INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

BOOTSTRAP TECHNIQUES FOR STATISTICAL PATTERN RECOGNITION

by

QUN WANG, M. Sc.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of Master of Science
Information and System Science
School of Computer Science

Carleton University

Ottawa, Ontario

April 2000

© Copyright

2000, Qun Wang

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52407-8

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Bootstrap Techniques for Statistical Pattern Recognition

submitted by

Qun Wang, M. Sc.

in partial fulfillment of the requirements

for the degree of Master of Science

Information and System Science

Director, School of Computer Science

Thesis Supervisor

CARLETON UNIVERSITY

APRIL 2000

ABSTRACT

This thesis studies the application of the bootstrap techniques to three fields of statistical pattern recognition: the estimation of the Bhattacharyya bound, the estimation of the error rate of a classifier and the classifier design. To initiate the study, the bootstrap technique, and its concepts, algorithms and properties are explained.

In the application of the bootstrap to estimating the Bhattacharyya bound, it is first argued, both theoretically and experimentally that the estimate of the Bhattacharyya bound is usually biased. Two types of the bootstrap estimators of the Bhattacharyya bound, the direct and indirect bias correction estimators, are suggested. It is proved experimentally that the suggested bootstrap estimators of the Bhattacharyya bound are superior to the general estimator.

In the application of the bootstrap to the error rate estimation, the previous works are reviewed, and the related algorithms are illustrated with the simulation results. These algorithms are catalogued against the real-sample algorithm, as they use only the original training samples to estimate the error rate. The pseudo-sample algorithms of the error rate estimation are then proposed. These use the bootstrap technique to generate pseudo-samples for estimating the error rate. Comparisons based on the experiments prove that the pseudo-sample algorithms perform better than most of the other algorithms.

In the application of the bootstrap to classifier design, the mixed-sample algorithm is introduced. It overcomes the limitation of the training sample size by adding

pseudo-samples to the original training samples for classifier design. The experimental results prove that this strategy is successful.

ACKNOWLEDGMENTS

First of all, I would like to thank my Professor, Dr. John Oommen, for his continued support, encouragement, and for believing in me. It is his guidance, ideas, and wisdom that led me to explore new scientific domains, and to discover new research horizons. I am deeply grateful to him for all that he has done for me.

I would like to thank Ms. Patricia King-Edwards who helped me by proofreading and editing most of my thesis. Above all, I would like to thank my wife, Ying Huang for her help, support, and encouragement. This thesis would never be done if it was not for her support.

DEDICATION

To my parents.

TABLE OF CONTENTS

ABSTRACT.....	III
ACKNOWLEDGMENTS.....	V
DEDICATION	VI
TABLE OF CONTENTS	VII
LIST OF TABLES.....	XI
LIST OF FIGURES	XIV
CHAPTER 1: INTRODUCTION.....	1
1.1 Introduction.....	1
1.2 General Contributions.....	4
1.3 Outline of the Thesis.....	5
1.4 Experiment Data Set	6
CHAPTER 2: THE BOOTSTRAP TECHNIQUE.....	11
2.1 Introduction.....	11
2.2 The Fundamentals of Bootstrap.....	13
2.3 Bootstrap Estimators	17
2.3.1 Bias	17

2.3.2 Variance	18
2.3.3 Quantile.....	20
2.3.4 Parametric Bootstrap.....	21
2.3.5 Density Function.....	21
2.4 Properties of Bootstrap	25
2.5 Other Resampling Approaches	29
2.5.1 Bayesian Bootstrap	30
2.5.2 Random Weighting Method.....	34
CHAPTER 3: THE BHATTACHARYYA BOUND	39
3.1 Introduction.....	39
3.2 Chernoff and Bhattacharyya Bounds.....	42
3.2.1 Chernoff Bound	42
3.2.2 Bhattacharyya Bound.....	43
3.3 The Estimation of Bhattacharyya Bound.....	44
3.3.1 Simulation Results	47
3.4 The Bootstrap Estimators - Direct Bias Correction	49
3.4.1 Simulation Results	57
3.5 The Bootstrap Estimators - Distance Bias Adjustment	59
3.5.1 AFMS (Adjust First and Mean Second) Schemes	61
3.5.2 MFAS (Mean First and Adjust Second) Schemes	66
3.6 Summary	71

CHAPTER 4: BOOTSTRAP BASED ERROR ESTIMATORS	73
4.1 Introduction.....	73
4.2 Algorithms of the Error Rate Estimating.....	75
4.2.1 Apparent Error	76
4.2.2 Cross-validation (Leave-one-out)	78
4.2.3 Basic Bootstrap	79
4.2.4 E0 Estimator.....	84
4.2.5 Leave-one-out Bootstrap.....	86
4.2.6 0.632 Estimators	89
4.3 Simulation Results	90
4.4 Pseudo-Sample Algorithms	92
4.4.1 Pseudo-Testing Algorithm	92
4.4.2 Pseudo-Classifer Algorithm.....	99
4.4.3 Leave-one-out Pseudo-Classifer Algorithm	101
4.5 Summary	105
 CHAPTER 5: BOOTSTRAP BASED CLASSIFIER DESIGN	 110
5.1 Introduction.....	110
5.2 Previous Work	111
5.3 Mixed-sample Classifier Design.....	116
5.4 Simulation Results	118
5.5 Discussions and Conclusions.....	124

CHAPTER 6: SUMMARY AND DISCUSSION.....	126
6.1 Introduction.....	126
6.2 Bias Correction	126
6.3 Cross-validation	129
6.4 Pseudo-pattern Generation.....	131
6.5 Conclusion	133
REFERENCES.....	136

LIST OF TABLES

TABLE 1.1 Expectations of the classes in Data Set I.....	8
TABLE 3.1 Theoretical values of Bhattacharyya distance and bound	47
TABLE 3.2 Estimates of Bhattacharyya distances with the General approach.....	47
TABLE 3.3 Estimates of Bhattacharyya bounds with the General approach(%)	47
TABLE 3.4 Estimates of Bhattacharyya bounds with the Basic Bootstrap(%)	58
TABLE 3.5 Estimates of Bhattacharyya bounds with the Basyesian Bootstra[(%)	58
TABLE 3.6 Estimates of Bhattacharyya bounds with the Random Weighting Method(%)	58
TABLE 3.7 Estimates of Bhattacharyya bounds(%) Basic Bootstrap, B-based AFMS..	63
TABLE 3.8 Estimates of Bhattacharyya bounds(%) Bayesian Bootstrap, B-based AFMS	63
TABLE 3.9 Estimates of Bhattacharyya bounds(%) Random Weighting Method, B- based AFMS	64
TABLE 3.10 Estimates of Bhattacharyya bounds(%) Basic Bootstrap, G-based AFMS	65
TABLE 3.11 Estimates of Bhattacharyya bounds(%) Bayesian Bootsrap, G-based AFMS	65
TABLE 3.12 Estimates of Bhattacharyya bounds(%) Random Weighting Method, G- based AFMS	66

TABLE 3.13 Estimates of Bhattacharyya bounds(%) Basic Bootstrap, B-based MFAS	68
TABLE 3.14 Estimates of Bhattacharyya bounds(%) Bayesian Bootstrap, B-based MFAS	69
TABLE 3.15 Estimates of Bhattacharyya bounds(%) Random Weighting Method, B-based MFAS	69
TABLE 3.16 Estimates of Bhattacharyya bounds(%) Basic Bootstrap, G-based MFAS	70
TABLE 3.17 Estimates of Bhattacharyya bounds(%) Bayesian Bootstrap, G-based MFAS	70
TABLE 3.18 Estimates of Bhattacharyya bounds(%) Random Weighting Method, G-based MFAS	70
TABLE 4.1 Error Rate Estimates of 3-NN classifiers (sample size 8)	91
TABLE 4.2 Error Rate Estimates of 3-NN classifiers (sample size 24)	91
TABLE 4.3 Error Rate Estimates of the Pseudo-Testing algorithm	98
TABLE 4.4 Error Rate Estimates of the Pseudo-Classifer algorithm (sample size 8) .	100
TABLE 4.5 Error Rate Estimates of the Pseudo- Classifier algorithm (sample size 24)	100
TABLE 4.6 Error Rate Estimates of the Leave-One-Out Pseudo-Classifer algorithm (sample size 8)	100
TABLE 4.7 Error Rate Estimates of the Leave-One-Out Pseudo- Classifier algorithm (sample size 24)	100

TABLE 4.8 Absolute difference between estimates and true errors (sample size 8).....	107
TABLE 4.9 Absolute difference between estimates and true errors (sample size 24)...	108
TABLE 4.10 Performance ranges according to the absolute difference from the true error (sample size 8).	109
TABLE 4.11 Performance ranges according to the absolute difference from the true error (sample size 24).	109

LIST OF FIGURES

Figure 1.1	Expectation Distribution of the Classes in Data Set I.	8
Figure 5.1	Testing Error of Mixed-Sample Classifier - Classe Pair (A, B).....	119
Figure 5.2	Testing Error of Mixed-Sample Classifier - Classe Pair (A, C).....	119
Figure 5.3	Testing Error of Mixed-Sample Classifier - Classe Pair (A, D).....	120
Figure 5.4	Testing Error of Mixed-Sample Classifier - Classe Pair (A, E).....	120
Figure 5.5	Testing Error of Mixed-Sample Classifier - Classe Pair (A, F)	121
Figure 5.6	Testing Error of Mixed-Sample Classifier - Classe Pair (A, G).....	121

CHAPTER 1

INTRODUCTION

1.1 Introduction

Since Efron's bootstrap technique was introduced in the late 1970's [Ef79], it has attracted great interest from both the theoretical and applied sides of statistics. Since it uses the simulation strategy to calculate estimates, the bootstrap technique is able to retrieve more information from sample data, and to solve problems that are not easily solved by the traditional methods: such as the bias, the variance and the parameters of the distribution of an estimator. Over the past two decades, great effort has been made to develop the theory and application of the bootstrap ([DH97], [Ma92] and [ST95]). In rapid succession, large numbers of problems have proved to be amenable to the new technique, which confirmed that the bootstrap is efficient, flexible and useful.

Efron's paper, "Estimating the error rate of a prediction rule: improvement on Cross-validation" [Ef83], may be the earliest work in applying the bootstrap technique to the error rate estimation. More endeavors, [CMN85], [CMN86], [Ef86], [Ha86], [JDC87], [Fu90], [We91], [DH92] and [ET97], have been made in this area since then. The other field in statistical pattern recognition, where the bootstrap technique has been applied to, is the classifier design [HUT97].

In this thesis, the bootstrap technique will be applied to three problems in statistical pattern recognition:

1. Estimation of the Bhattacharyya bound;

2. Estimation of the error rate of a classifier; and
3. Classifier design.

The Bhattacharyya bound is an upper bound of the error rate of a classifier and an exponential function of the Bhattacharyya distance. When the distributions of two classes are normal, the Bhattacharyya distance is a function of the first and second moments of the normal distributions. It has been proved, both theoretically and practically, that an estimate of the Bhattacharyya bound given by any traditional method is seriously biased. Applying the bootstrap technique, two types of new estimators of the Bhattacharyya bound are introduced, which are respectively called the direct and indirect bias correction estimators. The first estimator estimates the bound on the basis of directly correcting the bias of the Bhattacharyya bound, while the second estimates the bound on the basis of correcting the bias of the Bhattacharyya distance. The simulation results have proved that both of them successfully improve the estimates of the Bhattacharyya bound.

Error rate estimation is, of course, an active topic because it is the crucial measurement of a classifier. A lot of work has been done on the topic. It is generally agreed that the apparent error estimator is biased so as to underestimate the error, while the traditional leave-one-out estimator over estimates the apparent error estimator. New efforts have tried to improve the error estimator by introducing the bootstrap technique. Three bootstrap estimators were first introduced in Efron's paper [Ef83]. The basic bootstrap estimator uses the basic bootstrap technique to correct the estimate bias of the apparent error estimator. As opposed to them, the E0 estimator uses the training patterns excluded from the bootstrap samples to estimate the error rate. The 0.632 estimator is a

combination of the E0 and the apparent error estimators. Although, from then, a few other algorithms have been suggested, such as the MC and the convex bootstrap [CMN85], no essential progress was made until the algorithms of the leave-one-out bootstrap and 0.632+ were invented by Efron and Tibshirani [ET97]. The leave-one-out bootstrap is an algorithm combining the techniques of Cross-validation and bootstrap together, while the 0.632+ estimator is a combination of the leave-one-out bootstrap and the apparent error estimators.

As all of the error rate estimation algorithms mentioned above use only the original training samples, they are said to be the real sample algorithms. Motivated by the SIMDAT algorithm [TT92], the Bayesian bootstrap and random weighting method [ST95], pseudo-sample algorithms of the error rate estimation are introduced in this thesis. The basic concept of the pseudo-sample algorithms is to use the pseudo-samples generated by a bootstrap sampling technique to estimate the error rate. These pseudo-testing algorithms use the pseudo-samples as the testing samples, while the pseudo-classifier algorithms utilize the pseudo-samples to build the classifier, and use the original training samples as the testing samples. The experiments demonstrated that the pseudo-testing algorithm tends to under-estimate the error rate, while the pseudo-classifier algorithm performed very well. It can be seen that, combined with the Cross-validation technique, the pseudo-classifier algorithm is one of the best algorithms available to estimate the error rate.

The problem of classifier design is different from the estimation problems. The purpose of classifier design is to seek a better discriminant function. Little work has been

done in this area to incorporate the bootstrap technique. The algorithms for classifier design introduced by Hamamoto and his partners [HUT97] focus on creating artificial training samples instead of the original training samples so as to construct a classifier. Their main strategy uses the local mean to replace the original training pattern when constructing a classifier. Their experiments proved that this strategy works for the k-NN classifier.

In this thesis, the mixed-sample algorithms for classifier design will be proposed. The main idea of the algorithms is to enlarge the training sample for constructing the classifier by adding pseudo-training patterns into the original training sample set. The simulation results proved that this strategy works well when the proper bootstrap sampling schemes are used to generate the pseudo-training patterns.

1.2 General Contributions

The general contributions of this thesis are distributed into three areas:

- The Bhattacharyya bound estimation which uses:
 - a) Direct bias correction algorithm; and
 - b) Distance bias adjustment (indirect bias correction) algorithms.
- The error rate estimation incorporating:
 - a) A pseudo-testing algorithm;
 - b) A pseudo-classifier algorithm; and
 - c) A Leave-one-out pseudo-classifier algorithm.
- The classifier design which incorporates a Mixed-sample classifier design.

Variations of the above algorithms will also be discussed in this thesis. These include the modification of the resampling schemes (basic bootstrap, Bayesian bootstrap, random weighting method and the SIMDAT algorithm), the pseudo-sample size and the size of the nearest neighbors of a training pattern.

1.3 Thesis Outline

This thesis is composed of six chapters. Chapter 2 discusses the bootstrap technique. Section 2.1 is a general instruction to the bootstrap and its application to statistical pattern recognition. Section 2.2 describes the basic concept of Efron's bootstrap as well as a general algorithm of the basic bootstrap, followed by several examples of the bootstrap estimators that are the content of Section 2.3. Two theorems about the properties of the bootstrap are presented in Section 2.4. Section 2.5 illustrates the other two bootstrap sampling schemes – the Bayesian bootstrap and random weighting method.

Chapter 3 discusses the estimation problem of the Bhattacharyya Bound. The definitions and formulae of the Chernoff and Bhattacharyya bounds are presented in Section 3.2. Subsequently, Section 3.3 describes a traditional algorithm for estimating the Bhattacharyya bound, and provides the results of related experiments. Two kinds of the bootstrap estimators are discussed with the simulation results in Sections 3.4 and 3.5 respectively. Section 3.6 is a summary on the bootstrap-based Bhattacharyya bound estimation methods.

The bootstrap-based error estimators of a classifier are the topic of Chapter 4. Section 4.1 is a brief introduction of the prior work done regarding the error estimation of a classifier. The details of the main algorithms of the error estimators are represented in Section 4.2 followed by the results of related experiments that are illustrated in Section 4.3. Section 4.4 introduces three kinds of pseudo-sample algorithms of error estimation, and discusses the algorithms with the related simulation results. Section 4.5 summarizes all the algorithms discussed in this chapter.

The bootstrap-based classifier design is the content of Chapter 5. Section 5.1 states the problem of classifier design and discusses the problem to be solved. Previous works done on the bootstrap-based classifier design are discussed in Section 5.2. The algorithms of mixed-sample classifier design are introduced in Section 5.3, while the related results of simulations are discussed in Section 5.4. Section 5.5 provides the discussions and summarizes the work done in mixed-sample classifier design.

Chapter 6 summarizes the applications of the bootstrap technique to the field of statistical pattern recognition. The three main issues used in the thesis, bias correction, cross-validation and pseudo-pattern generation, are discussed in Sections 6.2, 6.3 and 6.4 respectively. Section 6.5 states the conclusions of our work.

1.4 Experiment Data Set

Throughout this paper, all the experiments have been carried out with a data set referred to as Data Set I, which contains 2-dimensional training samples of seven classes.

All the classes follow a normal distribution with the same covariance matrixes, the identity matrix, but they have different expectation vectors generated by **Algorithm 1.1**.

