, each is uniformly distributed on \((-1,1) \times (-1,1)\).

If all data are from class “1”, then a sensible method will classify the whole feature space to be of class “1”. Now with one point \( x_0 \) from class “0”, an oversensitive method will classify \( x_0 \) to be of class “0”. It is clear that the smaller the influence area \( P_X(\Delta(x_0)) \) of \( x_0 \) is, the more resistant the classification method is to outliers, and the more accurate the resulting decision rule will be. For the experiment below, we estimate the influence area for point \( x_0 \) for NN and PERT. The relationships between influence area and \( n \) are shown in Figure 4.2. This simulation shows that for one outlier, the influence areas of PERT and
NN follow different relationships:

\begin{align}
\text{IA}(\text{NN}) & = O\left(\frac{1}{n}\right), \\
\text{IA}(\text{PERT}) & = O\left(\frac{1}{n^2}\right).
\end{align}

This means that an outlier will have less influence for PERT than for the NN method. This gives one reason why PERT works better than the NN method.

For a general distribution, there is more than one point whose class is not the same as the Bayes rule predicts it to be (these will be called “bad” points). An oversensitive method will classify such points to be the observed class, and an extra error will be introduced around them. We need to know the total influence area of these “bad” points in order to know how good a classification method is. It turns out this total influence area is the Bayes deviance defined in section 2.3, i.e., $D_B = P_X(r(X) \neq C_B(X))$.

For Example 4.2.1 below, it is shown that for PERT, the relationship $D_B = O\left(E^2_B\right)$ still holds. Notice this distribution is very special, as $p(1|x) - p(0|x)$ is constant in $x$. But it does show that for PERT, classes which have fewer training points will usually get squeezed. For the same example, the NN method has $D_B = O(E_B)$ since each point controls an expected
Figure 4.3. Bayes deviance versus Bayes error for PERT

area of \( \frac{1}{n} \). We will find the squeezing effect of boosted c4.5 in section 5.1.

**Example 4.3.2** *(Example 4.2.1 revisited)* Here we let the priors of the two classes change. We let \( \pi_0 \) go from 0.01, 0.02, \ldots, 0.39, 0.40, and watch out for the change in \( D_B \). Notice in this case, the Bayes error for this example is \( E_B = \pi_0 \). The results are given in Figure 4.3. It is quite clear that \( \sqrt{D_B} \) is proportional to \( E_B \), i.e., \( D_B \) is linearly proportional to \( E_B^2 \).

### 4.3.2 Curse-of-Dimensionality

The curse-of-dimensionality is a property of high dimensional feature spaces. Breiman et al. [1984] gives the following example to illustrate the curse-of-dimensionality. Given 100 points, constructing a 10-cell histogram on the unit interval is a reasonable procedure. In 20-dimensional space, 100 points is like “oases in the desert,” and a histogram constructed from 100 data points in 20 dimensions will be very inaccurate. Devroye et al. [1996] also provides some view on curse-of-dimensionality.

The NN method is known to be greatly influenced by the curse-of-dimensionality. The result of this is a huge variation among the decision rules. Suppose a point \( x \) is fixed, which
is close to the boundary, and \( p(0|x) = 0.4, p(1|x) = 0.6 \). Let \( I \) be the largest interval that contains \( x \) such that \( p(1|x) > p(0|x) \) for any \( x \) in \( I \). Let \( p_I \) be the probability such that one point will fall in this interval. Now if the nearest neighbor \( t \) of \( x \) is in \( I \), then \( x \) is more likely to be classified as the right class. The sample size we need so that we expect to get one point in this interval is \( \frac{1}{p_I} \).

Now imagine the data is \( d \) dimensional, and different coordinates are independent and identically distributed, then a sample size of \((\frac{1}{p_I})^d\) will be required so that there will be one point in the hypercube. If we let the sample size be fixed, then the classification error will degrade quickly as the dimensions increase. Ji and Ma [1997] uses the overlapping normal distribution (Example 4.3.3) to carry out such a simulation to show that NN and neural networks suffer from the curse-of-dimensionality, while the boosted hyperplane (as introduced in section 3.4, for details see Ji and Ma [1997]) is free of the curse-of-dimensionality.

In this subsection, the same example (such as the one given in Figure 4.4) gives a training set of is used to investigate how PERT works in terms of the high dimensionality. Figure 4.5 shows that PERT suffers much less severely from the curse-of-dimensionality. But it does work worse as the dimensions keep increasing. Experiments are also carried out for two variants of PERT, PERT2, and PERTQ (details in the next subsection). They appear to be free of the curse-of-dimensionality even for \( d = 25 \).

**Example 4.3.3 (Overlapping normal)**

This is 2-class, \( d \)-dimension data. Class “0” are drawn from \( N(0,I) \) and class “1” are drawn from \( N(0,4I) \), where \( I \) is the \( d \times d \) identity matrix. The two classes are drawn with equal priors.