Algorithm 1.1 Expectation Vector Generating

Input: d , the dimension.
Output: **ExpectationList**, the expectation vectors.
Method
BEGIN
 For ($j = 1$ to d)
 $EX[j] = 0$
 End For
 Put **EX** in **ExpectationList**
 $\text{delta} = 0.5$
 Do
 $k = \text{random integer in } [1, d]$
 $EX[k] = EX[k] + \text{delta}$
 $\text{delta} = \text{delta} + 0.1$
 Put **EX** in **ExpectationList**
 While (**ModulusOfVector**(**EX**, d) < 3)
 Return **ExpectationList**
END Expectation Vector Generating

Procedure ModulusOfVector

Input: (i) $v[1], v[2], \dots, v[d]$, a vector;
(ii) d , the dimension;
Output: The Modulus of a vector.
Method
BEGIN
 $\text{modulus} = 0$
 For ($j = 1$ to d)
 $\text{modulus} = \text{modulus} + v[j] * v[j]$
 End For
 $\text{modulus} = \sqrt{\text{modulus}}$
 Return modulus
END Procedure ModulusOfVector

In **Algorithm 1.1**, the expectation vector of the first class is set to (0, 0). Then, in the **Do...While** loop, a delta, with an initial value of 0.5, is randomly added to one of the

elements of the previous expectation vector to yield the next expectation vector, and delta is then increased by 0.1. Expectation vectors for the classes are generated in this way until the modulus of the last expectation vector exceeds 3. With **Algorithm 1.1**, expectations of seven classes of training samples were generated. Their expectations are listed in TABLE 1.1, and Figure 1.1 depicts the distribution of the expectations.

TABLE 1.1 Expectations of the classes in Data Set I

CLASS	A	B	C	D
EXPECTATION	(0.0, 0.0)	(0.5, 0.0)	(1.1, 0.0)	(1.1, 0.7)
CLASS	E	F	G	
EXPECTATION	(1.1, 1.5)	(2.0, 1.5)	(3.0, 1.5)

Figure 1.1 Expectation Distribution of the Classes in Data Set I

Each sample point is a 2-dimensional vector with a normal distribution. The sample vectors were generated by **Algorithm 1.2**.

Algorithm 1.2 d -Dimension Normal Vector

Input: (i) d , the dimension;
(ii) $E[1], E[2], \dots, E[d]$, the expectation vector;
Output: A vector follows a normal distribution.
Method
BEGIN
 For ($k = 1$ to d)
 MltNormal [k] = **StandardNormal**
 MltNormal [k] = **MltNormal** [k] + $E[k]$
 End For
 Return **MltNormal** [1], ..., **MltNormal** [d]
END d -Dimension Normal Vector

Procedure StandardNormal

Input: nothing.
Output: An approximate standard normal distribution variable.
Method
BEGIN
 RandomNM = 0
 For ($j = 1$ to 12)
 RandomNM = RandomNM + Random Uniform [$1, 0$]
 End For
 RandomNM = RandomNM - 6.0
 Return RandomNM
END StandardNormal

Algorithm 1.2 is a typical algorithm for generating a d -dimension normal distributed random vector with the identity matrix as the covariance matrix. It calls the procedure **StandardNormal** d times to obtain d variables of approximate $N(0, 1)$. The procedure **StandardNormal** accumulates 12 variables of uniform distribution $U[0, 1]$ to return a variable which is approximately $N(0, 1)$. Since the expectation of the random normal vector required is generally not $\underline{0}$, a corresponding expectation value is added to each element of the d -dimension random vector returned by the procedure **StandardNormal**. In this way, a d -dimension normally distributed random vector is

generated with the given expectation vector and the identity matrix as the covariance matrix.

200 separate samples each of size 50 were generated for each class in Data Set I, and were used for the simulation. Another sample of size 1000 was also generated separately for each class, and was used for testing.

CHAPTER 2

THE BOOTSTRAP TECHNIQUE

2.1 Introduction

The bootstrap technique is one of the most popular statistical approaches today for estimating parameters and their distributions. It was first introduced by Efron in the late 1970's [Ef79]. The basic strategy of bootstrap is based on resampling and simulation. Over the last two decades, bootstrap has been applied to a wide range of statistical problems. Its theory has been extensively developed, and a variety of techniques related to bootstrap have been proposed for various application problems, including bias estimation, standard error estimation, confidence interval estimation, hypothesis testing, estimation of prediction error, sample survey, linear modeling, kernel density estimation, time series and dependent data (see [ET93], [DH97] and [ST95]).

Efron's work on Error Estimation [Ef83] can be regarded as the start of applying the bootstrap techniques in the area of statistical pattern recognition. Then Chernick [CMN86], Jain [JDC87] and Weiss [We91] performed simulations to compare bootstrap error estimates with other estimates, mainly with the Cross-validation estimation schemes. Efron [Ef86] and [ET97], Davison [DH92] and Fukunaga [Fu92] discussed also theoretical issues of the bootstrap error estimation. Detailed descriptions on the above works will be given in Chapters 3 and 4 respectively. There were also suggestions of

using the bootstrap techniques to construct pattern classifiers, by Hamamoto [HUT97], which will be discussed in Chapter 5.

In this chapter, the essential concept and theory of the bootstrap technique will be discussed. Some important bootstrap estimates useful in statistical pattern recognition and the properties of bootstrap will be given, and other related resampling plans will be introduced as well. In Section 2.2, the concept and basic procedure of bootstrap are discussed, and examples are given to illustrate the bootstrap procedure. Several bootstrap estimations, including bias, variance, quantile and density function, are discussed in Section 2.3,. These estimating procedures are very helpful in the understanding of Efron's bootstrap theory, and are also useful for estimating the error bound and the error rate of a classifier, which are the topics of the next two chapters. Section 2.4 discusses the theoretical aspects of the bootstrap, which are mainly concerned with the large sample properties, consistency and asymptotic distribution of the bootstrap estimations. The theorems in Section 2.4 concern the theoretical properties of the indices on invoking bootstrap. In the last section of the chapter, Section 2.5, we will discuss other resampling plans, the Bayesian bootstrap and the Random Weighting method, which were introduced by Rubin [ST95] and Zhen [Zh87] respectively. Both of them generalized Efron's bootstrap approach in some sense. Both of them will also be applied in later chapters to problems of statistical pattern recognition.

2.2 The Fundamentals of Bootstrap

Let $\mathbf{X} = \{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$ be an i.i.d. d -dimension sample from an unknown distribution F . We consider an arbitrary functional of F , $\theta = \theta(F)$, which for example, could be an expectation, a quantile, a variance, etc. The θ is estimated by a functional of the empirical distribution $\hat{F}$, $\hat{\theta} = \theta(\hat{F})$, where

$$\hat{F} : \text{mass } \frac{1}{n} \text{ at } \underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \dots, \underline{\mathbf{x}}_n,$$

where n is the sample size, $\{\underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \dots, \underline{\mathbf{x}}_n\}$ are the observed values of the sample $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$. A host of unanswered questions now arise such as:

- (i.) What is the bias of the estimation $\hat{\theta}$?
- (ii.) What is the variance of $\hat{\theta}$?
- (iii.) What is the distribution of $\hat{\theta}$?

These problems are generally not easy to answer, especially in the case when F is unknown. For example, the bias is well defined as

$$\text{Bias} = E_F [\hat{\theta} - \theta] = E_F [\theta(\hat{F}) - \theta(F)]. \quad (2.1)$$

where “ E_F ” indicates the expectation under distribution F . Though it is possible to calculate $\hat{\theta} = \theta(\hat{F})$ having observed $\underline{\mathbf{X}}_1 = \underline{\mathbf{x}}_1, \underline{\mathbf{X}}_2 = \underline{\mathbf{x}}_2, \dots, \underline{\mathbf{X}}_n = \underline{\mathbf{x}}_n$, it is not possible to derive the bias directly because of the fact that both θ and F are unknown. The question is thus one of determining how to get an estimator of the bias?

From one perspective, the quantity $\hat{\theta}$ can be regarded as a simulation of θ , since we use the empirical distribution $\hat{F}$ to mirror the characteristic of the unknown distribution F via the sampling process. Extending the idea to the bias estimation, we can use the same strategy to solve the problem. As the empirical distribution $\hat{F}$ is known, it is easy to generate a manual sample from $\hat{F}$. Assuming the size of the sample generated is n , the sample is

$$\mathbf{x}^* = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_n^*\}. \quad (2.2)$$

Thus we have an empirical distribution $\hat{F}^*$ of the empirical distribution $\hat{F}$, where,

$$\hat{F}^* : \text{mass } \frac{1}{n} \text{ at } \mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_n^*,$$

and $\{\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_n^*\}$ are the observed values of the sample $\{\mathbf{X}_1^*, \mathbf{X}_2^*, \dots, \mathbf{X}_n^*\}$, and a corresponding $\hat{\theta}^* = \theta(\hat{F}^*)$ is the estimate of $\hat{\theta}$. Thus an estimator of the bias is

$$\hat{\text{Bias}} = E^* [\hat{\theta}^* - \hat{\theta}] = E^* [\theta(\hat{F}^*) - \theta(\hat{F})], \quad (2.3)$$

where E^* is the conditional expectation of $\hat{F}$ given $\{\mathbf{X}_1^* = \mathbf{x}_1^*, \mathbf{X}_2^* = \mathbf{x}_2^*, \dots, \mathbf{X}_n^* = \mathbf{x}_n^*\}$. The procedure to estimate the bias described above is called *Bootstrap*. An example to clarify issues is given below.

Example 2.1

Consider a simple case, in which θ is the expectation of the unknown distribution F ,

$$\theta(F) = \int \mathbf{x} dF.$$

Then the estimate of θ is

$$\hat{\theta} = \theta(\hat{F}) = \int \underline{x} d\hat{F} = \bar{\underline{x}},$$

where $\bar{\underline{x}} = \frac{1}{n} \sum_i \underline{x}_i$ is the sample mean. Now if the observation values $\{\underline{x}_1^*, \underline{x}_2^*, \dots, \underline{x}_n^*\}$ of a bootstrap sample are carried out by a random generator distributed uniformly on $\{1, 2, \dots, n\}$, we can calculate a resampling value of $\hat{\theta}$,

$$\hat{\theta}^* = \theta(\hat{F}^*) = \int \underline{x} d\hat{F}^* = \bar{\underline{x}}^*,$$

where $\bar{\underline{x}}^* = \frac{1}{n} \sum_i \underline{x}_i^*$ is the mean of the bootstrap sample. The procedure for generating a bootstrap sample is repeated B times, and there are B values of $\hat{\theta}$ obtained from the B bootstrap samples $\{\underline{x}_1^{b*}, \underline{x}_2^{b*}, \dots, \underline{x}_n^{b*}\}$, $b = 1, 2, \dots, B$,

$$\hat{\theta}_b^* = \bar{\underline{x}}_b^*.$$

The bootstrap estimate of $\hat{\theta}$ is the average of these $\hat{\theta}_b^*$, so,

$$\hat{\theta}^* = \frac{1}{B} \sum_b \hat{\theta}_b^* = \frac{1}{B} \sum_b \bar{\underline{x}}_b^*. \quad (2.4)$$

Thus, the bootstrap estimate of the bias $\bar{\underline{x}} - \theta$ is

$$\text{Bias} = \hat{\theta}^* - \bar{\underline{x}} = \frac{1}{B} \sum_b \bar{\underline{x}}_b^* - \bar{\underline{x}}. \quad (2.5)$$

The procedure presented above is a general bootstrap procedure called basic or simple bootstrap. ■

In summary, the bootstrap method takes the strategy of replacing $\hat{F}$ with $\hat{F}^*$ and F with $\hat{F}$ in estimating a parameter $Q(\hat{F}, F)$ with unknown distribution F . A typical bootstrap procedure which takes the above steps, is given formally below.

Algorithm 2.1 Basic Bootstrap

Input: (i) n , the size of the training sample;
(ii) $\mathbf{x}[1], \mathbf{x}[2], \dots, \mathbf{x}[n]$ the sample data of the training sample;
(iii) B , the number of time bootstrap resampling is repeated.

Output: the bootstrap estimator.

Method
BEGIN
bootstrap-estimator = 0
For ($i = 1$ to B)
 For ($j = 1$ to n) (2.6)
 $m = \text{random integer in } [1, n]$
 $\mathbf{y}[j] = \mathbf{x}[m]$
 End-For
 bootstrap-estimator = bootstrap-estimator + $Q(\mathbf{y}[1], \mathbf{y}[2], \dots, \mathbf{y}[n])$ (2.7)
End-For
bootstrap-estimator = bootstrap-estimator / B
Return bootstrap-estimator
END Basic Bootstrap

Note, the **For** loop (2.6) is to generate a bootstrap sample, e.g. a sample from the empirical distribution $\hat{F}$; and the $Q(\mathbf{y}[1], \mathbf{y}[2], \dots, \mathbf{y}[n])$ in (2.7) is to calculate a bootstrap estimator of the parameter Q with the bootstrap sample, e.g.

$$Q(\mathbf{y}[1], \mathbf{y}[2], \dots, \mathbf{y}[n]) = Q(\hat{F}^*, \hat{F}).$$

The question of how to calculate $Q(\mathbf{y}[1], \mathbf{y}[2], \dots, \mathbf{y}[n])$ depends on the estimating formula of parameter Q . For instance, in **Example 2.1** where we tried to estimate the bias of the mean, it is

$$Q(\mathbf{y}[1], \mathbf{y}[2], \dots, \mathbf{y}[n]) = \frac{1}{n} \sum_j \mathbf{y}[j] - \frac{1}{n} \sum_i \mathbf{x}[i].$$

Efron suggested choosing $B = 200$ as the number of times the bootstrap resampling is repeated[E83], which is quite adequate for most of purposes.

2.3 Bootstrap Estimators

2.3.1 Bias

In this section we will discuss the bootstrap approaches for estimating the bias, variance, and the quantile. Although the bootstrap method is regarded as a distribution free method, it is also possible to have a parameterized bootstrap. The concept of the parametric bootstrap will also be introduced in the section, in order to present an alternative view of the bootstrap techniques. At the end of the section, we will present a bootstrap procedure, the SIMDAT algorithm, for density function simulation, which shows an example of a more involved bootstrap procedure. The SIMDAT algorithm will be applied in subsequent chapters to some related problems of statistical pattern recognition.

As defined in section 2.2, the bias of an estimate $\theta(\hat{F})$ is $\text{Bias} = E_F[\hat{\theta} - \theta] = E_F[\theta(\hat{F}) - \theta(F)]$. After drawing each bootstrap sample $\{\mathbf{x}_1^{b*}, \mathbf{x}_2^{b*}, \dots, \mathbf{x}_n^{b*}\}$ from $\hat{F}$ and replacing $\hat{F}$ with $\hat{F}^*$ to get the corresponding $\hat{\theta}_b^*$, the bootstrap estimate of bias will be

$$\hat{\text{Bias}} = \frac{1}{B} \sum_b \hat{\theta}_b^* - \hat{\theta}. \quad (2.8)$$

Comparing this expression to (2.3), we can substitute $\frac{1}{B} \sum_b \hat{\theta}_b^*$ for $E^*[\hat{\theta}^*]$ here, which is analogous to using $\bar{\mathbf{x}}$ for estimating $E_F[\mathbf{X}]$.

In order to apply (2.8), it is not necessary to require $\hat{\theta}$ to be “the same statistic as θ ” [E82]. We can take $\hat{\theta}$ to be the sample median while θ can be set to be $E_F[\mathbf{X}]$. Hence the bootstrap technique provides more flexibility to yield an estimator. If $\hat{\theta}$ is an estimate of θ and a bootstrap estimate of the bias (2.3) is provided, a bootstrap estimate of θ can be obtained by correcting the bias of the original estimate as:

$$\hat{\theta}_{\text{BOOT}} = \hat{\theta} - \hat{\text{Bias}} = 2\hat{\theta} - \hat{\theta}^* \quad (2.9)$$

It will be seen, in Chapter 3 and 4, how this strategy can be used to estimate the error bound and error rate of a classifier.

2.3.2 Variance

In numerous applications and from a theoretical point of view, there is usually an interest in estimating the variance of a statistic $\hat{\theta}$. In general, it is not easy to give a variance estimate of statistics $S^2 = \frac{1}{n} \sum_i (\mathbf{x}_i - \bar{\mathbf{x}})^2$ while the variance of statistics $\bar{\mathbf{x}}$ is shown to be

$$\text{VAR}(\bar{\mathbf{x}}) = \frac{1}{n(n-1)} \sum_i (\mathbf{x}_i - \bar{\mathbf{x}})^2.$$

Generally it is hard to compute the variance $\text{VAR}(\hat{\theta})$. With the bootstrap method, $\text{VAR}(\hat{\theta})$ can be estimated as

$$\hat{\text{VAR}}_{\text{BOOT}}(\hat{\theta}) = \frac{1}{B} \sum_b (\hat{\theta}_b^* - \bar{\theta}^*)^2, \quad (2.10)$$

where $\bar{\theta}^* = \frac{1}{B} \sum_b \hat{\theta}_b^*$, after obtaining each $\hat{\theta}_b^*$ from the bootstrap sample $\{\mathbf{x}_1^{b*}, \mathbf{x}_2^{b*}, \dots, \mathbf{x}_n^{b*}\}$.

Example 2.2

For the standard deviation error $\hat{\theta} = S = \sqrt{\frac{1}{n} \sum_i (\mathbf{x}_i - \bar{\mathbf{x}})^2}$, the corresponding bootstrap standard error is

$$\hat{\theta}_b^* = S_b^* = \sqrt{\sum_i \mathbf{w}_i^{b*} (\mathbf{x}_i - \bar{\mathbf{x}}^{b*})^2}, \quad b = 1, 2, \dots, B. \quad (2.11)$$

The $\mathbf{w}_i^{b*}$ on the right side is the weight of $\mathbf{x}_i$ in the bootstrap sample $\mathbf{X}^{b*} = \{\mathbf{x}_1^{b*}, \mathbf{x}_2^{b*}, \dots, \mathbf{x}_n^{b*}\}$, which equals the number n_i^{b*} of $\mathbf{x}_i$ appearing in $\mathbf{X}^{b*}$ divided by the bootstrap sample size n ,

$$\mathbf{w}_i^{b*} = n_i^{b*} / n. \quad (2.12)$$

Also, $\bar{\mathbf{x}}^{b*}$ is the mean of the bootstrap sample $\mathbf{X}^{b*}$,

$$\bar{\mathbf{x}}^{b*} = \sum_i \mathbf{w}_i^{b*} \mathbf{x}_i. \quad (2.13)$$

After obtaining all $\{\hat{\theta}_b^* : b = 1, 2, \dots, B\}$, the bootstrap variance estimate of S is just a direct computing of (2.10),

$$\mathbf{V}\hat{\mathbf{A}}\mathbf{R}_{\text{BOOT}}(S) = \frac{1}{B} \sum_b (S_b^* - \bar{S}^*)^2,$$

where $\bar{S}^* = \frac{1}{B} \sum_b S_b^*$. ■

The expressions (2.11) - (2.13) imply that the bootstrap method provides a scheme by which we can assign weights to the sample data, which leads to other resampling strategies which will be discussed later in this chapter.

2.3.3 Quantile

Another quantity of considerable interest in statistics is the confidence interval of $\hat{\theta}$. Typically, with a given value θ_0 , a question of pertinence is the probability of

$$\Pr(\hat{\theta} < \theta_0). \quad (2.14)$$

Alternatively, with a given level α , the question of interest is the boundary θ_0 such that

$$\Pr(\hat{\theta} < \theta_0) = \alpha. \quad (2.15)$$

It is very hard to answer the above questions in most cases if the distribution F is unknown. With the bootstrap approach, the bootstrap samples can be first used to obtain a set of estimates $\{\hat{\theta}_b^* : b = 1, 2, \dots, B\}$, and then these $\{\hat{\theta}_b^* : b = 1, 2, \dots, B\}$ are just taken as a sample of $\hat{\theta}$. The estimate of (2.14) will thus be

$$\hat{\Pr}(\hat{\theta} < \theta_0) = \frac{1}{B} \#\{\hat{\theta}_b^* < \theta_0\}. \quad (2.16)$$

By sorting $\{\hat{\theta}_b^* : b = 1, 2, \dots, B\}$ as below:

$$\hat{\theta}_{(0)}^* < \hat{\theta}_{(1)}^* < \dots < \hat{\theta}_{(B)}^*,$$

the estimator of θ_0 in (2.15) will be

$$\hat{\theta}_0 = \hat{\theta}_{(\lfloor \alpha(B+1) \rfloor)}^* \quad (2.17)$$

where $\lfloor \bullet \rfloor$ is the floor function.

2.3.4 Parametric Bootstrap

Though it is a distribution free approach the bootstrap can also be used in a parametric way. Consider the case when the sample data $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n\}$ follows a normal distribution $F = N(\mu, \Sigma)$, where the μ and σ^2 are unknown parameters. Then the parametric maximum likelihood estimator of F is

$$\hat{F}_{\text{NORMAL}} = N(\hat{\mu}, \hat{\Sigma}).$$

Now the bootstrap procedure is carried out exactly as described above except that $\hat{F}_{\text{NORMAL}}$ takes the place of $\hat{F}$ in (2.6). Thus the loop (2.6) in **Algorithm 2.1** is now changed to

```

For (j = 1 to n)
     $\mathbf{y}[j]$  = Random vector drawn from  $N(\hat{\mu}, \hat{\Sigma})$ .
End-For

```

Therefore the bootstrap sample will be drawn from $\hat{F}_{\text{NORMAL}}$, which means that the bootstrap is not merely putting weights on the data $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n\}$.

2.3.5 Density Function

Density function is, of course, a great interest in statistics, because numerous problems in statistics would be solved if the density function was known. Unfortunately, in all of the statistical problems, the density functions involved are unknown, or at most, only a model or an explicit form of the density functions is known. In statistical pattern recognition, density functions play more important roles, because knowing the density

functions of the classes will help us to build and evaluate the corresponding classifiers. But the question of how to estimate a density function is still a significant problem, especially in a small sample size case. It can be seen that the bootstrap provides a useful tool for estimating density functions.

The algorithm presented here is due to Taylor and Thompson [TT92], called the *SIMDAT* algorithm. The purpose of the *SIMDAT* algorithm is to provide a sampling scheme which could generate a pseudo-data sample very close to that drawn from the kernel density function estimator of

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_i K(\mathbf{x} - \mathbf{X}_i, \Sigma_i), \quad (2.18)$$

where the Σ_i in $K(\bullet)$ is a locally estimated covariance matrix. The details of the *SIMDAT* algorithm are given below.

Algorithm 2.2 SIMDAT

Input: (i) n , the size of the training sample;
(ii) $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[n]$ the sample vector of the training sample;
(iii) m , the number of the nearest neighborhood of each sample data.
(iv) N , the repeated times of the bootstrap resampling.

Output: $\underline{z}[1], \underline{z}[2], \dots, \underline{z}[N \times m]$, a set of pseudo-sample vector.

Method

BEGIN

Re-scale the sample data set $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[n]$ so that the marginal sample variances in each vector component are the same

For ($i = 1$ to n)

find the m nearest neighbors $\underline{y}[i, 1], \underline{y}[i, 2], \dots, \underline{y}[i, m]$ of $\underline{x}[i]$ (include $\underline{x}[i]$ itself),

For ($j = 1$ to m)

$$\underline{y}[i, 0] = \underline{y}[i, 0] + \frac{1}{m} \underline{y}[i, j]$$

End-For

For ($j = 1$ to m)

$$\underline{y}[i, j] = \underline{y}[i, j] - \underline{y}[i, 0]$$

End-For

End-For

For ($i = 1$ to N)

k = random integer in $[1, n]$

For ($p = 1$ to m)

$$\underline{z}[(i - 1) * m + p] = 0$$

For ($q = 1$ to m)

$$u = \text{random number from } U\left(\frac{1}{m} - \sqrt{\frac{3(m-1)}{m^2}}, \frac{1}{m} + \sqrt{\frac{3(m-1)}{m^2}}\right)$$

$$\underline{z}[(i - 1) * m + p] = \underline{z}[(i - 1) * m + p] + u * \underline{y}[k, q]$$

End-For

$$\underline{z}[(i - 1) * m + p] = \underline{z}[(i - 1) * m + p] + \underline{y}[k, 0]$$

End-For

End-For

Return $\underline{z}[1], \underline{z}[2], \dots, \underline{z}[N \times m]$

END SIMDAT

There are three things done in the first **For** loop of the SIMDAT algorithm. They are:

1. Find the m nearest neighbors of a sample vector, including the sample vector itself;
2. Calculate the mean of the m nearest neighbors which is put in $\underline{y}[i, 0]$;
3. Deduce the mean from each neighbor.

The third step above makes the neighbor sample vectors to have a local distribution with the mean of zero. In generating pseudo-sample, a randomly weighted linear combination of the nearest neighbors is given, and then the local mean $\underline{y}[i, 0]$ is added back. Here we see that the local mean $\underline{y}[i, 0]$ takes the place of the original sample vector $\underline{x}[i]$ in the SIMDAT algorithm, which incorporates smoothness into the pseudo-sample vector.

Some properties of the pseudo-data sample are not difficult to see. First, it is easy to calculate the expectations, variances and covariances of the uniform distribution sample $\{\underline{u}_j\}_{j=1}^m$; these quantities are:

$$E(\underline{u}_j) = \frac{1}{m},$$

$$\text{Var}(\underline{u}_j) = \frac{m-1}{m^2}$$

and

$$\text{Cov}(\underline{u}_i, \underline{u}_j) = 0, \quad \text{for } i \neq j.$$

If we now treat the m nearest neighbors $\{\underline{x}_{i,1}, \underline{x}_{i,2}, \dots, \underline{x}_{i,m}\}$ of sample vector $\underline{x}_i$ as a sample from a truncated distribution with mean vector $\mu = (\mu_1, \mu_2, \dots, \mu_d)^T$ and covariance matrix $\Sigma = (\sigma_{i,j})$, it can be seen that the linear combination

$$\underline{Z} = \sum_j u_j \underline{x}_{i,j},$$

where $\underline{x}_{i,m} = \underline{x}_i$, will satisfy

$$E(\underline{Z}) = \underline{\mu}$$

and

$$\text{COV}(\underline{Z}) = (\sigma_{i,j}^*),$$

where

$$\sigma_{i,j}^* = \begin{cases} \sigma_i^2 + \frac{m-1}{m} \mu_i^2, & i = j \\ \sigma_{i,j} + \frac{m-1}{m} \mu_i \mu_j, & i \neq j \end{cases}.$$

It is clear that the linear combination $\underline{Z}$ will have the same expectation and covariance as the sample data $\{\underline{x}_{i,1}, \underline{x}_{i,2}, \dots, \underline{x}_{i,m}\}$ if $\underline{\mu} = 0$. Therefore the pseudo-data sample generated by the above procedure will be very close to that generated by the density function of (2.18).

2.4 Properties of Bootstrap

In the previous section we presented several examples to show applications of the bootstrap technique to different problems. Though the bootstrap is a very convenient and appealing tool for data analysis, it is still necessary to confirm theoretically its suitability for the problems at hand. There are already examples in the literature for which bootstrap has failed [Ma92]. Therefore we would like to know “when does bootstrap work?”. Theorem 2.1 and 2.2 of this section answer the question within a large sample context.

To illustrate the large sample results, we use the following notations,

- a) $\mathbf{X}_n = \{\mathbf{X}_{n,1}, \mathbf{X}_{n,2}, \dots, \mathbf{X}_{n,n}\}$ is an i.i.d. sample of size n with unknown distribution F_n ;
- b) $\hat{F}_n$ is the empirical distribution of the sample $\mathbf{X}_n$;
- c) $\mathbf{X}_n^* = \{\mathbf{X}_{n,1}^*, \mathbf{X}_{n,2}^*, \dots, \mathbf{X}_{n,n}^*\}$ is a bootstrap sample, an i.i.d. sample, with the empirical distribution $\hat{F}_n^*$;
- d) $\hat{F}_n^*$ is the empirical distribution of bootstrap sample $\mathbf{X}_n^*$;
- e) $\theta_n(F)$ is a functional of the interest. In the theorem given below it is a linear functional

$$\theta_n(F) = \int \mathbf{g}_n(\mathbf{x}) d F(\mathbf{x}).$$

What the bootstrap procedure tries to do now is to estimate the distribution

$$P(\theta_n(\hat{F}_n) - \theta_n(F_n) \leq t).$$

The theorem below gives the necessary and sufficient conditions under which the bootstrap procedure will be successful.

Theorem 2.1

Consider a sequence $\mathbf{X}_{n,1}, \mathbf{X}_{n,2}, \dots, \mathbf{X}_{n,n}$ of i.i.d variables with distribution F_n . For a function $\mathbf{g}_n$ consider $\hat{\theta}_n = \theta_n(\hat{F}_n) = \frac{1}{n} \sum_i \mathbf{g}_n(\mathbf{X}_{n,i})$. Consider a bootstrap sample $\mathbf{X}_{n,1}^*, \mathbf{X}_{n,2}^*, \dots, \mathbf{X}_{n,n}^*$ with empirical distribution $\hat{F}_n^*$. Denote $\hat{\theta}_n^* = \theta_n(\hat{F}_n^*)$, d_∞ to be the Kolmogorov distance, and $\mathcal{L}^*(\dots)$ to be the conditional law $\mathcal{L}^*(\dots | \mathbf{X}_{n,1}, \mathbf{X}_{n,2}, \dots, \mathbf{X}_{n,n})$. Then for every sequence t_n the following assertions are equivalent:

(i) $\hat{\theta}_n$ is asymptotically normal: There exist σ_n with

$$d_{\infty}(\mathcal{L}(\hat{\theta}_n - t_n), \mathcal{N}(0, \sigma_n^2)) \rightarrow 0, \quad \text{a.s.} \quad (2.19)$$

(ii) The normal approximation with estimated variance works:

$$d_{\infty}(\mathcal{L}(\hat{\theta}_n - t_n), \mathcal{N}(0, \hat{S}_n^2)) \rightarrow 0 \quad (\text{in probability}), \quad (2.20)$$

where $\hat{S}_n^2 = \frac{1}{n^2} \sum_i (g_n(X_{n,i}) - \hat{\theta}_n)^2$.

(iii) Bootstrap works:

$$d_{\infty}(\mathcal{L}(\hat{\theta}_n - t_n), \mathcal{L}(\hat{\theta}_n^* - \hat{\theta}_n)) \rightarrow 0 \quad (\text{in probability}). \quad (2.21)$$

If (i), (ii) or (iii) hold then t_n can be chosen as the mean of the truncated variables

$$\mu + E[(g_n(X_{n,i}) - \mu) I(|g_n(X_{n,i}) - \mu| \leq n\sigma_n)],$$

where μ is a median of the distribution of $g_n(X_{n,i})$. ■

The meaning of Theorem 2.1 is explicit, e.g. if a sequence t_n satisfies either of the conditions (i), (ii) and (iii) then it can be shown to satisfy the other two. The results of Theorem 2.1 provide the condition under which the bootstrap works for the i.i.d. models. It implies that if $\theta_n(\hat{F}_n)$ is a good simulation of $\theta_n(F_n)$ then $\theta_n(\hat{F}_n^*) - \theta_n(\hat{F}_n)$ will be a good simulation of $\theta_n(F_n) - \theta_n(\hat{F}_n)$, and vice versa. After adding more conditions the bootstrap scheme can be shown to work under non i.i.d. models as well, which is the result of Theorem 2.2 given below.

Theorem 2.2

Consider a sequence $X_{n,1}, X_{n,2}, \dots, X_{n,n}$ of independent variables with distribution $F_{n,i}$. For a function g_n the $\hat{\theta}_n$ and $\hat{\theta}_n^*$ are defined as in *Theorem 2.1*. Then for every sequence t_n the following assertions are equivalent:

(i) There exist σ_n such that for every $\varepsilon > 0$

$$\sup_{1 \leq i \leq n} P(|\frac{g_n(X_{n,i}) - t_n}{n\sigma_n}| \geq \varepsilon) \rightarrow 0 \quad (\text{asymptotic negligibility}), \quad (2.22)$$

$$\sum_i (E[(\frac{g_n(X_{n,i}) - t_n}{n\sigma_n}) I(|\frac{g_n(X_{n,i}) - t_n}{n\sigma_n}| \leq \varepsilon)])^2 \rightarrow 0 \quad (2.23)$$

and such that

$$d_\infty(\mathcal{L}(\hat{\theta}_n - t_n), N(0, \sigma_n^2)) \rightarrow 0.$$

(ii) The normal approximation with estimated variance works:

$$d_\infty(\mathcal{L}(\hat{\theta}_n - t_n), N(0, \hat{S}_n^2)) \rightarrow 0 \quad (\text{in probability}),$$

$$\text{where } \hat{S}_n^2 = \frac{1}{n^2} \sum_i (g_n(X_{n,i}) - \hat{\theta}_n)^2.$$

(iii) Bootstrap works:

$$d_\infty(\mathcal{L}(\hat{\theta}_n - t_n), \mathcal{L}(\hat{\theta}_n^* - \hat{\theta}_n)) \rightarrow 0 \quad (\text{in probability}). \quad \blacksquare$$

Note that Theorem 2.2, $X_{n,1}, X_{n,2}, \dots, X_{n,n}$ are not required to have the same distribution although they are still assumed independent. Since we forfeit the i.i.d property of the samples, the extra conditions (2.22) and (2.23) are required for ensuring the consistency of bootstrap.

The proofs the of above two theorems are omitted here as they are outside the scope of the thesis. The reader is referred to Mammen's paper [Ma92] for more details.

2.5 Other Resampling Approaches

In **Example 2.2** it was demonstrated that the bootstrap scheme, in some sense, assigns weights to the sample data. This idea can be used to generalize the bootstrap in the following way.

Let $\underline{\mathbf{P}}^* = (\mathbf{P}_1^*, \mathbf{P}_2^*, \dots, \mathbf{P}_n^*)$ be any probability vector on the n -dimensional simplex

$$\varphi_n = \{\underline{\mathbf{P}}^* : \mathbf{P}_i^* \geq 0, \sum_i \mathbf{P}_i^* = 1\}, \quad (2.24)$$

called a *resampling vector* [Ef82]. For a sample $\mathbf{X} = \{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$ a re-weighted empirical probability distribution $\hat{\mathbf{F}}^*$ is defined with a resampling vector $\underline{\mathbf{P}}^*$ as

$$\hat{\mathbf{F}}^* : \text{mass } \mathbf{P}_i^* \text{ on } \underline{\mathbf{x}}_i, i = 1, 2, \dots, n, \quad (2.25)$$

where $\{\underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \dots, \underline{\mathbf{x}}_n\}$ are the observed values of the sample $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$. Thus, a resampled value of $\hat{\theta}$, say $\hat{\theta}^*$, will be

$$\hat{\theta}^* = \theta(\hat{\mathbf{F}}^*(\underline{\mathbf{P}}^*)) = \theta(\underline{\mathbf{P}}^*). \quad (2.26)$$

As before, after repeatedly generating the resampling vector $\underline{\mathbf{P}}^*$ B times, a sample of $\hat{\theta}$ will be obtained, say $\{\hat{\theta}_b^* : b = 1, 2, \dots, B\}$ which is just what was obtained in step 3 of the basic bootstrap mentioned above. The basic bootstrap is, from this point of view, a specific case of (2.25) - (2.26) in which the resampling vector $\underline{\mathbf{P}}^*$ takes the form

$$\underline{\mathbf{P}}_i^* = n_i^* / n, \quad (2.27)$$

where n_i^* is the number of x_i appearing in a bootstrap sample. This means that $\underline{\mathbf{P}}^*$ follows a multinominal distribution,

$$\underline{\mathbf{P}}^* \sim \frac{1}{n} \text{Mult}(n, \underline{\mathbf{P}}_0), \quad (2.28)$$

where $\underline{\mathbf{P}}_0 = (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n})$ is a n -dimensional vector. In other words, This is possible to execute a resampling procedure by choosing another $\underline{\mathbf{P}}^*$. These arguments lead to the Bayesian Bootstrap and Random Weighting Method introduced by Rubin [ST95] and Zhen [Zh87] respectively.