Notice the Bayes error for this distribution is:

\[
E_B(d) = \frac{1}{2}(1 - \text{pchisq}(D_d^2, d) + \text{pchisq}(D_d^2/4, d)),
\]

with \( \text{pchisq} \) representing the cumulative density of the Chi-square distribution. The Bayes
Figure 4.4. 2-dimensional training data from the overlapping normal

rule is:

\[ C_B(x) = \begin{cases} 
0, & \text{if } dist(x) < D_d \\
1, & \text{else} 
\end{cases} \]

where \( dist(x) \) represents the Euclidean distance from \( x \) to the origin. \( D_d = \sqrt{\frac{8}{3} \ln 2} \), is the radius of the decision boundary. Figure 4.4 is an example of a training set of size 200, with the Bayes decision boundary drawn.

Following Ji and Ma [1997], we choose the training size to be 2000 and testset size to be 4000. Then we increase the dimensions from 2 to 25. For each method, the misclassification error is estimated by the average of 100 errors, each of which is obtained by applying the method on one set of training and test data. The result is given in Figure 4.5.

PERT seems to be free of the curse-of-dimensionality when the dimension \( d \) is less than 6. However, as the dimension \( d \) increases further, PERT can not catch the structure well. The difference between the Bayes error and PERT's error increases as \( d \) increases. The two variants PERT2 and PERTQ, which will be introduced in the next section, work well for \( d < 25 \). In the following subsection, we try to give an interpretation of why PERT also suffers from the curse-of-dimensionality, although it is less severe than for NN.

Notice for the simple case where there is only one outlier from class "0" and the dimension is 2, the outlier is surrounded by class "1" points. This way, the squeezing effect of
Different behaviors on curse-of-dimensionality

Figure 4.5. Different behavior as dimensionality of feature space increases

PERT comes into play, and each point will help to push the boundary towards the outlier, thus the influence area of the outlier shrinks. In high dimensional cases, the geometry is kind of fuzzy, because the data are generally very sparse. For example, when \( d = 20 \), if we want to have one point in each quadrant, it requires a sample of size \( 2^{20} \), about 1,000,000. Therefore, in a 20-dimensional space, a sample of size 2000 is really sparse. Neither of the classes enjoys a “local” majority, and the squeezing effect will not happen. In other words, for those points in the inner region of the data, their influence region may stretch out to the boundary, as it is not squeezed backwards. This is like having only three data points in the two dimensional case, so the influence of one point will be very big.

To verify this intuition, we calculate the influence areas of those that are in the inner region and check to see whether these regions stretch outside the Bayes decision boundary. We keep track of those points that are misclassified. We carry out the simulation for Example 4.3.3 with \( d = 21 \), where the radius of the decision boundary is \( r = 38.82 \). Again we use training size=2000, testsize=4000. The average error for 10 runs is 6.89%. Among the misclassified points, more than 5.90% are those that are outside the Bayes boundary, and being misclassified as class “0”. This suggests that our reasoning is correct. Our
simulation shows that boosted c4.5 works well for this data, the error being around 4.6%. It works well because the base classifier tree method is stronger, thus helping to reduce the error rate outside the Bayes decision boundary.

4.4 Two Variants of PERT

Cutler [1999] reported that PERT does not work well for Ringnorm. The previous experiments show that the reason is the curse-of-dimensionality, or put it another way, its inefficiency in using the information in the data. It is very clear that for the NN method, the prediction at one point $x$ uses very little information and only local information.

PERT is doing better, but it is not enough. We suggest a new variant of PERT that will stop splitting the node when the number of data points in the node is less than a given number. Again, these component trees are then aggregated to give a final decision rule. Notice that this procedure does not involve any of the optimization steps usually used by tree methods such as CART or c4.5.

Another factor that makes the new variant of PERT attractive is that PERT might have difficulty in discriminating between the classes if the classes are very mixed up. The problem here is its oversensitive nature. For the example 4.2.1, if the training size $n = 10000$, one PERT component tree may have close to 10000 nodes. This is caused by the nonstructural nature of the problem. This problem can be solved if we stop early, and let them vote to get the right class.

We call the new variant of PERT PERT2. We prove the strong consistency of PERT2 under suitable conditions. Then we use simulations to check its performance. We also propose another variant PERTQ of PERT based on the squeezing interpretation.

4.4.1 PERT2, a Variant of PERT That Stops Early

Given the training sample of size $N$, $T = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$, with each $x_i$ a d-dimensional vector. Estimate the priors for each class $\{\pi_0,\pi_1,\ldots,\pi_{k-1}\}$. Choose a
tuning parameter $\lambda$.

Initialize the stack so it contains one node comprising $T$ itself. Repeat until there are no more nodes in the stack.