2.5.1 Bayesian Bootstrap

Consider the case where an i.i.d sample $\mathbf{X} = \{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$ comes from a distribution $F(\mathbf{X} \mid \theta)$ with a unknown parameter θ which is of interest. In frequency statistical analysis the θ is regarded as a nonrandom constant. In Bayesian statistical analysis the θ is taken as a random variable (or vector) with a prior distribution, and the sample $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$ is considered coming from a conditional distribution of $\mathbf{X}$ given θ . The Bayesian estimator of θ will be given in a form of conditional expectation of θ given $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\}$,

$$\hat{\theta} = E(\theta \mid \underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n) = \int \theta \, dF(\theta \mid \underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n), \quad (2.30)$$

where the conditional distribution $F(\theta \mid \underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n)$ is known as the posterior distribution. Usually, posterior distributions are expressible only in terms of complicated

analytical functions, and it is hard to calculate the marginal distributions and moments of posterior distributions. Commonly, a normal approximation to the posterior distribution is suggested. Although it is easy to estimate, this approximation lacks high order accuracy. To address this problem Rubin introduced a nonparametric method called the *Bayesian Bootstrap* under a specific assumption on the prior distribution. Here we discuss the basic algorithm of the Bayesian bootstrap with a noninformative prior distribution. The algorithm is formally given below,

Algorithm 2.3 Bayesian Bootstrap

Input: (i) n , the size of the training sample;
(ii) $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[n]$ the sample data of the training sample;
(iii) B , the repeated times of the bootstrap resampling.

Output: the Bayesian bootstrap estimator.

Method
BEGIN
bootstrap-estimator = 0
For ($i = 1$ to B)
 $u[n] = 1$
 For ($j = 1$ to $n - 1$)
 $u[j] = \text{random number from } U[0, 1]$
 End-For
 Sort $u[1], u[2], \dots, u[n]$
 For ($j = n$ to 2)
 $P[j] = u[j] - u[j - 1]$
 End-For
 $P[1] = u[1]$
 calculate a Bayesian bootstrap estimating value $\hat{\theta}^* = \theta(\hat{F}^*)$
 bootstrap-estimator = bootstrap-estimator + $\hat{\theta}^*$
End-For
bootstrap-estimator = bootstrap-estimator / B
Return bootstrap-estimator
END Bayesian Bootstrap

In the algorithm, an i.i.d. random sample of size $n - 1$ from the uniform distribution $U[0, 1]$ is first generated. After obtaining the order statistics

$$0 \leq u[1] \leq u[2] \leq \dots \leq u[n] = 1$$

by sorting the uniform random sample, the component $P[j]$ of the resampling vector $\underline{P}$ is assigned the value of the difference $u[j] - u[j - 1]$ of the uniform order statistics, for $j = 2, \dots, n$, and $P[1] = u[1]$ is the difference $u[1] - 0$. Thus, the re-weighted empirical probability $\hat{F}^*$ defined in (2.25) is given by the algorithm, as

$$\hat{F}^* : \text{mass } P[i] \text{ on } \underline{x}[i], i = 1, 2, \dots, n.$$

Accordingly a calculation of $\hat{\theta}^* = \theta(\hat{F}^*)$ is carried out.

From the algorithm given above the relationship between the prior distribution and the resampling vector $\underline{P}^*$ defined in (2.31) is not obvious. To clarify issues, consider a very simple case in which the sample $\{X_1, X_2, \dots, X_n\}$ follows a 0-1 distribution, $P(X = 1) = \theta$, and $P(X = 0) = 1 - \theta$. Let $m = \sum_i x_i$, where $\{x_1, x_2, \dots, x_n\}$ are the observation values of the sample. Then,

$$P(X_1 = x_1, X_2 = x_1, \dots, X_n = x_n | \theta) = \theta^m (1 - \theta)^{n-m}.$$

Under the assumption of using a noninformative prior distribution, the posterior density function will be

$$c \theta^m (1 - \theta)^{n-m},$$

where c is a constant. Therefore the posterior distribution is a Beta distribution. On the other hand, the difference of the order statistics of the uniform distribution $U[0, 1]$ has also a form of Beta distribution (see [PC94]). Thus the resampling vector $\underline{P}^*$ of the Bayesian bootstrap fits the noninformative prior distribution case.

Example 2.3

Consider the same problem discussed in **Example 2.1**. We use the Bayesian bootstrap method to estimate the bias of $\bar{x} - \theta$, where $\bar{x}$ is the sample mean and θ is the expectation of the unknown distribution F . Now we draw an i.i.d. sample of size $n - 1$ from the uniform distribution $U[0, 1]$, say $\{u_1, u_2, \dots, u_{n-1}\}$, and get the order statistics by sorting the sample,

$$0 = u_{(0)} \leq u_{(1)} \leq u_{(2)} \leq \dots \leq u_{(n-1)} \leq u_{(n)} = 1.$$

After this, a resampling vector $\underline{P}^*$ is made by assigning each of its components the value

$$P_i^* = u_{(i)} - u_{(i-1)}. \quad (2.31)$$

This time the corresponding resampling value of $\hat{\theta}$ will be

$$\hat{\theta}^* = \theta(\hat{F}^*) = \int \underline{x} d\hat{F}^* = \bar{x}^* = \sum_i P_i^* \underline{x}_i. \quad (2.32)$$

Repeatedly drawing B samples of size $n - 1$ from the uniform distribution $U[0, 1]$, we obtain B resampling vectors $\underline{P}^{b*} = (P_1^{b*}, P_2^{b*}, \dots, P_n^{b*})$, $b = 1, 2, \dots, B$, and so we get B resampling values of $\hat{\theta}$,

$$\hat{\theta}_b^* = \bar{x}_b^*,$$

where $\bar{x}_b^* = \sum_i P_i^{b*} \underline{x}_i$. The Bayesian bootstrap estimator of $\hat{\theta}$ is the average of these $\hat{\theta}_b^*$'s,

$$\hat{\theta}^* = \frac{1}{B} \sum_b \hat{\theta}_b^* = \frac{1}{B} \sum_b \bar{x}_b^*.$$

Therefore the Bayesian bootstrap estimator of the bias $\bar{x} - \theta$ is

$$\text{Bias} = \hat{\theta}^* - \bar{x} = \frac{1}{B} \sum_b \bar{x}_b^* - \bar{x}. \quad \blacksquare$$

It can be seen that the basic concept of the Bayesian bootstrap algorithm is almost the same as that of the basic bootstrap algorithm except that the resampling vector $\underline{p}^*$ is continuously distributed in the Bayesian bootstrap while it is discretely distributed in the basic bootstrap case. Hence the Bayesian bootstrap algorithm can be regarded as a smoothed version of the basic bootstrap scheme.

2.5.2 Random Weighting Method

It can be argued that the method discussed above does not necessarily have to be related to the prior distributions. With some insight it can be seen that a continuously distributed resampling vector $\underline{p}^*$ can also be used to get a re-weighted empirical probability distribution $\hat{F}^*$ as given in (2.25). In other words, we can randomly generate the weighting vector $\underline{p}^*$ to get the resampling empirical distribution $\hat{F}^*$. This concept was introduced by Zhen (1987) and is called the *Random Weighting Method*. Thus, the random weighting vector $\underline{p}^*$ can be obtained by the following scheme, given formally below.

Algorithm 2.4 Random Weighting Method

Input: (i) n , the size of the training sample;
(ii) $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[n]$ the sample data of the training sample;
(iii) B , the repeated times of the bootstrap resampling.

Output: the random weighting estimator.

Method
BEGIN
random-weighting-estimator = 0
For ($i = 1$ to B)
 $z[0] = 0$
 For ($j = 1$ to n)
 $z[j]$ = random number from a nonnegative distribution P
 $z[0] = z[0] + z[j]$
 End-For
 For ($j = 1$ to n)
 $P[j] = z[j] / z[0]$
 End-For
 calculate a Bayesian bootstrap estimating value $\hat{\theta}^* = \theta(\hat{F}^*)$
 random-weighting-estimator = random-weighting-estimator + $\hat{\theta}^*$
End-For
random-weighting-estimator = random-weighting-estimator / B
Return random-weighting-estimator
END Random Weighting Method

In this algorithm, an i.i.d. random sample of size n is first generated from a nonnegative distribution P . Then a resampling vector P is obtained by assigning its component $P[j]$ the normalized value of $z[j] / z[0]$, for $j = 1, 2, \dots, n$. Similar to the Bayesian bootstrap, the re-weighted empirical probability distribution $\hat{F}^*$ is given, in this algorithm, as

$$\hat{F}^* : \text{mass } P[i] \text{ on } \underline{x}[i], i = 1, 2, \dots, n.$$

Then a calculation of $\hat{\theta}^* = \theta(\hat{F}^*)$ will be carried out accordingly.

One of the simplest ways is to take an i.i.d. sample $\{Z_1, Z_2, \dots, Z_n\}$ from the uniform distribution $U[0, 1]$ to construct the random weighting vector $\underline{P}^*$. Indeed, the

uniform distribution $U[0, 1]$ will be used for random weighting method simulations through this thesis. Other steps to carry out a random weighting estimator are similar to the basic bootstrap and the Bayesian bootstrap.

If we now examine expressions (2.11) - (2.13) given in **Example 2.2**, it can be seen that the w_i^{b*} used in the bootstrap can also be regarded as, in some sense, a random weighting method. Although the Bayesian bootstrap is another random weighting method, in this thesis the phrase “random weighting method” will only be used for the case of the random weighting vector $\mathbf{P}^*$ constructed with a sample from the uniform distribution $U[0, 1]$.

A few results related to the theoretical aspects of the Bayesian bootstrap and random weighting method are available in the literature. Under certain assumptions both the Bayesian bootstrap and random weighting method have excellent analytic properties, such as consistency, and both approximately normal distributed. As this is outside the scope of this thesis, these concepts will not be discussed here. The reader is referred to [ST95] for more details.

2.6 Summary

In this chapter, we introduced the concept of bootstrap and discussed some details of the bootstrap technique. The idea of the bootstrap technique is simple and elegant, it involves “simulating the simulator”. What the bootstrap mainly does is to draw a random sample, called a bootstrap sample, from the empirical distribution. It then combines the bootstrap and original samples together to obtain the estimates sought for. With this

technique, we are able to extract some information of the relationship between the true distribution F and empirical distribution $\hat{F}$, such as the bias, standard deviation, and the empirical distribution of a parameter estimate.

While Algorithm 2.1 is a general illustration of the basic bootstrap procedure, the details of calculating the bootstrap estimates are given individually in the examples of Section 2.3. Though the bootstrap calculations are usually similar to that of the traditional estimation, (as shown in Examples 2.1 and 2.2), schemes more complex than traditional estimation, such as the SIMDAT algorithm, are possible. The idea of using pseudo-samples in the SIMDAT algorithm is very interesting, it provides a way by which we are able to obtain 'extra' sample data. In Chapters 4 and 5, we will extend this idea to the problems of error rate estimating and constructing classifiers.

The results of Theorems 2.1 and 2.2 provided, at least, some theoretical foundation for the bootstrap technique, though they are only considered the one-dimensional linear functional cases. Of course, in practice, we would pay more attention to small sample cases, but the large sample properties are necessary for theoretical purposes.

The discussion of the Bayesian bootstrap and random weighting method provided alternative schemes to obtain the re-weighted empirical distribution $\hat{F}^*$. Instead of using discrete resampling vectors in the basic bootstrap algorithm, the Bayesian bootstrap and random weighting method use continuous resampling vectors. Because of this, there is no bootstrap sample drawn in both of these schemes. We calculate the bootstrap estimate by assigning a random weight on each corresponding item of the sample data. Both of the

schemes will be used to estimate Bhattacharyya bounds and error rates of classifiers in the next two chapters. They will also be applied to construct classifiers in Chapter 5. It will be seen that the performances of the Bayesian bootstrap and random weighting methods are very similar, although they are different from that of the basic bootstrap.

CHAPTER 3

THE BHATTACHARYYA BOUND

3.1 Introduction

In a statistical pattern recognition problem, a random vector $\underline{\mathbf{X}}$, called a pattern, is composed of a pair

$$\underline{\mathbf{X}} = (\underline{\mathbf{V}}, \omega), \quad (3.1)$$

where $\underline{\mathbf{V}}$ is a d -dimension vector called the feature vector, and ω is a variable representing a class. Generally, a feature vector $\underline{\mathbf{V}}$ can take a continuous value in a d -dimension space and a class variable ω can take one of a finite set of values. In this thesis, we consider only the case when there are two class cases, i.e. $\omega \in \{1, 2\}$. The priori probabilities of the two classes are assumed known,

$$\mathbf{P}_1 = \mathbf{P}(\omega = 1), \text{ and } \mathbf{P}_2 = \mathbf{P}(\omega = 2).$$

For convenience, in later discussions it is always assumed that the prior probabilities of the two classes are equal, and so,

$$\mathbf{P}_1 = \mathbf{P}_2 = 1/2. \quad (3.2)$$

The conditional density of a feature vector $\underline{\mathbf{V}}$ given ω is denoted as $\mathbf{p}_\omega(\underline{\mathbf{V}})$, $\omega = 1, 2$.

According to the Bayesian theorem, the posterior probability of j given $\underline{\mathbf{V}}$ is

$$\mathbf{q}_\omega(\underline{\mathbf{V}}) = \frac{\mathbf{P}_\omega \mathbf{p}_\omega(\underline{\mathbf{V}})}{\mathbf{p}(\underline{\mathbf{X}})}, \quad \omega = 1, 2 \quad (3.3)$$

where $\mathbf{p}(\underline{\mathbf{X}})$ is the mixed density function and is a constant independent of ω .

The purpose of pattern recognition is to determine whether a given feature vector $\underline{V}$ belongs to class 1 or 2. In other words, the aim is to predict the value of ω from a given feature vector $\underline{V}$. A decision rule based on probabilities is to maximize the posterior probability with the given feature vector $\underline{V}$. Thus, the classification rule is:

$$\begin{aligned}\omega &= 1 && \text{if } q_1(\underline{V}) > q_2(\underline{V}); \\ \omega &= 2 && \text{if } q_1(\underline{V}) < q_2(\underline{V}).\end{aligned}\tag{3.4}$$

Because $p(\underline{x})$ is positive, from (3.3), the above rule can be equally expressed as

$$\begin{aligned}\omega &= 1 && \text{if } \frac{p_1(\underline{V})}{p_2(\underline{V})} > \frac{P_2}{P_1}; \\ \omega &= 2 && \text{if } \frac{p_1(\underline{V})}{p_2(\underline{V})} < \frac{P_2}{P_1}.\end{aligned}\tag{3.5}$$

The rule expressed by (3.4) or (3.5) is called the Bayesian Decision Rule. When a feature vector $\underline{V}$ is classified to 1 by the Bayesian Decision Rule, it means that there is a better chance of having $\omega = 1$. This does not mean, however, that ω is exactly equal to 1, as there is always a possibility of making an error in the classification process. The conditional error probability caused by rule (3.4) or (3.5) is

$$r(\underline{V}) = \min\{q_1(\underline{V}), q_2(\underline{V})\}.\tag{3.6}$$

The Bayesian error is the average of $r(\underline{V})$,

$$\begin{aligned}\varepsilon &= E\{r(\underline{V})\} = \int r(\underline{V})p(\underline{V})d\underline{V} \\ &= \int \min\{P_1p_1(\underline{V}), P_2p_2(\underline{V})\}d\underline{V}.\end{aligned}\tag{3.7}$$

Equation (3.7) is the crucial measurement for the performance of the decision rule. Unfortunately, it is generally very hard to directly calculate a Bayesian error even if the conditional probabilities $p_1(\mathbf{V})$ and $p_2(\mathbf{V})$ are known. Therefore, many researches have devised upper bounds for (3.7) which are easier to calculate and estimate. Two such bounds, the Chernoff and Bhattacharyya bounds, are the most well-known ones and will be discussed in the second section. As the chapter title depicts, the Bhattacharyya bound is the main topic discussed here. Thus, the third section will consider the general approach of how to estimate the Bhattacharyya bound and the theoretical properties of the estimate. Simulation results will be given in order to help us understand the properties of the estimates. In Sections 3.4 and 3.5, we shall attempt to estimate the Bhattacharyya bound by utilizing bootstrap techniques. The algorithms of the bootstrap approaches are illustrated there along with the discussion of the simulation results. Although the problem of estimating the Chernoff bound is not mentioned here; it is easy to see that the approaches of the Bhattacharyya bound estimations discussed here could be applied to the Chernoff bound as well.

The data set used for all the experiments in this chapter is the set referred to as Data Set I, which consists of training samples for seven classes. Each class has a 2-dimension normal distribution, and a training sample size of eight. Only two classes are involved in each experiment: one is the class A, and the other is selected from the rest. Hence, there are, in total, six experimental pairs of classes, (A, B), (A, C), (A, D), (A, E), (A, F), and (A, G). With each class pair, an experiment will repeatedly do 200 trials of

simulation for an algorithm. The results of the experiments given in this chapter are the statistics from the 200 trials.