1. If the stack is empty, stop. Otherwise take a node from the stack.
2. Count the numbers $n_i$ of class $i$ in the current node. If $n_i < \lambda N \pi_i$ for some $i$, the node is a terminal node. Vote the examples in this node to give a prediction. Go to step 1.
3. Randomly pick two indices $i$ and $j \in \{1, 2, \ldots, n\}$ until $y_i \neq y_j$.
4. Randomly pick an attribute index $l \in \{1, 2, \ldots, d\}$.
5. Randomly pick a number $s$ between $x_{il}$ and $x_{jl}$.
6. Divide the parent node into two child nodes,
   - left node = all examples with $x_{il} > s$, put this node on the stack.
   - right node = all examples with $x_{il} < s$, put this node on the stack.
7. Go to step 1.

The above procedure is repeated $N$ times and the random trees are voted to give the final decision rule.

Note here that in each terminal node, there are samples from each class. When the sample size becomes big, then the sample proportions for each class will tend to their theoretical counterparts, and this will make PERT2 Bayes consistent.

**Lemma 4.4.1** Let $\{\lambda_1, \lambda_2, \ldots, \lambda_n, \ldots\}$ be a sequence that satisfies: (1) $\lambda_n \to 0$ and (2) $n\lambda_n \to \infty$, then for any point $x$, Let $D_{\text{max}}$ be the diameter of the terminal node containing $x$, then $D_{\text{max}} \to 0$ in probability.

**Theorem 4.4.2** (Strong consistency of PERT2)

For any data structure $D = \{X, \pi, f\}$, if the resulting posterior probabilities are all continuous, then for any given sequence of tuning parameter $0 < \lambda_n < 1$, if $n\lambda_n \to \infty$, then PERT2 using this sequence $\{\lambda_n, n = 1, 2, \ldots\}$ will be strongly consistent.
Proof: Let $X$ be the random feature, and $\mathcal{X}$ be the feature space. Firstly, there exists a bounded closed set $A$ such that $P(\mathcal{X} - A) < \epsilon$. Now for any point $x \in A$, which has $f_X(x) > 0$, assume $d_x = p(i_0|x) - \max_{i \neq i_0} p(i|x) > 0$. The continuity of $f_i(x)$ and therefore, $p(i|x)$ guarantees that there is an open neighborhood $\Delta(x)$ such that:

1. $P(\Delta(x)) > 0$.
2. for any $t \in \Delta(x)$, $d_t = p(i_0|t) - \max_{i \neq i_0} p(i|t) > \frac{1}{2}d_x$.

Lemma 4.4.1 guarantees that the terminal node containing $x$ will finally fall inside $\Delta(x)$. Condition 2 then guarantees that voting this nodes will have $\hat{p}(i_0|x) > \max_{i \neq i_0} p(i|x)$. Finally, the Heine-Borel covering theorem guarantees that such a convergence can be made universally.

Remark 4.4.3 Many other proofs of consistency try to prove that $\hat{p}(i|x)$ converges to $p(i|x)$. For classification, it suffices to show that $\arg\max\hat{p}(i|x)$ converges to $\arg\max p(i|x)$. The latter case is usually easier. Theorem 4.4.2 assumes the continuity of densities for each class. It helps with the proof and is an easy-to-satisfy condition in real life.

4.4.2 PERTQ, Another Variant of PERT That Uses Quadratic Terms

For the ideal case when there is only one outlier in the training data, and the feature space is 2-dimensional, we empirically showed in section 4.3 that for the NN method, the influence area of one outlier is $O\left(\frac{1}{n}\right)$, while for PERT the influence area is $O\left(\frac{1}{n^2}\right)$. An intuitive explanation is that the average distance in one coordinate is $\left(\frac{1}{n}\right)$. PERT is fitting curves of the form $f(x, y) = axy + bx + cy + d$. Therefore, one outlier controls an area of $O\left(\frac{1}{n^2}\right)$. Now if we can fit a smoothing curve that is quadratic in both $x$ and $y$, then we may be able to further squeeze the influence area of one outlier. The PERTQ algorithm uses this intuition. Instead of manually fitting a bi-quadratic function, we simply add squared features to the original features, and then fit PERT on this new set of features. This is like fitting a smoothing polynomial surface that has higher degree.
The idea of using extra features has been discussed by several other authors. Vapnik [1995] developed the Support Vector Machine (SVM) to use powers of the original features. Amit and Geman [1997] used random linear combinations of the original features to help build trees and vote them. Ho [1998] also suggested using extra features so more random trees can be obtained in her algorithm. Our situation provides a new motivation of this idea.

4.4.3 A Simulation Comparison of PERT, PERT2, and PERTQ

We apply PERT and its two revised versions of PERT to four synthetic data distributions (twonorm, threenorm, ringnorm, and waveform), which are also used by Breiman [1998a] and Cutler [1999]. PERT does not work very well for ringnorm. The two revised versions PERT2 and PERTQ perform very well, better than boosted tree methods. However, the performance of PERTQ on the easier task is inferior. Further investigations on this behavior should be carried out in the future.

To make this dissertation self-contained, the definitions of the four data distributions are given below (from Breiman [1998a] and Breiman et al. [1984]).

\textit{Twonorm}: This is 20-dimensional, 2-class data. Each class is drawn with equal probability from a multivariate Normal distribution with unit covariance. Class one has mean \((a, a, \ldots, a)\) and class 2 has mean \((-a, -a, \ldots, -a); a = \frac{2}{\sqrt{20}}.\)

\textit{Threenorm}: This is 20-dimensional, 2-class data. Class one is drawn with equal probability from a unit multivariate normal with mean \((a, a, \ldots, a)\) and from a unit multivariate normal with mean \((-a, -a, \ldots, -a).\) Class 2 is drawn from a unit multivariate normal with mean \((a, -a, a, -a, \ldots, a, -a); a = \frac{2}{\sqrt{20}}.\)

\textit{Ringnorm}: This is 20-dimensional, 2-class data. Class 1 is multivariate normal with mean zero and covariance matrix 4 times the identity. Class 2 has unit covariance but with mean \((a, a, \ldots, a); a = \frac{1}{\sqrt{20}}.\)

\textit{Waveform}: This is 21-dimensional, 3-class data. Each class is generated by random
Figure 4.6. Three fundamental waveforms for the waveform data

combinations of two of the three fundamental waveforms $h_1(t)$, $h_2(t)$ and $h_3(t)$. All three waveforms are defined on the set \{1, 2, 3, \ldots, 21\}. and image vectors for three mappings are:

- $h_1 = (0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$.
- $h_2 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0)$.
- $h_3 = (0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0)$.

Figure 4.6 gives the graphs of the three fundamental waveforms.

Now one single observation for waveform data is generated as follows: Generate a unit uniform data $U$, and a 21-dimensional unit multivariate normal with mean zero. Let $(X, Y)$ be the feature and class label, then

If ($U < \frac{1}{3}$), it’s a class 0: $Y = 0, X = Uh_1 + (1 - U)h_2 + \epsilon_1$,

if ($U > \frac{2}{3}$), it’s a class 1: $Y = 1, X = Uh_1 + (1 - U)h_3 + \epsilon_2$,

otherwise it is a class 2: $Y = 2, X = Uh_2 + (1 - U)h_3 + \epsilon_3$.

The setup for the simulation is exactly same as used by Cutler [1999]. Each time a training set of size 300 is used, and the decision rules are tested on an independently drawn
testset of size 3000. One hundred repetitions are carried out for each data distribution and each version of PERT. The averages of the testset errors are reported.

For PERT and its two variants, we vote 101 component random trees to give final decision rules. PERT2 involves a parameter $\lambda$. We actually adopt a simpler version where we stop splitting trees when the number of examples in one node is less than 11. We give the simulation results in Table 4.3. The results for bagged and boosted CART (from Breiman [1998a]) are also included for comparison purposes. For PERT and its two variants, the numbers in parenthesis are the standard deviation of the error estimation.

Table 4.3. Comparison of Two Variants of PERT and Other Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>PERT</th>
<th>PERT2</th>
<th>PERTQ</th>
<th>Bagged CART</th>
<th>Boosted CART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twonorm</td>
<td>3.6(0.3)</td>
<td>3.6(0.6)</td>
<td>5.2(0.7)</td>
<td>7.4</td>
<td>4.8</td>
</tr>
<tr>
<td>Threenorm</td>
<td>17.6(1.1)</td>
<td>17.4(1.8)</td>
<td>24.8(2.3)</td>
<td>20.4</td>
<td>18.8</td>
</tr>
<tr>
<td>Ringnorm</td>
<td>13.2(3.6)</td>
<td>9.3(3.0)</td>
<td>7.4(1.8)</td>
<td>11.0</td>
<td>6.9</td>
</tr>
<tr>
<td>Waveform</td>
<td>17.9(1.1)</td>
<td>18.4(1.7)</td>
<td>18.6(1.3)</td>
<td>19.9</td>
<td>17.8</td>
</tr>
</tbody>
</table>

For ringnorm data, both of the variants give significant improvement. PERT does not work well for ringnorm data. This may be because the boundary of ringnorm is complicated. PERTQ fits quadratic surfaces. It works best for ringnorm. PERT2 works by voting local nodes. This also works well for ringnorm. For twonorm, threenorm, and waveform, PERT2 works as well as PERT. Notice for our simulation here, the parameter $k = 11$. Better performance should be possible if we search for the best $k$ values using cross-validation. PERTQ does not work as well as PERT for these three data distributions. The mechanic that PERTQ does not work well seems to be more complicated than we have expected.
CHAPTER 5

ROBUSTNESS AND EFFICIENCY SIMULATIONS

This chapter contains two sections. The first section is devoted to discussing the robustness of PERT. It is compared with bagged and boosted CART. The squeezing effect (section 4.3) is also discussed for boosted CART. In the second section, we design simulations to compare the efficiency of different methods.