As only the case of two classes is discussed in this thesis, for simplification, it is preferred to use the notation $\underline{X}_{i, \omega} = (\underline{V}, \omega)$. So $\underline{X}_{i,1}$ represents sample data from class 1, and $\underline{X}_{i,2}$ represents a sample data from class 2.

3.2 Chernoff and Bhattacharrya Bounds

3.2.1 Chernoff Bound

For any real numbers, $a, b \geq 0$, it is well known that the following inequality holds:

$$\min \{a, b\} \leq a^s b^{1-s}, \quad 0 \leq s \leq 1. \quad (3.8)$$

Applying inequality (3.8) to (3.7), we have

$$\int \min \{P_1 p_1(\underline{V}), P_2 p_2(\underline{V})\} d\underline{V} \leq P_1^s P_2^{1-s} \int p_1^s(\underline{V}), p_2^{1-s}(\underline{V}) d\underline{V}.$$

The right side of the above inequality

$$\epsilon_u = P_1^s P_2^{1-s} \int p_1^s(\underline{V}), p_2^{1-s}(\underline{V}) d\underline{V}. \quad (3.9)$$

is called the Chernoff bound. The optimum s is the value that minimizes the value of ϵ_u .

When the conditional density functions are a normal distribution

$$p_j(\underline{V}) = N_j(\underline{M}_j, \Sigma_j), \quad j = 1, 2.$$

The integration part of (3.9) can be expressed as

$$\int p_1^s(\underline{V}), p_2^{1-s}(\underline{V}) d\underline{V} = e^{-\mu(s)}, \quad (3.10)$$

where

$$\begin{aligned} \mu(s) = & \frac{s(1-s)}{2} (\underline{\mathbf{M}}_1 - \underline{\mathbf{M}}_2)^T [s \underline{\Sigma}_1 + (1-s) \underline{\Sigma}_2]^{-1} (\underline{\mathbf{M}}_1 - \underline{\mathbf{M}}_2) \\ & + \frac{1}{2} \ln \frac{|s \underline{\Sigma}_1 + (1-s) \underline{\Sigma}_2|}{|\underline{\Sigma}_1|^s |\underline{\Sigma}_2|^{1-s}}. \end{aligned} \quad (3.11)$$

The quantity $\mu(s)$ is called the Chernoff distance. It is obvious that the optimum s will maximize the value of $\mu(s)$, which can be obtained by studying the variation of $\mu(s)$ for various values of s for the given $\underline{\mathbf{M}}_j$ and $\underline{\Sigma}_j$ ($j = 1, 2$).

3.2.2 Bhattacharyya Bound

The Bhattacharyya bound results when the Chernoff bound is simplified by selecting the value $s = 1/2$. In such a case, (3.9) is simplified to

$$\varepsilon_u = \sqrt{\mathbf{P}_1 \mathbf{P}_2} \int \sqrt{\mathbf{p}_1(\underline{\mathbf{V}}) \mathbf{p}_2(\underline{\mathbf{V}})} d\underline{\mathbf{V}} = \sqrt{\mathbf{P}_1 \mathbf{P}_2} e^{-\mu(1/2)}. \quad (3.12)$$

The upper bound given by (3.12) is called the Bhattacharyya bound. In our case, it is assumed that $\mathbf{P}_1 = \mathbf{P}_2 = 1/2$; therefore, the Bhattacharyya bound will take a form of:

$$\varepsilon_u = \frac{1}{2} \int \sqrt{\mathbf{p}_1(\underline{\mathbf{V}}) \mathbf{p}_2(\underline{\mathbf{V}})} d\underline{\mathbf{V}} = \frac{1}{2} e^{-\mu(1/2)}. \quad (3.13)$$

Correspondingly, the Bhattacharyya distance is:

$$\begin{aligned} \mu(1/2) = & \frac{1}{8} (\underline{\mathbf{M}}_1 - \underline{\mathbf{M}}_2)^T \left(\frac{\underline{\Sigma}_1 + \underline{\Sigma}_2}{2} \right)^{-1} (\underline{\mathbf{M}}_1 - \underline{\mathbf{M}}_2) \\ & + \frac{1}{2} \ln \frac{|\frac{\underline{\Sigma}_1 + \underline{\Sigma}_2}{2}|}{\sqrt{|\underline{\Sigma}_1| |\underline{\Sigma}_2|}}. \end{aligned} \quad (3.14)$$

There are two terms on the right side of the equation in (3.14), as well as (3.11). It is obvious that the first term will be zero if $\underline{\mathbf{M}}_1 = \underline{\mathbf{M}}_2$ and the second term will be zero if $\underline{\Sigma}_1 = \underline{\Sigma}_2$. So the first term reflects the class separability, which is caused by the difference of the means; while the second term reflects the class separability caused by the difference of the covariances. It is not difficult to calculate the Bhattacharyya bound in a normal distribution case when the parameters $\underline{\mathbf{M}}_j$ and $\underline{\Sigma}_j$ ($j = 1, 2$) are given.

3.3 The Estimation of Bhattacharyya Bound

In most cases, the means and covariances are unknown; even though the conditional distributions can be answered to be normal. In such cases, it is necessary to estimate the Bhattacharyya bound from a given training sample. Let $\mathbf{X} = \{\underline{\mathbf{X}}_{1,1}, \underline{\mathbf{X}}_{2,1}, \dots, \underline{\mathbf{X}}_{m,1}, \underline{\mathbf{X}}_{1,2}, \dots, \underline{\mathbf{X}}_{n,2}\}$ be the training sample: the first m sample data $\underline{\mathbf{X}}_{1,1}, \underline{\mathbf{X}}_{2,1}, \dots, \underline{\mathbf{X}}_{m,1}$ belongs to class 1 with a normal conditional distribution

$$\underline{\mathbf{X}}_{1,1}, \dots, \underline{\mathbf{X}}_{m,1} \sim N_1(\underline{\mathbf{M}}_1, \underline{\Sigma}_1), \quad (3.15)$$

and the other n sample data $\underline{\mathbf{X}}_{1,2}, \dots, \underline{\mathbf{X}}_{n,2}$ belongs to class 2 with a normal conditional distribution

$$\underline{\mathbf{X}}_{1,2}, \dots, \underline{\mathbf{X}}_{n,2} \sim N_2(\underline{\mathbf{M}}_2, \underline{\Sigma}_2). \quad (3.16)$$

Then the maximum likelihood estimates of the means $\underline{\mathbf{M}}_j$ and covariances $\underline{\Sigma}_j$ ($j = 1, 2$) are given by

$$\hat{\underline{\mathbf{M}}}_1 = \bar{\underline{\mathbf{X}}}_1 = \frac{1}{m} \sum_{i=1}^m \underline{\mathbf{X}}_{i,1},$$

$$\begin{aligned}\hat{\Sigma}_1 &= \frac{1}{m-1} \sum_{i=1}^m (\mathbf{X}_{i,1} - \hat{\mathbf{M}}_1)(\mathbf{X}_{i,1} - \hat{\mathbf{M}}_1)^T, \\ \hat{\mathbf{M}}_2 &= \bar{\mathbf{X}}_2 = \frac{1}{n} \sum_{i=1}^m \mathbf{X}_{i,2}, \\ \hat{\Sigma}_2 &= \frac{1}{n-1} \sum_{i=1}^n (\mathbf{X}_{i,2} - \hat{\mathbf{M}}_2)(\mathbf{X}_{i,2} - \hat{\mathbf{M}}_2)^T.\end{aligned}\tag{3.17}$$

It is well known that the estimates given by (3.17) have good properties, and are the optimized estimates of the parameters $\mathbf{M}_j$ and Σ_j ($j = 1, 2$). Based on (3.17), an estimate of the Bhattacharyya bound can be calculated by the following steps:

1. Replace the parameters in (3.14) with their estimates given by (3.17) to get an estimate of Bhattacharyya distance, say $\hat{\mu}(1/2)$.
2. Replace $\mu(1/2)$ in (3.12) with $\hat{\mu}(1/2)$ to get an estimate of the Bhattacharyya bound.

We refer to the estimating method described above as the *Direct Estimating approach* or the *General approach*. The unanswered question is one of determining how good the estimate of the General approach will be? The answer to this question is by no means trivial or direct because the Bhattacharyya distance and bound are fairly complex functions of the parameters $\mathbf{M}_j$ and Σ_j .

Here is a brief discussion on the estimate properties of a function of the parameters. Let $\underline{\theta} = (\theta_1, \theta_2, \dots, \theta_t)^T$ be a parameter vector, its estimate be $\hat{\underline{\theta}} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_t)^T$, and $\mathbf{f} = \mathbf{f}(\underline{\theta})$ be a function of $\underline{\theta}$. Using the general approach, an estimator of $\hat{\mathbf{f}} =$

$f(\hat{\underline{\theta}})$ can be obtained. Assuming that $\underline{\theta}$ and $\hat{\underline{\theta}}$ are close enough, a Taylor series can be used by expanding $\hat{f} = f(\hat{\underline{\theta}})$ up to the second order terms,

$$\hat{f} = f(\hat{\underline{\theta}}) \cong f(\underline{\theta}) + \frac{\partial f}{\partial \underline{\theta}}^T \Delta \underline{\theta} + \frac{1}{2} \text{tr} \left(\frac{\partial^2 f}{\partial \underline{\theta}^2} \Delta \underline{\theta} \Delta \underline{\theta}^T \right), \quad (3.18)$$

where $\Delta \underline{\theta} = \hat{\underline{\theta}} - \underline{\theta}$. If $\hat{\underline{\theta}}$ is an unbiased estimate of $\underline{\theta}$, $E(\Delta \underline{\theta}) = 0$. Thus,

$$E(\hat{f}) \cong f(\underline{\theta}) + \frac{1}{2} \text{tr} \left(\frac{\partial^2 f}{\partial \underline{\theta}^2} E\{\Delta \underline{\theta} \Delta \underline{\theta}^T\} \right). \quad (3.19)$$

We, therefore, have $\hat{f}$ which is generally a biased estimator of f . In our case

$$\underline{\theta} = (\underline{\mathbf{M}}_1^T, \underline{\mathbf{M}}_2^T, \text{svec}(\underline{\Sigma}_1)^T, \text{svec}(\underline{\Sigma}_2)^T)^T,$$

where $\text{svec}(\underline{\mathbf{A}})$ represents a vector consisting of all different components of a symmetric matrix $\underline{\mathbf{A}} = (a_{ij})_{n \times n}$,

$$\text{svec}(\underline{\mathbf{A}}) = (a_{11}, \dots, a_{n1}, a_{22}, \dots, a_{n2}, \dots, a_{nn}),$$

and its estimator

$$\hat{\underline{\theta}} = (\hat{\underline{\mathbf{M}}}_1^T, \hat{\underline{\mathbf{M}}}_2^T, \text{svec}(\hat{\underline{\Sigma}}_1)^T, \text{svec}(\hat{\underline{\Sigma}}_2)^T)^T$$

given in (3.17) is unbiased. For the Bhattacharyya distance, the second term on the right side in (3.19) is extremely complicated (the details will not be discussed here as it is beyond the scope of this thesis. The reader can refer to Fukunaga's book [Fu92] for more details). It is thus apparent that the estimate of the Bhattacharyya distance deduced from the General approach is biased, and so is the Bhattacharyya bound. The simulation results given below support the conclusion, and give a visual illustration of how biased the estimate of the General approach can be.

3.3.1. Simulation Results

As we know the parameters of the multi-normal distribution of each class, it is not difficult to calculate the theoretical values of the Bhattacharyya distances and the bounds of each class pair involved in an experiment. These theoretical values are given in TABLE 3.1. Note that the Bhattacharyya bounds are given in terms of a percentage, i.e.

$$\text{Bhattacharyya bound} = \frac{1}{2} e^{-\mu(1/2)} \times 100\%.$$

TABLE 3.1 Theoretical values of Bhattacharyya distance and bound

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Bhattacharyya distance	0.0625	0.1375	0.16298	0.232513	0.3125	0.419263
Bhattacharyya bound	46.9707	43.5767	42.4804	39.6270	36.5808	32.8766

As all the multi-normal distributions of the classes have the same covariance matrix, the identical matrix, the Bhattacharyya distance of a class pair in TABLE 3.1 contains only the first term on the right side of (3.14). It is seen to be in proportion to the square of the Euclidean distance of the expectations of the class pair. Thus, the results of the experiments reflect, in some sense, the effect of the classifying error rate because the class pair changes in each experiment, and so does the classifier error rate.

TABLE 3.2 Estimates of Bhattacharyya distances with the General approach

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	0.0625	0.1375	0.16298	0.232513	0.3125	0.419263
Means	0.236728	0.407315	0.458343	0.758463	1.122263	2.000085
Standard Deviations	0.138292	0.247854	0.271062	0.405374	0.558753	1.026716

TABLE 3.2 lists the means and standard deviations of the 200 trials of the Bhattacharyya distance estimates. It can be seen from the table that, on average, the Bhattacharyya distance estimate is biased, i.e. the General approach usually overestimates the Bhattacharyya distance. The bias increases when the distance of two classes

increases. The standard deviation of the estimate increases along with the increased distance of the two classes as well.

TABLE 3.3 Estimates of Bhattacharyya bounds with the General approach (%)

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Means	39.82136	34.17429	32.66137	25.14584	18.486	9.542846
Standard Deviations	5.177134	7.171909	7.634124	8.511053	8.14592	6.466988
Maximum	49.139	48.38373	48.92767	43.63775	39.1444	29.63764
90%	46.0513	42.66327	42.31919	35.69748	29.33709	18.65459
80%	44.45931	39.93726	39.36365	32.32033	25.18118	15.29243
Median	40.47395	35.12138	33.43917	26.226	18.80952	8.908418
20%	32.04913	24.81682	22.53725	12.84652	7.013423	1.792992
10%	30.74716	21.76738	20.07424	10.15759	6.252015	1.142376
Minimum	23.42602	7.285027	9.03749	6.522464	1.377495	0.018612

TABLE 3.3 provides the statistics of the 200 trials of the Bhattacharyya distance estimates in terms of percentages. Besides the means and standard deviations of the 200 estimates, the maximum, median, minimum, and 90%, 80%, 20% and 10% quantiles of the estimates are also listed in the table. For instance, the mean and median of the 200 estimates of the Bhattacharyya bound for the class pair (A, E) are 25.14584% and 26.226% respectively, the 90% quantile of them is 35.69748%. Comparing them to the theoretical value which is 39.627%, we can see that, for the class pair (A, E), the General approach underestimates the quantile significantly. It is expected that the General approach tends to under-estimate the Bhattacharyya bound, because the estimate of the Bhattacharyya bound is just an exponential function of the negative estimate of the Bhattacharyya distance. Of course, the bias of the estimate increases, while the distance of the two classes increases. One interesting fact is that with the class pair (A, B), the rate of the estimates which are under-estimating the Bhattacharyya bound is about 90% of the

total 200 trials, while for the class pair (A, G), the rate increases to 100%. We infer that, generally speaking, the probability of under-estimating the Bhattacharyya bound usually increases in proportion to the increase of the distance of two classes.

In the case of the Bhattacharyya distance, the cause for the over-estimation is due to the second term in (3.19) or the third term in (3.18). Observe that the Bhattacharyya bound is a function of the Bhattacharyya distance $\mu(1/2)$, and $\hat{\mu}(1/2)$ given by the General approach, is a biased estimate of $\mu(1/2)$. Therefore, the cause for under-estimating is not only due to the third term in (3.18), but also due to the second term in (3.18). Therefore, the question before us is one of correcting the bias, and as we shall see, the bootstrap technique provides a way of achieving this. The next sections will illustrate how the bootstrap technique can be applied when estimating the Bhattacharyya bound.

3.4 The Bootstrap Estimators — Direct Bias Correction

When considering the problem of estimating the Bhattacharyya bound, there are two different types of schemes available by which a bootstrap estimate of the quantity can be obtained. The difference between the two schemes is a question of how we want to correct the bias. First, the estimate bias may be directly corrected by using (2.9). Schemes of this type are called the *Direct Bias Correction* schemes. Second, since the estimate bias of the Bhattacharyya bound is caused by the estimate bias of the Bhattacharyya distance, the bias of the Bhattacharyya distance can be estimated first, and then the bias estimate can be used to adjust the estimate of the Bhattacharyya bound. Schemes of this

type are called the *Distance Bias Adjustment* schemes. This section will discuss the Direct Bias Correction schemes with three different resampling approaches, the basic bootstrap, Bayesian bootstrap and the random weighting method. A discussion of the distance bias adjustment schemes will be the topic of the next section.

The direct bias correction schemes use directly the bootstrap estimate of the Bhattacharyya bound to correct the bias, i.e. to apply (2.9) given in Chapter 2. The algorithms with different resampling schemes are described below:

Algorithm 3.1 Basic Bootstrap

Input: (i) n , the size of the training sample of a class;
(ii) $\underline{x}[1, 1], \underline{x}[2, 1], \dots, \underline{x}[n, 1]$ the training sample data of the first class;
(iii) $\underline{x}[1, 2], \underline{x}[2, 2], \dots, \underline{x}[n, 2]$ the training sample data of the second class;
(iv) B , the repeated times of the bootstrap resampling.

Output: The Bhattacharyya bound estimate.