5.1 Robustness Studies for Boosted Methods

The robustness properties of boosting methods are topics of much recent research. It has been widely accepted that ensemble methods such as bagging and boosting generally improve the performance of a base classifier. It is also pointed out that boosting usually works better than bagging, especially for big datasets (Breiman [1998a]). However, there are cases when boosting a base classifier can work worse than the base classifier itself. It is suspected (Quinlan [1996]) that there are outliers in those datasets. Since boosting methods will finally classify everything in training data correctly, they will suffer extra error around those outliers. This causes boosting methods to make more errors while predicting testset samples. In contrast, bagging methods do not “overfit” training data, they will not suffer from this kind of extra error, and they rarely work worse than one single classifier. In this sense, bagging methods are more robust than boosting methods. This is confirmed by Dietterich [1998b], Breiman [1999a], and Bauer and Kohavi [1999]. It is empirically shown by these authors that when there are outliers or heavy output noise in the training data, the performance of boosting methods will deteriorate greatly, while iid ensemble methods such as bagging or randomized tree ensembles drop only a little in performance.

In this section, we first consider the boosting behavior in terms of the “squeezing effect.” We compare the effect of outliers for boosting methods with that of the NN classifier. Then
we use simulation to study the robustness of PERT. The behavior of PERT is then compared to bagged and boosted CART on several synthetic distributions as well as three real datasets.

5.1.1 The Relationship of $D_B$ and $E_B$ for Boosted CART

Recall that we know that boosting methods will finally classify all training samples correctly (theoretically and empirically shown by Schapire et al. [1998]); therefore, boosted c4.5 is an oversensitive method. It will suffer the extra errors represented by Formula 4.11. Notice the squeezing effect can be used to describe the influence of outliers or those "bad" training samples whose classes are not consistent with what Bayes rule predicts them to be.

As for PERT case, we model the relationship of $D_B$ and $E_B$ for boosting methods by

\[ D_B = O(E_B^r), \]

and we need to estimate $r$.

We use the same distribution (Example 4.2.1) to carry out the simulation. Recall that Example 4.2.1 is a mixture of two multivariate uniform distributions, with varying priors. As carried out for PERT, the prior $\pi_0$ for class "0" (which is also the Bayes error for this data distribution) are chosen to change from $0.01, 0.02, \ldots, 0.39, 0.40$. For each of these priors, a sample of size 2000 is drawn, and boosted CART is applied to it. Bayes deviance $D_B$ is estimated by an independent testset from the same distribution. This is done 20 times. The average of the Bayes deviance, together with those of the NN and PERT, is plotted against the Bayes error. Figure 5.1 and 5.2 are figures for 2-dimensional and 8-dimensional feature spaces.

Both graphs indicate that power equation 5.1 is a reasonable fit. We fit a nonlinear function $D_B = kE_B^r$ to estimate $r$ values. For dimension=2, the estimates of $r$ for boosted CART is 1.54, less than the estimate of $r = 1.99$ for PERT. A similar simulation is also carried out for dimension=8 case. Again, the estimated value for PERT is higher, with $r_{pert} = 9.3$, and $r_{BoostedCART} = 3.0$. This simulation indicates that if some outliers are
Figure 5.1. The relationship between Bayes deviance $D_B$ and Bayes error $E_B$ for NN, PERT, and boosted CART dimension=2

Figure 5.2. The relationship between Bayes deviance $D_B$ and Bayes error $E_B$ for NN, PERT, and boosted CART dimension=8
randomly put into an otherwise pure distribution, then their influence for PERT will be smaller than that for the boosted CART, and they will have greatest influence for the nearest neighbor classifier due to the linear relationship between $D_B$ and $E_B$. Therefore, in this ideal situation, PERT is more resistant than boosted CART. The next section shows that this is also true for general data distributions.

5.1.2 Robustness or Resistance to Outliers

Outliers are very common in statistical datasets. So it is important that a classification method should not change much if some parts of the data, either in the feature space or the class labels, are changed. In classification, a common concern is mislabeling of the training examples. It is not that the class labels are more apt to errors, but that errors in class labels usually generate more severe problems in prediction accuracy. Also when the Bayes error is nonzero, the feature space of two classes may be overlapping. As a result of this, one point in a feature space can be from one class or another, with the probability $p(i|x)$ of being from class “i”. This is so-called output noise. When a classification method is applied to such training data, the effect of the “bad” points works just like outliers, as those class labels are not the same as the Bayes rule predicts them to be. We will to study the robustness of PERT, boosted and bagged CART for these kinds of situations.

In order to have a better understanding of what we observed, we use synthetic data distributions as well as some of the real datasets. All three data distribution, all of which are 2-class problems, are described below:

1. High-Dim XOR: This is extended version of the continuous XOR data. All Coordinates are Unif(-1,1), and independent from each other. Class is “0” if the first two coordinates have different signs, and “0” otherwise.

2. Diagonal: Features are uniform in the hypercube $(0, 1)^d$. Class is “0” if the sum of the coordinates is less than $\frac{d}{2}$. Otherwise, class is “1”.