Method
BEGIN
 $\varepsilon_0 = \text{CalculateBound}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2])$
 $\varepsilon = 0$
For ($i = 1$ to B)
 For ($j = 1$ to n)
 $m = \text{random integer in } [1, n]$
 $\underline{y}[j, 1] = \underline{x}[m, 1]$
 End-For
 For ($j = 1$ to n)
 $m = \text{random integer in } [1, n]$
 $\underline{y}[j, 2] = \underline{x}[m, 2]$
 End-For
 $\varepsilon = \varepsilon + \text{CalculateBound}(\underline{y}[1, 1], \dots, \underline{y}[n, 1], \underline{y}[1, 2], \dots, \underline{y}[n, 2])$
End-For
 $\varepsilon = 2 \times \varepsilon_0 - \varepsilon / B$
Return ε
END Basic Bootstrap

Procedure CalculateBound

Input: (i) n , the size of the sample of a class;
(ii) $\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1]$ the sample data of the first class;
(iii) $\underline{z}[1, 2], \underline{z}[2, 2], \dots, \underline{z}[n, 2]$ the sample data of the second class;

Output: A Bhattacharyya bound estimate.

Method

BEGIN

$$\mu(1/2) = \text{CalculateDistance}(\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1], \underline{z}[1, 2], \dots, \underline{z}[n, 2])$$

$$\varepsilon = 0.5 * \text{EXP}(-\mu(1/2))$$

Return ε

END Procedure CalculateBound

Procedure CalculateDistance

Input: (i) n , the size of the sample of a class;
(ii) $\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1]$ the sample data of the first class;
(iii) $\underline{z}[1, 2], \underline{z}[2, 2], \dots, \underline{z}[n, 2]$ the sample data of the second class;

Output: a Bhattacharyya bound estimate.

Method

BEGIN

$$\bar{\underline{z}}[1] = \frac{1}{n} \sum_{j=1}^n \underline{z}[j, 1]$$

$$\Sigma_1 = \frac{1}{n-1} \sum_{j=1}^n (\underline{z}[j, 1] - \bar{\underline{z}}[1])(\underline{z}[j, 1] - \bar{\underline{z}}[1])^T$$

$$\bar{\underline{z}}[2] = \frac{1}{n} \sum_{j=1}^n \underline{z}[j, 2]$$

$$\Sigma_2 = \frac{1}{n-1} \sum_{j=1}^n (\underline{z}[j, 2] - \bar{\underline{z}}[2])(\underline{z}[j, 2] - \bar{\underline{z}}[2])^T$$

$$\mu(1/2) = \frac{1}{8} (\bar{\underline{z}}[1] - \bar{\underline{z}}[2])^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\bar{\underline{z}}[1] - \bar{\underline{z}}[2]) + \frac{1}{2} \ln \frac{|\frac{\Sigma_1 + \Sigma_2}{2}|}{\sqrt{|\Sigma_1| |\Sigma_2|}}$$

Return $\mu(1/2)$

END Procedure CalculateDistance

The first step of this algorithm is to calculate the general estimate of the Bhattacharyya bound. The first **For** loop is used to calculate the B bootstrap estimates of the Bhattacharyya bound. After that, (2.9) is applied to obtain the estimate of the

Bhattacharyya bound. The two **For** loops nested in the first **For** loop are for generating bootstrap samples, so the bootstrap sample is drawn separately from each training sample. The reason for doing this is so that we can have a larger probability to ensure that the estimated covariance matrixes $\hat{\Sigma}_1$ and $\hat{\Sigma}_2$ are non-singular. If a bootstrap sample is drawn from the merged training sample, the bootstrap sample may only contain a few data points from one class sample. Thus, the estimated covariance matrixes $\hat{\Sigma}_1$ or $\hat{\Sigma}_2$ have a higher probability of being singular, which may happen, especially, in the case of small training samples, or when the dimension of the feature vectors is “large”. Considering that an estimate of a covariance matrix may be singular, a real algorithm should inspect the three determinants $|\Sigma_1|$, $|\Sigma_2|$ and $|\Sigma_1 + \Sigma_2|$ after each bootstrap sampling. If any of them is zero or significantly close to zero, a bootstrap sample must be drawn and inspected again until all of the three determinants are non-zero.

By changing the way of drawing the bootstrap samples, the other two estimating algorithms for the Bhattacharyya bound, the Bayesian bootstrap and the random weighting algorithms can be obtained. The algorithm of the Bayesina bootstrap is presented below.

Algorithm 3.2 Bayesian bootstrap

Input: (i) n , the size of the training sample of a class;
(ii) $\underline{x}[1, 1], \underline{x}[2, 1], \dots, \underline{x}[n, 1]$ the training sample data of the first class;
(iii) $\underline{x}[1, 2], \underline{x}[2, 2], \dots, \underline{x}[n, 2]$ the training sample data of the second class;
(iv) B , the repeated times of the bootstrap resampling.

Output: The Bhattacharyya bound estimate.

Method
BEGIN
 $\varepsilon_0 = \text{CalculateBound}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{general})$
 $\varepsilon = 0$
For ($i = 1$ to B)
 bootstrap-bound = bootstrap-bound
 + $\text{CalculateBound}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{bootstrap})$
End-For
 $\varepsilon = 2 \times \varepsilon_0 - \varepsilon / B$
Return ε
END Bayesian bootstrap

Procedure CalculateBound

Input: (i) n , the size of the sample of a class;
(ii) $\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1]$ the sample data of the first class;
(iii) $\underline{z}[1, 2], \underline{z}[2, 2], \dots, \underline{z}[n, 2]$ the sample data of the second class;
(iv) flag to indicate 'general' or 'bootstrap'

Output: A Bhattacharyya bound estimate.

Method
BEGIN
 $\mu(1/2) = \text{CalculateDistance}(\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1], \underline{z}[1, 2], \dots, \underline{z}[n, 2], \text{flag})$
 $\varepsilon = 0.5 * \text{EXP}(-\mu(1/2))$
Return ε
END Procedure CalculateBound

Procedure CalculateDistance

Input: (i) n , the size of the sample of a class;
(ii) $\underline{z}[1, 1], \underline{z}[2, 1], \dots, \underline{z}[n, 1]$ the sample data of the first class;
(iii) $\underline{z}[1, 2], \underline{z}[2, 2], \dots, \underline{z}[n, 2]$ the sample data of the second class;

Output: A Bhattacharyya bound estimate.

Method**BEGIN**

If flag = 'bootstrap'

GetWeights_Bayesian_Bootstrap(n)

Else

$w[1] = w[2] = \dots = w[n] = 1/n$

End-If

$\bar{\underline{z}}[1] = \sum_{j=1}^n w[j] \underline{z}[j, 1]$

$\Sigma_1 = \sum_{j=1}^n w[j] (\underline{z}[j, 1] - \bar{\underline{z}}[1]) (\underline{z}[j, 1] - \bar{\underline{z}}[1])^T$

If flag = 'bootstrap'

GetWeights_Bayesian_Bootstrap(n)

Else

$w[1] = w[2] = \dots = w[n] = 1/n$

End-If

$\bar{\underline{z}}[2] = \sum_{j=1}^n w[j] \underline{z}[j, 2]$

$\Sigma_2 = \sum_{j=1}^n w[j] (\underline{z}[j, 2] - \bar{\underline{z}}[2]) (\underline{z}[j, 2] - \bar{\underline{z}}[2])^T$

$$\mu(1/2) = \frac{1}{8} (\bar{\underline{z}}[1] - \bar{\underline{z}}[2])^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\bar{\underline{z}}[1] - \bar{\underline{z}}[2]) + \frac{1}{2} \ln \frac{|\frac{\Sigma_1 + \Sigma_2}{2}|}{\sqrt{|\Sigma_1| |\Sigma_2|}}$$

Return $\mu(1/2)$

END Procedure CalculateDistance

Procedure GetWeights_Bayesian_Bootstrap**Input:** n , the size of the training sample of a class;**Output:** $w[1], w[2], \dots, w[n]$, the resampling vector**Method****BEGIN****For** ($j = 1$ to $n - 1$) $w[j] = \text{random uniform } U[0, 1]$ **End-For** $w[n] = 1$ **Sort**($w[1], w[2], \dots, w[n]$)**For** ($j = n$ to 2) $w[j] = w[j] - w[j - 1]$ **End-For****Return** $w[1], w[2], \dots, w[n]$ **END Procedure GetWeights_Bayesian_Bootstrap**

The goal of the procedure **GetWeights_Bayesian_Bootstrap** above is to set the resampling vector, and the **Sort** procedure is any standard sorting procedure which puts $w[1], w[2], \dots, w[n]$ in an increasing order. The main difference between **Algorithm 3.1** and **Algorithm 3.2** is the **CalculateDistance** procedure. In **Algorithm 3.1**, the bootstrap sample is passed to the procedure to calculate the estimates of the means and covariances, and so the weights used in the calculation are always $1/n$. In **Algorithm 3.2**, the parameters passed to the **CalculateDistance** procedure are always the training sample and a flag. If the flag is 'bootstrap', the procedure will invoke **GetWeights_Bayesian_Bootstrap** to obtain the weights for calculating the corresponding bootstrap estimate. Otherwise, it uses equivalent weights $1/n$ to calculate the estimates, as is done in **Algorithm 3.1**.

As it can be directly seen, **Algorithm 3.2** is more general than **Algorithm 3.1**. **Algorithm 3.1** can be obtained from **Algorithm 3.2** by just changing the

GetWeights_Bayesian_Bootstrap procedure to the **GetWeights_Basic_Bootstrap** procedure, as shown below:

Algorithm 3.3. GetWeights_Basic_Bootstrap

Procedure GetWeights_Basic_Bootstrap

Input: n , the size of the training sample of a class;

Output: $w[1], w[2], \dots, w[n]$, the resampling vector

Method

BEGIN

For ($j = 1$ to n)

$w[j] = 0$

End-For

For ($j = 1$ to n)

$m = \text{random integer in } [1, n]$

$w[m] = w[m] + 1$

End-For

For ($j = 1$ to n)

$w[j] = w[j] / n$

End-For

Return $w[1], w[2], \dots, w[n]$

END Procedure GetWeights_Basic_Bootstrap

From what was explained earlier, the algorithm for the random weighting method can be obtained from **Algorithm 3.2** by changing the **GetWeights_Bayesian_Bootstrap** procedure as well. As the other sections of the algorithm of the random weighting method are exactly the same as that of **Algorithm 3.2**, only the changed procedure which is now the **GetWeights_Random_Weighting** procedure of the random weighting method, is given here.

Algorithm 3.3 GetWeights_Random_Weighting**Procedure GetWeights_Random_Weighting****Input:** n , the size of the training sample of a class;**Output:** $w[1], w[2], \dots, w[n]$, the resampling vector**Method****BEGIN**

sum = 0

For ($j = 1$ to n) $w[j] = \text{random uniform } U[0, 1]$ sum = sum + $w[j]$ **End-For****For** ($j = 1$ to n) $w[j] = w[j] / \text{sum}$ **End-For****Return** $w[1], w[2], \dots, w[n]$ **END Procedure GetWeights_Random_Weighting****3.4.1.Simulation Results**

The simulation experiments were done with the above three algorithms and the statistics for them are listed in TABLE 3.4 – 4.6. The results are given in terms of percentages. For instance, in TABLE 3.4, the mean and median of the 200 estimates given by the Basic Bootstrap algorithm are 31.83517% and 32.58316% for the class pair (A, E). In TABLE 3.5, for the same class pair, the mean and median of the 200 estimates given by the Bayesian Bootstrap algorithm are 29.67647%, and 30.61675%. In TABLE 3.6, the mean and median given by the Random Weighting algorithm are 27.34608% and 28.27401%. Comparing these estimates to the corresponding estimate value given by the General approach, we do have less biased estimates. One noticeable thing is that for class pair (A, B), the 80% quantiles of all of the three algorithms are above the theoretical value of the Bhattacharyya bounds, while the 90% quantile of the General approach is still under the theoretical value of the Bhattacharyya bounds.

TABLE 3.4 Estimates of Bhattacharyya bounds with the Basic Bootstrap (%)

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	48.33761	42.15751	40.52867	31.8351	18.51542	12.6412
STD	5.942545	8.293123	8.835997	10.20969	14.41235	8.457236
Maximum	58.67531	58.84465	58.44089	55.32598	49.51755	37.1651
90%	55.50185	52.47521	51.50358	44.83127	37.95566	24.76845
80%	53.55391	49.2396	48.08842	40.64219	33.08389	19.96274
Median	49.89679	42.7631	41.23105	32.58316	19.41409	11.5166
20%	39.7612	32.01948	27.83918	16.56401	-0.317	2.481518
10%	37.15958	27.38551	26.60906	13.38149	-3.6393	1.527423
Minimum	30.43987	8.334777	12.07729	8.920697	-11.0023	0.020404

TABLE 3.5 Estimates of Bhattacharyya bounds with the Bayesian Bootstrap (%)

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Means	44.91215	39.29967	37.68654	29.67647	22.37645	11.87401
Standard Deviations	5.526691	7.718864	8.194747	9.441294	9.392886	7.868162
Maximum	55.07567	54.28726	54.21996	49.74175	45.16487	35.47709
90%	51.34415	48.68374	47.92519	41.36576	34.70719	22.85728
80%	50.02776	45.81377	44.98748	37.70615	30.52193	18.5203
Median	45.9033	39.97589	38.46402	30.61675	23.3626	11.13167
20%	36.63972	29.56032	26.53268	15.41786	9.069386	2.302328
10%	34.90043	25.20838	24.0665	12.64825	7.779151	1.528754
Minimum	28.28766	7.880876	11.71846	8.367018	1.818351	0.012888

TABLE 3.6 Estimates of Bhattacharyya bounds with the Random Weighting Method (%)

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Means	42.3308	36.65799	35.1367	27.34608	20.44759	10.71386
Standard Deviations	5.317276	7.391367	7.875731	8.917867	8.771155	7.196015
Maximum	51.68313	51.24357	51.34823	46.67435	42.29843	32.54132
90%	48.64045	45.7033	44.87394	38.15151	32.52625	20.85488
80%	47.11296	43.09073	41.96743	34.85278	27.6164	17.18219
Median	43.2103	37.4952	36.0776	28.27401	21.15995	9.937261
20%	34.18869	27.1069	24.33814	13.9911	8.127152	2.067278
10%	32.72992	24.08489	22.3457	11.45671	7.306231	1.311337
Minimum	25.69699	8.288063	10.0383	7.621519	1.543205	0.014472

As can be seen from the tables, the bootstrap techniques does work in this case. On average, the means of the 200 estimates for all the six class pairs improves to some degree. As can be seen, the percentage of the estimates with values over the theoretical value also increased. We note, however, that there are a few disadvantages to the three algorithms. The standard deviations of the estimates are larger than that of the General approach. In the experiment of the basic bootstrap algorithm with the class pair (A, F), about 20% of the estimates are even below zero, which is quite unacceptable. Of course, a restriction to the algorithm could be added to discard negative estimate values. A simple way of doing this is to just reject an estimate when a negative estimate value is reported, and to re-draw the bootstrap samples and calculate the estimate again. This procedure would be repeated until a nonnegative estimate is produced. Comparatively, the Bayesian bootstrap and random weighting method algorithms have smaller standard deviations and no negative estimate values.

3.5 The Bootstrap Estimators — Distance Bias Adjustment

As the Bhattacharyya bound ε_u is an exponential function of the Bhattacharyya distance $\mu(1/2)$, $\varepsilon_u = \frac{1}{2} e^{-\mu(1/2)}$, it is not difficult to rewrite it in terms of the Taylor series expanding ε_u as,

$$\varepsilon_u = \varepsilon_{u0} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \Delta\mu^k, \quad (3.20)$$

where $\varepsilon_{u0} = \frac{1}{2} e^{-\mu_0(1/2)}$, and $\Delta\mu = \mu - \mu_0$. Suppose ε_u is the theoretical value of the Bhattacharyya bound and ε_{u0} is an estimated value, then (3.20) provides a way of representing ε_u in terms of ε_{u0} if the difference $\Delta\mu = \mu - \mu_0$ is known. Fortunately, the difference $\Delta\mu = \mu - \mu_0$ is just the bias of the Bhattacharyya distance and can be estimated by using the bootstrap technique. Thus the distance bias adjustment schemes of the Bhattacharyya bound estimate can be effectively utilized. After obtaining the estimates of the Bhattacharyya bound $\hat{\varepsilon}_u$ and bias of the Bhattacharyya distance $\Delta\hat{\mu}$, a new estimate of $\hat{\varepsilon}_A$ is obtained by adjusting $\hat{\varepsilon}_u$ with $\Delta\hat{\mu}$, as follows:

$$\hat{\varepsilon}_A = \hat{\varepsilon}_u \times (1 - \Delta\hat{\mu} + \frac{1}{2} \Delta\hat{\mu}^2). \quad (3.21)$$

In (3.21), the approximation utilizes only up to the second order terms of the Taylor series. Observe that although it is feasible to use more terms of the Taylor series expansion, as the order of a term increases, the contribution of the item decreases. Therefore, adding more items into (3.21) will not generally effect the final estimate much.

It is still necessary to figure out some details before a formal algorithm can be obtained for the distance bias adjustment schemes. The first question to be answered is: what will be chosen as $\hat{\varepsilon}_u$? There are two possible options, namely, to use either a General approach estimate, or a bootstrap estimate. The scheme will be called a *G-based (general approach based)* estimate if the General approach estimate is selected, and called a *B-based (bootstrap based)* estimate if a bootstrap estimate is selected. The second is when the distance bias adjustment is performed. In a bootstrap procedure, B sets

of bootstrap samples are repeatedly drawn, an estimate is calculated for each bootstrap sample, and then these estimates are averaged to obtain the final bootstrap estimate. So the adjustment can be done at each step of the calculation of the bootstrap samples, or after the averaging. The first case is referred to as the AFMS (Adjustment First and Mean Second) approach, and the second case is referred to as the MFAS (Mean First and Adjustment Second) approach. It is obvious that the B-based estimate will definitely be effected by the selection of AFMS or MFAS strategy. The choice of AFMS or MFAS has an influence on the G-based estimate as well. Although the value of $\hat{\epsilon}_n$ is not effected by the bootstrap samples, as $\Delta\hat{\mu}^2$ is subject to the selection of AFMS or MFAS strategy, they will be correspondingly different. Given below are the algorithms of the distance bias adjustment schemes with AFMS and MFAS respectively.