3. Spherical: Features are multivariate normal with mean 0 and identity covariance.
matrix. The class boundary is a sphere. The radius of the sphere is chosen so that
the two classes have the same priors. Class “0” are those points inside the sphere.
Class “1” are those outside the sphere. (this distribution is a special case of the
example used by Friedman et al. [1998]).

In addition to the synthetic data distributions, three real data sets are also used in this
simulation. They are: breast cancer, diabetes, and the ionosphere. All of the three data are
2-class problems, with feature spaces of 8, 9, and 33 dimensions, respectively. The diabetes
and the breast cancer are cleaned up data used in Ripley [1996], and their sample sizes
are 683 and 532. The ionosphere data has size 351. One variable is deleted because it is
constant for all observations.

Experimental setup: according to Breiman [1999a], the outlier proportion is set to be
5%, i.e., we randomly switch 5% of the class labels. The training size is fixed to be 2000,
and test sets of 4000 are used. For the three synthetic data distributions, the dimensions
are fixed to be 10. For twonorm, threenorm, and ringnorm, the training set size is set to
be 300. For PERT, we vote 101 component random trees. For arcing and bagging, the
iteration numbers are fixed to be 51. For real datasets, there is no test set. We randomly
leave out 10% of the dataset as a testset, and PERT, bagged CART, and boosted CART
are applied to the remaining 90%.

Specification for CART: We use the Gini impurity (section 1.2) to split the trees. For
boosting, we build CART with Gini impurity of 0.10. A very small impurity may make
boosting exit the loop (Adaboost algorithm, section 1.3) and stops early. This may affect the
performance of boosting. On the other hand, bagging needs a more flexible base classifier.
So we choose impurity to be 0.02. By choosing impurity different, we may obtain the best
possible performance for each method.

For each data set, we run both the case with 5% output noise and the case of no output
noise. The testset error reported are the average of 100 runs for each case.

Table 5.1 shows that both PERT and boosted CART deteriorate when outliers are
present. But the performance of bagging shows no significant change. We calculate the percentage of increase in error rate for all three methods. The results are in Table 5.2. With 5% outliers, the performance of boosted CART drops about 37% (High Dim XOR excluded). While for PERT, it is about 14%. For bagging, no significant change in performance is observed, except for the High-dim XOR data. But high-dim XOR is the kind of data for which CART can easily separate the classes, and with 5% of outliers, CART can no longer separate very well. The error increases a little. But the percentage will be big. PERT does not has such a big increase in error. But it does not work as well as boosted CART or bagged CART.

Table 5.1. Error Rates With and Without 5% Outliers.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PERT</th>
<th>Boosted CART</th>
<th>Bagged CART</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
<td>5% outliers</td>
<td>0%</td>
</tr>
<tr>
<td>Twonorm</td>
<td>3.6</td>
<td>4.4</td>
<td>4.6</td>
</tr>
<tr>
<td>Threenorm</td>
<td>17.6</td>
<td>20.3</td>
<td>18.5</td>
</tr>
<tr>
<td>Ringnorm</td>
<td>13.2</td>
<td>15.0</td>
<td>6.7</td>
</tr>
<tr>
<td>Sphere</td>
<td>21.7</td>
<td>23.0</td>
<td>9.8</td>
</tr>
<tr>
<td>Diagonal</td>
<td>8.4</td>
<td>9.3</td>
<td>7.6</td>
</tr>
<tr>
<td>High-dim XOR</td>
<td>10.7</td>
<td>12.8</td>
<td>0.2</td>
</tr>
<tr>
<td>Breast cancer</td>
<td>2.7</td>
<td>3.4</td>
<td>3.0</td>
</tr>
<tr>
<td>Diabetes</td>
<td>24.0</td>
<td>24.6</td>
<td>24.6</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>8.8</td>
<td>9.6</td>
<td>6.3</td>
</tr>
</tbody>
</table>

Table 5.2. Percentage of Error Increase When 5% Outliers Are Added

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PERT</th>
<th>boosted CART</th>
<th>bagged CART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twonorm</td>
<td>22.2</td>
<td>37.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Threenorm</td>
<td>15.3</td>
<td>8.6</td>
<td>3.4</td>
</tr>
<tr>
<td>Ringnorm</td>
<td>13.6</td>
<td>44.8</td>
<td>0.0</td>
</tr>
<tr>
<td>Sphere</td>
<td>6.0</td>
<td>51.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Diagonal</td>
<td>10.7</td>
<td>52.6</td>
<td>-2.4</td>
</tr>
<tr>
<td>High-dim XOR</td>
<td>19.6</td>
<td>650.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Breast cancer</td>
<td>26.0</td>
<td>70.0</td>
<td>-2.9</td>
</tr>
<tr>
<td>Diabetes</td>
<td>6.1</td>
<td>6.1</td>
<td>0.0</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>9.1</td>
<td>28.6</td>
<td>4.8</td>
</tr>
</tbody>
</table>
This simulation shows that boosted CART is the best choice when there are no outliers. PERT can work very well for many cases, but it may not work well for some synthetic data distribution such as Ringnorm, High-dim XOR, and Diagonal. Bagging is the best choice if there are outliers in the training data. However, PERT can be very fast (Cutler [1999]), and many real data are simple. It may be the best method to use when time efficiency is considered.

5.2 Efficiency Studies for PERT and Other Methods

This section is devoted to the efficiency studies for PERT and boosted CART. Both are compared to the NN classifier, whose geometric properties lend itself for a quite straightforward understanding. After a new criterion to measure efficiency is defined, several simulations are carried out to estimate the efficiency factors.

5.2.1 Efficiency Criterion

The generalization error is a commonly used measure to determine how well a classification method works. As pointed out in section 3.2, for the case of nonzero Bayes error, the generalization error will not go to 0 as the training size $n$ goes to $\infty$. This makes the discussion of convergence rate impossible. In this section we use Bayes deviance to compare different methods. We also define a relative efficiency measure based on the NN method.

The NN method is a very common method. For any point $x$, the prediction of the NN classifier is only related to its nearest neighbor in the training data. It is a highly localized model fitting method. When the training data are uniformly distributed, each single point controls an expected probability volume of $\frac{1}{n}$, regardless of the class label of the point. We define the relative efficiency of a classification method as the ratio of the Bayes deviance of the NN classifier to that of the method under consideration.

Friedman [1997] models the misclassification error by:

$$e_M((d, n)) = c(d) \times n^{-\frac{1}{r(d)}}$$
where \( r(d) \) is called "the convergence factor." It depends only on the classification method and the dimension \( d \) of the problem under consideration. Notice that the right-hand side of equation 5.2 can be rewritten as \( c(d) \frac{1}{\sqrt{n}} \). This means a small value of \( r \) implies faster convergence.

Since this definition will not be applicable for nonzero Bayes error, we use the Bayes deviance instead, i.e.,

\[
d_B(d, n) = c(d) \times n^{-\frac{1}{r(d)}}.
\]

Notice estimating \( d_B \) is not a straightforward problem. In this dissertation, we only use it to compare different methods on data distributions for which the Bayes rule can be calculated. In this case \( d_B \) can be estimated by its sample version

\[
\hat{d}_B(d, n) = \frac{\# \{ x_i | r(x_i) \neq C_B(x_i) \} }{N},
\]

where \( x_i \) are the test samples, and \( N \) is the total number of test samples.

The relative efficiency given below has a more intuitive interpretation. It indicates how good the classification method is compared to the NN method.

**Definition 5.2.1** (Relative efficiency of method \( M \))

\[
\text{Eff}(M, n, d)_{R} = \frac{D_B(NN, n)}{D_B(M, n)}.
\]

### 5.2.2 Simulations and Results

The goal for this simulation is to find out how the convergence factor \( r \) changes with respect to the dimensions of the problem. We choose to run simulation on the diagonal data defined in previous section. We compare PERT, and boosted CART. The setup for the simulation is as follows.

The dimension \( d \) is chosen to be 2, 4, 8, and 16. For each dimension, training sets of size \( n = 100, 200, 400, 800, 1600, 3200, 6400, \) and 12800 are used. Testset errors are obtained by using an independently drawn testset of size of 4000. This is carried out 20 times for each
dimension and training size. For each of $d$, the log-transformed Bayes deviance is fitted to the log-transformed sample size. Now denoting the slope of the linear fit by $\hat{m}(d)$, then estimated convergence factor is $\frac{1}{\hat{m}(d)}$.

First of all, the log-transformed Bayes deviance versus log-transformed sample size are plotted for two dimensions $d = 4$ and $d = 8$ (Figure 5.3). This near-linear relationship validates the appropriateness of the use of equation 5.3 to model the Bayes deviance. Notice this is also consistent with what is observed by Friedman [1997] for the k-NN method.

Table 5.3 gives the convergence factor lists the convergence factor obtained by the above procedure. It is very clear that for low dimension, there is not much difference between the three methods. However, when the dimension is greater than 4, the convergence factor of the NN method increase faster than other two methods. This indicates a slower convergence. Surprisingly, the convergence factors of PERT and boosted CART are very close. This indicates that PERT and CART are equally efficient in terms of an increase in sample size. However, the constant term $c(d)$ in formula 5.2 might be different.