3.5.1 AFMS (Adjustment First and Mean Second) Schemes

The main part of a generalized algorithm with the B-based AFMS strategy is listed below.

Algorithm 3.4 Distance Bias Adjustment – B-based AFMS

Input: (i) n , the size of the training sample of a class;
(ii) $\underline{x}[1, 1], \underline{x}[2, 1], \dots, \underline{x}[n, 1]$ the training sample data of the first class;
(iii) $\underline{x}[1, 2], \underline{x}[2, 2], \dots, \underline{x}[n, 2]$ the training sample data of the second class;
(iv) B , the repeated times of the bootstrap resampling.

Output: The Bhattacharyya bound estimate.

Method
BEGIN
 $\mu_0 = \text{CalculateDistance}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{general})$
 $\varepsilon_0 = 0.5 \times \text{EXP}(-\mu_0)$
 $\varepsilon = 0$
For ($i = 1$ to B)
 $\mu = \text{CalculateDistance}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{bootstrap})$
 $\Delta\mu = \mu - \mu_0$
 $\varepsilon = \varepsilon + (2 \times \varepsilon_0 - 0.5 \times \text{EXP}(-\mu)) \times (1 - \Delta\mu + 0.5 \times \Delta\mu^2)$
End-For
 $\varepsilon = \varepsilon / B$
Return ε
END-MAIN

The **CalculateDistance** procedure is exactly the same as that in **Algorithm 3.2**.

Therefore, the appropriate **GetWeights** procedure is also required in this algorithm. Of course, we have to use different **GetWeights** procedures to carry out either a basic bootstrap, Bayesian bootstrap or random weighting method scheme respectively. As each **GetWeights** procedure for the three bootstrap schemes has already been represented in Section 3.4, we omit the discussion of their details again.

In the algorithm above, the factor $(2\varepsilon_0 - 0.5 \times \text{EXP}(-\mu))$ is the bootstrap Bhattacharyya bound estimate for the currently drawn bootstrap sample, multiplied by the distance bias adjustment factor $(1 - \Delta\mu + 0.5 \times \Delta\mu^2)$. If $(2\varepsilon_0 - 0.5 \times \text{EXP}(-\mu))$ is replaced by ε_0 , and nothing else has been changed, it can be seen that the algorithm will become a G-based algorithm. Hence, we will not give another description separately for

the G-based and AFMS algorithms. With the same training sample data, experiments have been done for the B-based/G-based and AFMS algorithms with all of the three bootstrap schemes, namely, the basic bootstrap, the Bayesian bootstrap and the random weighting method. The statistics of the experiments are listed in TABLE 3.7 – 3.12.

TABLE 3.7 Estimates of Bhattacharyya bounds (%)
Basic Bootstrap, B-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	35.29617	31.89302	30.07561	26.28996	24.80948	20.63062
STD	15.28205	15.54667	7.296333	9.389851	24.24049	23.34433
Maximum	243.7531	207.0035	76.74092	107.5065	312.5824	283.3539
90%	38.76487	36.44793	36.27044	33.17568	33.20911	35.54862
80%	37.92635	34.51293	34.32448	30.24766	28.52795	25.68454
Median	34.46269	30.74694	30.12942	25.24358	21.20245	15.16199
20%	29.64628	24.34977	21.65784	17.98987	13.27882	8.732412
10%	27.23526	21.95029	20.4463	17.11695	11.51436	6.913009
Minimum	21.26367	15.21436	13.35947	7.644248	5.994843	2.454454

TABLE 3.8 Estimates of Bhattacharyya bounds (%)
Bayesian Bootstrap, B-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	37.54501	32.01081	30.43654	23.354	17.06923	9.294273
STD	4.440113	5.779325	6.587644	6.646145	6.343796	4.665902
Maximum	44.26512	43.12695	44.03085	39.06706	32.86951	23.54554
90%	42.97227	39.20749	38.53849	32.17855	25.71094	15.19422
80%	41.30967	37.0198	36.16435	29.33561	23.04196	13.61435
Median	38.31426	32.71642	31.01248	23.01215	17.25403	8.719324
20%	30.81623	24.76361	21.20125	14.85177	9.325499	3.80995
10%	28.80029	20.86961	17.90184	12.18928	5.799734	2.368375
Minimum	23.35308	14.83666	11.17671	6.761167	3.384352	0.617094

TABLE 3.9 Estimates of Bhattacharyya bounds (%)
Random Weighting Method, B-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	39.79466	34.1781	32.52961	25.13843	18.18391	9.316001
STD	4.630449	5.965157	6.878829	7.183616	7.181104	5.153994
Maximum	47.13392	45.76277	46.36956	41.06053	35.87448	25.25058
90%	45.45606	41.21526	40.64846	34.68022	27.83729	16.19525
80%	43.64801	39.33556	38.4615	31.65831	24.87642	13.89037
Median	40.56556	34.83044	33.39507	25.15333	18.67758	8.375691
20%	33.34305	26.20512	23.36485	16.01357	9.248596	3.016075
10%	30.90892	22.91647	19.32754	13.20445	5.328012	2.07233
Minimum	23.94745	16.23455	12.13985	7.275378	2.749891	0.376614

TABLE 3.7 – 3.9 list the results of the B-based AFMS schemes. The estimate given by the Basic Bootstrap algorithm with the B-based AFMS scheme listed in TABLE 3.7. For instance, the mean and median of the 200 estimates given by this algorithm are 26.2899% and 25.24358 for the class pair (A, E). For the same class pair, the mean and median of the 200 estimates given by the Bayesian Bootstrap algorithm are 23.354% and 23.01215% as listed in TABLE 3.8. In TABLE 3.9, for the same class pair, the mean and median are 25.13843% and 23.15333% respectively, as obtained with the Random Weighting algorithm. These indicate that the performances of the three bootstrap schemes are quite distinct. The basic bootstrap does not perform well with the class pairs (A, B), (A, C) and (A, D), while on average, the estimates of the basic bootstrap are higher than the General approach estimates with the class pairs (A, E), (A, F) and (A, G). It may also be noticed that the standard deviations of the basic bootstrap estimates are quite large. The cause of this is because some estimates are too high (even more than 100%), and so they are not reasonable. It is possible to reduce the standard deviations of estimates of this scheme by putting a restriction on the maximum estimated value of the algorithm.

With the B-based AFMS strategy, the Bayesian bootstrap and random weighting method schemes do not improve the General approach estimate much, except that they have smaller estimate deviations.

TABLE 3.10 Estimates of Bhattacharyya bounds (%)
Basic Bootstrap, G-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	41.11082	38.85462	36.8919	34.21707	37.32694	33.01038
STD	9.063351	15.95164	12.17684	19.99692	56.18202	43.78814
Maximum	154.2297	184.7436	150.1341	259.0628	759.5583	523.3873
90%	45.55304	44.52779	43.38001	45.43152	55.12306	57.11908
80%	44.38369	41.69149	40.72589	38.4882	40.48444	44.96434
Median	40.82089	37.27138	35.64145	31.64667	28.33757	22.48356
20%	35.40031	28.76217	26.77807	23.64481	18.4257	11.56859
10%	33.1682	27.0222	24.7717	21.33968	15.03369	8.713007
Minimum	28.26328	18.50479	15.16554	8.598911	6.976958	3.766313

TABLE 3.11 Estimates of Bhattacharyya bounds (%)
Bayesian Bootstrap, G-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	40.86336	34.99263	33.2107	25.69247	18.77965	10.36139
STD	4.82879	6.159856	7.092814	7.24333	6.981074	5.058096
Maximum	48.36243	47.50654	48.01179	42.32066	36.17906	25.25234
90%	46.73866	42.50228	41.79076	35.41971	28.3465	16.83113
80%	45.00507	40.23935	39.52222	32.11332	25.31579	14.83372
Median	41.77362	35.60061	33.85971	25.49169	18.89684	9.72574
20%	34.18005	27.09707	23.51851	16.66913	10.07797	3.912425
10%	31.19996	24.19316	19.42777	13.89853	6.757514	2.851593
Minimum	24.52091	16.4869	13.39581	7.800159	3.203021	0.451379

TABLE 3.12 Estimates of Bhattacharyya bounds (%)
Random Weighting Method, G-based AFMS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	41.451	35.65973	33.8914	26.15482	18.82925	9.461315
STD	4.822625	6.220326	7.161089	7.574049	7.552264	5.443392
Maximum	48.80662	48.0047	48.40474	42.86147	36.85711	25.59972
90%	47.31289	43.0634	42.49743	36.21686	29.09597	16.7321
80%	45.50021	40.95507	40.10739	33.02417	25.90956	14.12225
Median	42.37573	36.42466	34.75562	26.20663	19.19099	8.662151
20%	35.02211	27.3674	24.15503	16.70679	9.472335	2.736777
10%	31.85959	23.37162	20.186	13.23641	5.494032	1.976811
Minimum	25.19115	17.09629	12.86532	7.429857	2.736784	0.434437

The results of the G-base AFMS are listed through TABLE 3.10 – 3.12. In TABLE 3.10, for the class pair (A, E), the mean and median of the 200 estimates given by the Basic bootstrap algorithm are 34.21707% and 31.64667% respectively. For the same class pair, TABLE 3.11 gives the mean 25.69247% and median 25.49169%, as estimated by the Bayesian bootstrap algorithm. The corresponding mean and median given by the Random Weighting algorithm are 26.15482% and 26.20663% listed in TABLE 3.12. Compared to the General approach, the estimates tend to improve with all of the bootstrap schemes. The standard deviations of the basic bootstrap scheme are still too large because of some extremely high estimate values, while the standard deviations of the other two bootstrap schemes are smaller than that of the General approach estimates. On the average, however, the estimate bias of the Bayesian bootstrap and random weighting method is bigger than that of the basic bootstrap.

3.5.2 MFAS (Mean First and Adjustment Second) Schemes

The main part of the algorithm with the B-based AFMS strategy is presented below:

Algorithm 3.5 Distance Bias Adjustment – B-based MFAS

Input: (i) n , the size of the training sample of a class;
(ii) $\underline{x}[1, 1], \underline{x}[2, 1], \dots, \underline{x}[n, 1]$ the training sample data of the first class;
(iii) $\underline{x}[1, 2], \underline{x}[2, 2], \dots, \underline{x}[n, 2]$ the training sample data of the second class;
(iv) B , the repeated times of the bootstrap resampling.

Output: The Bhattacharyya bound estimate.

Method
BEGIN
 $\mu_0 = \text{CalculateDistance}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{general})$
 $\varepsilon_0 = 0.5 \times \text{EXP}(-\mu_0)$
 $\varepsilon = 0$
 $\mu = 0$
For ($i = 1$ to B)
 $\text{temp} = \text{CalculateDistance}(\underline{x}[1, 1], \dots, \underline{x}[n, 1], \underline{x}[1, 2], \dots, \underline{x}[n, 2], \text{bootstrap})$
 $\mu = \mu + \text{temp}$
 $\varepsilon = \varepsilon + 0.5 \times \text{EXP}(-\text{temp})$
End-For
 $\mu = \mu / B$
 $\Delta\mu = \mu - \mu_0$
 $\varepsilon = \varepsilon / B$
 $\varepsilon = (2\varepsilon_0 - \varepsilon) \times (1 - \Delta\mu + 0.5 \times \Delta\mu^2)$
Return ε
END-MAIN

Like **Algorithm 3.4**, the **CalculateDistance** procedure is the same as that in **Algorithm 3.2**. It is also possible to carry out the algorithm with different bootstrap schemes by selecting the proper **GetWeights** procedures respectively. The difference between **Algorithm 3.4** and **Algorithm 3.5** is that in the latter, the bootstrap estimates of the Bhattacharyya distance and bound are accumulated respectively each time a bootstrap sample is drawn. Then the averages of them are used to obtain the Bhattacharyya bound estimate $(2\varepsilon_0 - \varepsilon)$ and the estimate of the distance bias adjustment factor $(1 - \Delta\mu + 0.5 \times \Delta\mu^2)$. The final estimate is given by multiplying the two estimates. To get a G-based MFAS algorithm, we need only replace $(2\varepsilon_0 - \varepsilon)$ with ε_0 , and omit $\varepsilon = \varepsilon + 0.5 \times \text{EXP}(-$

temp) and related operations in **Algorithm 3.5**. Due to the obvious similarities, we feel that it is not necessary to present the formal code for the G-based MFAS algorithm.

The experiment statistics of the B-based/G-based MFAS algorithms are given in TABLE 3.13 – 3.18 with the three bootstrap schemes, the basic bootstrap, the Bayesian bootstrap and the random weighting method. TABLE 3.13 is the statistics for the basic bootstrap algorithm with the B-based MFAS schemes, which contains the mean and median of 17.65553% and 17.1264% respectively for the class pair (A, E). In TABLE 3.14, the mean and median of the same class pair are 21.73415% and 21.80673%, which are the statistics of the Bayesian bootstrap algorithm with the B-based MFAS schemes. In TABLE 3.15, the random weighting algorithm provided the mean and median of 24.55108% and 24.55135% respectively for the same class pair.

TABLE 3.13 Estimates of Bhattacharyya bounds (%)
Basic Bootstrap, B-based MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	31.75356	26.10766	24.52919	17.65553	11.82273	5.644303
STD	4.693594	5.853703	6.573252	6.412899	5.444358	3.239644
Maximum	39.78992	37.05428	37.09475	32.68469	27.9204	17.74331
90%	37.44745	33.28894	32.89474	26.34556	19.44508	10.04395
80%	36.15411	31.13656	30.51698	22.96689	16.55583	8.355129
Median	32.10064	26.39337	24.97604	17.1264	11.85741	4.83922
20%	25.39029	18.55317	15.42574	9.626741	5.197099	1.919836
10%	23.01557	14.71796	12.2922	7.838717	3.781602	1.471255
Minimum	14.50959	10.26381	6.866701	4.245385	1.597374	0.38931

TABLE 3.14 Estimates of Bhattacharyya bounds (%)
Bayesian Bootstrap, B-based MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	37.02005	31.12021	29.36038	21.73415	14.92415	6.704326
STD	4.805898	6.163128	7.020338	7.255609	6.854069	4.399026
Maximum	44.23898	42.91679	44.22529	38.8137	32.35773	21.23623
90%	43.00015	38.78015	37.71012	31.56844	24.22171	12.68715
80%	41.1839	36.61838	35.5281	28.20391	21.04442	10.70742
Median	37.79584	31.84929	29.94806	21.80673	15.11878	5.807963
20%	30.49735	23.25631	19.71188	12.32024	6.431077	1.644345
10%	27.31983	18.60559	15.80402	9.984579	3.219682	1.117016
Minimum	19.63683	13.67765	8.157827	5.483799	1.643343	0.167245

TABLE 3.15 Estimates of Bhattacharyya bounds (%)
Random Weighting Method, B-based MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	39.59857	33.91181	32.13099	24.55108	17.35893	8.372892
STD	4.732293	6.086491	7.049376	7.382788	7.324002	5.140045
Maximum	46.78544	45.55008	46.56028	41.08643	35.47229	24.64247
90%	45.28959	41.18236	40.54022	34.406	27.31003	15.19041
80%	43.72768	39.05634	38.31622	31.04702	23.96976	12.87637
Median	40.43511	34.62662	32.90511	24.55135	17.62171	7.580274
20%	33.22375	25.94493	22.31655	15.30676	8.060977	2.309026
10%	30.02714	22.15524	18.72613	12.17501	4.435716	1.559719
Minimum	23.50559	15.6272	9.873922	6.648424	2.105345	0.218007

TABLE 3.16 – 3.18 give the statistics of the G-based MFAS schemes. The statistics of the basic bootstrap algorithm are listed in TABLE 3.16 which gives mean and median of 22.18424% and 21.586879% respectively for the class pair (A, E), for instance. The mean and median of the Bayesian bootstrap algorithm for the same class pair are 25.03786% and 24.84284% listed in TABLE 3.17. In TABLE 3.18, the mean and median of the random weighting algorithm for the same class pair are 26.24318% and 26.29509%.