Tables 5.4 and 5.5 give the relative efficiency of PERT and boosted CART against the NN method for different dimensions and sample sizes. For the 2-dimensional case, PERT works almost identically as NN. Boosted CART does not work since it consistently exits from the boosting loop (Adaboost algorithm in section 1.3). The early exit makes boosting not work well. For higher dimensions, boosted CART does not have this problem. And it works consistently better than NN. PERT also works better NN, but it does not work as well as boosted CART. Its relative efficiency is smaller than those of boosted

Table 5.3. Convergence Rate for Different Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>Dimension $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 4 8 16</td>
</tr>
<tr>
<td>NN</td>
<td>2.1 3.7 6.4 11.7</td>
</tr>
<tr>
<td>PERT</td>
<td>1.9 3.3 4.3 4.9</td>
</tr>
<tr>
<td>boosted CART</td>
<td>2.0 3.3 4.4 4.9</td>
</tr>
</tbody>
</table>
Figure 5.3. The log-transformed Bayes deviance versus log-sample size, for NN, PERT, and boosted CART dimension=4 and 8
Table 5.4. Relative Efficiency for PERT

<table>
<thead>
<tr>
<th>d</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
<th>6400</th>
<th>12800</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.94</td>
<td>0.95</td>
<td>0.97</td>
<td>1.02</td>
<td>1.02</td>
<td>0.99</td>
<td>1.03</td>
<td>1.08</td>
</tr>
<tr>
<td>4</td>
<td>1.12</td>
<td>1.15</td>
<td>1.20</td>
<td>1.23</td>
<td>1.26</td>
<td>1.30</td>
<td>1.28</td>
<td>1.32</td>
</tr>
<tr>
<td>8</td>
<td>1.33</td>
<td>1.49</td>
<td>1.63</td>
<td>1.74</td>
<td>1.83</td>
<td>1.91</td>
<td>1.96</td>
<td>2.02</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.50</td>
<td>1.67</td>
<td>1.82</td>
<td>2.00</td>
<td>2.12</td>
<td>2.20</td>
<td>2.31</td>
</tr>
</tbody>
</table>

Table 5.5. Relative Efficiency for Boosted CART

<table>
<thead>
<tr>
<th>d</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
<th>6400</th>
<th>12800</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.64</td>
<td>0.63</td>
<td>0.64</td>
<td>0.63</td>
<td>0.63</td>
<td>0.61</td>
<td>0.69</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>1.30</td>
<td>1.35</td>
<td>1.41</td>
<td>1.46</td>
<td>1.53</td>
<td>1.52</td>
<td>1.53</td>
<td>1.56</td>
</tr>
<tr>
<td>8</td>
<td>1.77</td>
<td>2.02</td>
<td>2.13</td>
<td>2.28</td>
<td>2.43</td>
<td>2.48</td>
<td>2.53</td>
<td>2.59</td>
</tr>
<tr>
<td>16</td>
<td>1.69</td>
<td>2.04</td>
<td>2.32</td>
<td>2.58</td>
<td>2.73</td>
<td>2.94</td>
<td>3.04</td>
<td>3.13</td>
</tr>
</tbody>
</table>

Note: The * indicates a missing value. My boosting code fails for this case.

CART at corresponding dimension and sample size. The difference seems to increase as the dimensionality increases.

The simulations presented in this section attempt to quantify the convergence rate for classification methods. If a classification method is Bayes consistent, then the Bayes deviance tends to 0 as the sample size increases to ∞. Therefore, we can estimate such a convergence factor for any consistent method. Even though a classifier is not strong consistent, Bayes deviance will still make sense as it is an increasing function of the generalization error. Further application of this efficiency idea should be possible, but it is beyond the scope of this dissertation.
CHAPTER 6

CONCLUSIONS AND FUTURE STUDY

We proposed a new framework for understanding the voting/aggregating method. We proved several theorems in this framework. These theorems help clarify the confusion about concepts in classification, such as the bias and variance. We developed the idea of an unbiased classifier and weak consistency. Robustness and efficiency were defined in terms of Bayes deviance in this framework, and studied by simulation. We also investigated the behavior of PERT and other kinds of "oversensitive" methods.

We list the work that is not finished at this time and also include possibilities for future study below:

1. Further simulation studies for efficiency studies. Apply to more classification methods, such as neural nets, logistic regression classifier, etc.

2. Develop better theoretical results for influence area. Is that as great as influence function for robustness study?

3. Strong consistency of PERT for the case of 0 Bayes error.

4. An investigation on when the iBoost algorithm (partition the data and combine the rules) will do better than the usual one-shot methods (use the data once and whole).

5. Find some applications where iBoost can really work better.
REFERENCES


CURRICULUM VITAE

GUOHUA ZHAO

Contact Information

Department of Mathematics and Statistics
Utah State University
Logan, UT 84322-3900
Fax: (435) 797-1822
Phone: (435) 797-2814
email: slzhao@math.usu.edu

Education Background

• Ph.D. (expected Dec. 2000), Statistics, Utah State University.
  Dissertation advisor: Dr. Adelle Cutler.
  Dissertation title: "A New Perspective On Classification".
• M.S. (1994), Mathematics, NanKai University (Tianjin, China).
• B.S. (1991), Mathematics, NanKai University (Tianjin, China).

Positions Held

• 1996-2000, teaching assistant, Utah State University, Logan, Utah
• 1994-1995, lecturer, Hangzhou University, Hangzhou, China.