TABLE 3.16 Estimates of Bhattacharyya bounds (%)
Basic Bootstrap, G-based MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	38.43518	32.07178	30.18913	22.18424	15.20784	7.534246
STD	5.210419	6.581673	7.401276	7.575083	6.817167	4.175871
Maximum	46.95407	44.17678	45.11681	40.08009	33.73086	21.25283
90%	44.42708	40.08821	39.25208	32.77827	24.87046	12.73597
80%	43.02655	37.84252	36.75578	28.93903	21.33118	11.13327
Median	39.06397	32.53594	31.00936	21.58687	15.04844	6.682093
20%	31.29004	23.81569	19.89397	12.71229	6.749445	2.491348
10%	28.29119	18.84137	16.34029	10.24023	5.714136	2.052147
Minimum	18.46924	13.20271	10.24718	5.659264	2.086469	0.384389

TABLE 3.17 Estimates of Bhattacharyya bounds (%)
Bayesian Bootstrap, G-based MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	40.92233	34.9339	33.06534	25.03786	17.48944	8.156097
STD	4.930905	6.36909	7.386044	7.787531	7.696632	5.238953
Maximum	48.48157	47.54306	48.11052	42.37459	36.15889	24.27576
90%	46.79528	42.58552	41.74863	35.38212	28.03542	15.25405
80%	45.08612	40.37395	39.32751	31.91701	24.80452	12.89838
Median	41.96837	35.45851	33.99645	24.84284	17.62768	7.174623
20%	34.279	26.28677	23.20067	15.07916	7.803587	1.893904
10%	30.79516	22.4292	18.75189	11.99698	4.299901	1.319445
Minimum	23.28211	16.16228	10.00416	6.362026	1.686659	0.215609

TABLE 3.18 Estimates of Bhattacharyya bounds (%)
Random Weighting Method, G-base MFAS

CLASS PAIR	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
Theoretical Value	46.9707	43.5767	42.4804	39.627	36.5808	32.8766
Mean	41.5418	35.78248	34.00968	26.24318	18.77967	9.197391
STD	4.797535	6.199704	7.179579	7.63339	7.747655	5.573234
Maximum	48.82647	47.96425	48.48275	42.98542	36.99365	26.06493
90%	47.37257	43.1945	42.51642	36.4239	29.26422	16.88564
80%	45.56246	41.05443	40.16449	33.11013	26.05862	14.12403
Median	42.47585	36.54791	34.91882	26.29059	19.0657	8.268751
20%	35.07868	27.60369	24.41967	16.75969	8.920452	2.358967
10%	31.93767	23.60087	20.13763	13.39481	5.079817	1.693297
Minimum	25.44232	17.30403	12.09235	7.45115	2.14916	0.237557

In conclusion, both the B-based and G-based MFAS algorithms have smaller estimate deviations, and tend to under-estimate the Bhattacharyya bound. Generally, they do not improve the estimate bias.

3.6 Summary

In this chapter, which mainly focused on the problem of estimating the Bhattacharyya bound for error analysis in classification, we showed both theoretically and experimentally that the estimate of the Bhattacharyya bound is biased. The experiments also showed that the larger the distance between two classes, the bigger the bias.

However Bootstrap techniques provide a formal solution by which we can reduce the estimate bias. Two kinds of bootstrap schemes were introduced to correct the estimate bias: one is the direct bias correction, and the other, the distance bias adjustment. The direct bias correction schemes are effective in reducing the estimate bias with all the three bootstrap schemes: the basic bootstrap, the Bayesian bootstrap and the random weighting method. The problems with these schemes are that the standard deviations of the estimates are big and some estimates are bad, especially, in the case of the basic bootstrap scheme.

The distance bias adjustment schemes come with different strategies depending on the choice of the base estimate, and the step of performing the distance bias adjustment. The experiment results showed that G-based algorithms are generally better than B-based ones. Although the MFAS related algorithms have smaller estimate deviations, they do not significantly reduce the estimate bias. The AFMS related algorithms have the

advantages of yielding good bias correctness for the more distant class pairs, especially, with the Basic Bootstrap schemes. However, the Basic Bootstrap scheme with the AFMS algorithm produces very large estimate deviations.

In summary, we conclude that the bootstrap technique is useful for improving the property of the estimate of the Bhattacharyya bound. We have proposed a direct bias correction scheme to estimate the Bhattacharyya bound of a class pair with a smaller distance. The basic bootstrap scheme with G-based AFMS strategy has been proposed to estimate the Bhattacharyya bound of a class pair with a larger distance.

CHAPTER 4

BOOTSTRAP BASED ERROR ESTIMATES

4.1 Introduction

In the previous chapter, the estimate problem of the Bhattacharyya bound was discussed. This bound is only an upper bound of the error rate of a classifier; it is not even an optimized bound. As the error rate is the crucial measurement of a classifier's performance, it is preferable to know this error rate, if it can be computed. However, in most cases, it is very hard to calculate the error rate; even though, the distributions of the classes are known. Hence, estimating the error is an essential problem in statistical pattern recognition.

Unfortunately, the approaches for estimating the classifier error rate are limited because of the limitation of both training and testing samples. The two main traditional methods are the Apparent Error Estimator and the Cross-validation Estimator (Leave-one-out). The former apparently under-estimates the error rate [Ef86], while the latter is much better than the first. New studies done in the last two decades based on Efron's bootstrap technique suggested that it could be used to improve the Cross-validation Estimator [Ef83]. It was argued that the bootstrap estimator had a much lower variation, although its bias is larger than that of the Cross-validation scheme. Some alternatives to the bootstrap approaches, such as the E0 estimator, the 0.632 estimator etc. were also discussed in Efron's paper. It pointed out that the 0.632 estimator was a clear winner of the sampling experiments.

After this, a comparative study on these and other methods, including the Apparent error, Cross-validation, MC and convex bootstrap estimators, was done by Chernick, Murthy and Nealy, [CMN85] and [CMN86]. Their experiments were carried out with a linear classifier and the results also suggested that the 0.632 estimator was better than the others. Jain, Dubes, and Chen also compared the bootstrap estimator with Cross-validation and Apparent error estimators [JDC87]. Their experiments were carried out with 1-NN and Quadratic classifiers. Their work again supported the conclusions that bootstrap estimators have a smaller standard deviation than the Cross-validation has, and that the 0.632 estimator has a better performance than the other estimator. Davison and Hall discussed the theoretical aspects of the bootstrap and the Cross-validation estimators [DH92]. They showed that both the bootstrap and the Cross-validation have a bias of order n^{-1} if the populations are fixed, and only the bias of the bootstrap will increase to order $n^{-1/2}$ when the populations converge towards one another at the rate of $n^{-1/2}$. To improve the bias of the bootstrap estimator, another modified bootstrap estimator, Leave-one-out bootstrap was introduced by Efron and Tibshibani [ET97], which is a combination of the Cross-validation and bootstrap estimators. Associated with the 0.632 method, the 0.632 + (Leave-one-out) method performed well in the sampling experiments.

In this chapter, the aforementioned error estimating methods will first be discussed except for the MC and Convex bootstrap. These are, in some ways, similar to the Leave-one-out bootstrap and other methods that will be introduced in this chapter.

The detailed descriptions of the algorithms for each error rate estimator, including the Apparent error estimator, Cross-validation or Leave-one-out, basic bootstrap, the E0 error estimator, and two 0.632 error estimators, will be given in the second section. Results of a series of simulations of these estimators will be illustrated and discussed in the third section.

In the latter parts of this chapter, some new algorithms will be introduced to estimate the error rate. These algorithms are mainly based on the idea of ‘local mean’ and ‘pseudo-sample data’, which is related to the idea of the SMIDAT algorithm as discussed in Chapter 2. Therefore, such an algorithm is referred to as the *Pseudo-sample* method, as opposed to the previous algorithms that only use the original sample data to do calculations, which are referred to as the *Real-sample* methods. The pseudo-sample method is the topic of the fourth section.

4.2 Algorithms of the Error Rate Estimating

Using the notation given in Chapter 3, let

$$\mathbf{X} = \{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2, \dots, \underline{\mathbf{X}}_n\} = \{(\underline{\mathbf{V}}_1, \omega_1), (\underline{\mathbf{V}}_2, \omega_2), \dots, (\underline{\mathbf{V}}_n, \omega_n)\} \quad (4.1)$$

be a training sample, $\eta(\underline{\mathbf{V}}, \mathbf{X})$ be a decision rule or classifier based on the training sample $\mathbf{X}$, and $Q[\omega, \eta(\underline{\mathbf{V}}, \mathbf{X})]$ be an error indicator,

$$Q[\omega, \eta(\underline{\mathbf{V}}, \mathbf{X})] = \begin{cases} 0, & \text{if } \eta(\underline{\mathbf{V}}, \mathbf{X}) = \omega \\ 1, & \text{otherwise} \end{cases} \quad (4.2)$$

Thus, the error rate of classifier $\eta(\underline{\mathbf{V}}, \mathbf{X})$ can be defined as

$$\text{Err} = E\{Q[\omega, \eta(\underline{\mathbf{V}}, \mathbf{X})]\}, \quad (4.3)$$

where $(\underline{V}, \omega)$ is a random vector independent of $\mathbf{X}$. In general, the quantity Err is unknown, and our task is to obtain an estimate for it.

As mentioned above, there are various algorithms available to estimate the error rate of a classifier. Below are the detailed descriptions of the algorithms.

4.2.1 Apparent Error

The apparent error rate estimator uses the training samples themselves as the testing samples to estimate the error rate. With the training sample given in (4.1), the apparent error estimator (Err_{app}) is

$$\text{Err}_{\text{app}} = \frac{1}{n} \sum_{i=1}^n Q[\omega, \eta(\underline{V}_i, \mathbf{X}_i)]. \quad (4.4)$$

A generalized algorithm of Err_{app} can be described as follows.

Algorithm 4.1 Apparent Error Estimator

Input: (i) c , the number of classes;
(ii) n_i , the size of the training samples of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;

Output: The error estimate.

Method

BEGIN

BuildClassifier()

Error = 0

Total = 0

For $i = 1$ **to** c **do**

Total = Total + n_i

For $j = 1$ **to** n_i **do**

Error = Error + **ClassifyingError**($\underline{x}[j, i]$)

End-For

End-For

Error = Error / Total

Return Error

END Apparent Error Estimator

The algorithm is very simple. The purpose of the **BuildClassifier** procedure in the above algorithm is to construct the classifier based on the given training samples. The algorithm for this procedure depends on the kind of classifier used and being tasked. If one wants to use the k -NN classifier with the Euclidean distance; the **BuildClassifier** procedure can be omitted. If one wants to use the Bayesian classifier; the task of the **BuildClassifier** procedure is to calculate the means and covariances for each class; and so on. The **ClassifyingError** procedure performs the function of evaluating the error indicator given in (4.2). First, the **ClassifyingError** procedure uses the classifier constructed by the **BuildClassifier** procedure to classify a training pattern $\underline{x}[j, i]$. It returns the value 0 if $\underline{x}[j, i]$ is classified to i , and it returns the value 1 if the sample is misclassified. As the details of the **BuildClassifier** and **ClassifyingError** procedures straightforward (depending on the discriminant function), the algorithms for these two procedures are not included here.

It is obvious that the algorithm of the Apparent error estimator only substitutes the training sample to calculate the error rate. Therefore, the estimated value of the error rate will always be zero with a probability 1 if 1-NN classifier is used in this algorithm and the feature vector $\underline{Y}$ follows a continuous distribution. It is thus clear that the Apparent error estimator under-estimates the error rate.

4.2.2 Cross-validation (Leave-one-out)

Let $\mathbf{X}_{(i)} = \{\underline{\mathbf{X}}_j : j \neq i\} = \{(\underline{\mathbf{V}}_j, \omega_j) : j \neq i\}$ denote the sub-set of the training samples which does not contain the pattern $(\underline{\mathbf{V}}_i, \omega_i)$. The Cross-validation, or Leave-one-out error estimator will then be;

$$\text{Err}_{\text{cv}} = \frac{1}{n} \sum_{i=1}^n Q[\omega_i, \eta(\underline{\mathbf{V}}_i, \mathbf{X}_{(i)})]. \quad (4.5)$$

A generalized algorithm of Err_{cv} is given below.

Algorithm 4.2 Leave-One-Out Error Estimator

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{\mathbf{x}}[1, 1], \dots, \underline{\mathbf{x}}[n_1, 1], \underline{\mathbf{x}}[1, 2], \dots, \underline{\mathbf{x}}[n_c, c]$, the training sample data;
Output: the error estimate.

Method

BEGIN

Error = 0

Total = 0

For $i = 1$ **to** c **do**

Total = Total + n_i

For $j = 1$ **to** n_i **do**

BuildClassifier(j, i)

Error = Error + **ClassifyingError**($\underline{\mathbf{x}}[j, i]$)

End-For

End-For

Error = Error / Total

Return Error

END Leave-One-Out Error Estimator

The algorithm is still simple and slightly different from the Apparent error algorithm. Here the **BuildClassifier** procedure is called n times, while in **Algorithm 4.1** it is called only once. Every time the **BuildClassifier** procedure is invoked, there are two parameters (j, i) passed to it; while there is no parameter passed to it in **Algorithm 4.1**.

The parameters are used to indicate that the pattern $\underline{x}[j, i]$ is removed from the training sample, and the remaining training patterns are used to construct the classifier, which means that $\mathbf{X}_{(i)}$, not $\mathbf{X}$, is used to build the classifier. Thus, the classifier will not have $\underline{x}[j, i]$ participating in when it is used to classify $\underline{x}[j, i]$. By this strategy, the problem caused by the Apparent error estimator is avoided.

4.2.3 Basic Bootstrap

As the Apparent error $\mathbf{Err}_{app}$ under-estimates the true error $\mathbf{Err}$, a positive bias (PB) of the $\mathbf{Err}_{app}$ can be defined as

$$\mathbf{PB}(\mathbf{X}, \mathbf{F}) = \mathbf{Err} - \mathbf{Err}_{app}. \quad (4.5)$$

The expectation of this bias can be written as

$$\mathbf{Bias}(\mathbf{F}) = \mathbf{E}[\mathbf{PB}(\mathbf{X}, \mathbf{F})] = \mathbf{E}[\mathbf{Err} - \mathbf{Err}_{app}]. \quad (4.6)$$

If $\mathbf{Bias}(\mathbf{F})$ were known, an optimised estimate of $\mathbf{Err}$ could be given as

$$\mathbf{Err}_{op} = \mathbf{Err}_{app} + \mathbf{Bias}(\mathbf{F}). \quad (4.7)$$

In most cases the $\mathbf{Bias}$ is unknown and should be estimated from the training samples.

The Cross-validation algorithm mentioned above estimates the $\mathbf{Bias}$ with:

$$\mathbf{Bias}^{(cv)} = \frac{1}{n} \sum_{i=1}^n \mathbf{Q}[\omega_i, \eta(\underline{\mathbf{V}}_i, \mathbf{X}_{(i)})] - \mathbf{Err}_{app}.$$

With the bootstrap technique, the $\mathbf{Bias}(\mathbf{F})$ may be estimated in several ways.

Suppose the observed patterns of the training sample $\mathbf{X}$ are $\underline{\mathbf{X}}_1 = \underline{\mathbf{x}}_1, \underline{\mathbf{X}}_2 = \underline{\mathbf{x}}_2, \dots, \underline{\mathbf{X}}_n = \underline{\mathbf{x}}_n$.

Similar to what was done in the previous chapters, a bootstrap sample $\mathbf{X}^* = \{\underline{\mathbf{X}}_1^*, \underline{\mathbf{X}}_2^*, \dots, \underline{\mathbf{X}}_n^*\}$,

..., $\underline{X}_n^*$ } may be drawn from the empirical distribution $\hat{F}$. Thus a bootstrap estimate of

Bias will be

$$\mathbf{Bias}^* = \sum_{i=1}^n \left(\frac{1}{n} - \mathbf{w}_i^* \right) Q[\omega_i, \eta(\underline{V}_i, \mathbf{X}^*)], \quad (4.8)$$

where $\mathbf{w}_i^*$ is the proportion of the bootstrap sample on $\underline{x}_i$,

$$\mathbf{w}_i^* = \frac{\# \{ \underline{X}_j^* = \underline{x}_i \}}{n} \quad i = 1, 2, \dots, n. \quad (4.9)$$

By repeatedly drawing the bootstrap samples B times, the bootstrap estimate of **Bias** is

$$\mathbf{Bias}^{(\text{boot})} = \frac{1}{B} \sum_{b=1}^B \hat{\omega}_b^*. \quad (4.10)$$

Thus, the bootstrap estimate of the **Err** is;

$$\mathbf{Err}_{\text{boot}} = \mathbf{Err}_{\text{app}} + \mathbf{Bias}^{(\text{boot})}. \quad (4.11)$$

The bootstrap estimator discussed above is called the basic bootstrap estimator.

An algorithm for it is given below:

Algorithm 4.3 Basic Bootstrap Error Eestimator

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.

Output: the error estimate.

Method

BEGIN

 Total = 0

For $i = 1$ **to** c **do**

 Total = Total + n_i

End-For

 Error-estimate = 0

For $k = 1$ **to** B **do**

 GetBootstrapSample()

 BuildClassifier($\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$)

 Error = 0

For $i = 1$ **to** c **do**

For $j = 1$ **to** n_i **do**

 Error = Error + $(\frac{1}{n} - w[j, i]) \times \text{ClassifyingError}(\underline{x}[j, i])$

End-For

End-For

 Error-estimate = Error-estimate + Error / Total

End-For

 Error-estimate = Error-estimate / B

 Error-estimate = Error-estimate + AppError()

 Return Error-estimate

END Basic Bootstrap Error Estimator

Procedure GetBootstrapSample

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
Output: (i) Bootstrap sample $\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$
(ii) weights $w[1, 1], w[2, 1], \dots, w[n_c, c]$.

Method**BEGIN**

Total = 0

For $i = 1$ **to** c **do** Total = Total + n_i **For** $j = 1$ **to** n_i **do** $w[j, i] = 0$ **End-For****End-For****For** $i = 1$ **to** c **do** **For** $j = 1$ **to** n_i **do** $m = \text{random integer in } [1, n_i]$ $\underline{y}[j, i] = \underline{x}[m, i]$ $w[j, i] = w[j, i] + 1$ **End-For****End-For****For** $i = 1$ **to** c **do** **For** $j = 1$ **to** n_i **do** $w[j, i] = w[j, i] / \text{Total}$ **End-For****End-For****Return** $\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c], w[1, 1], w[2, 1], \dots, w[n_c, c]$ **END Procedure GetBootstrapSample**

The parameters passed to the **GetBootstrapSample** procedure are given in the descriptive part of the procedure, although they are omitted when the procedure is called in the main body of the algorithm. The **GetBootstrapSample** procedure in this algorithm performs two functions: it draws a bootstrap sample and sets the weights for each pattern of the training sample. The bootstrap sample is drawn separately from each class. The reason for doing this is to avoid the bias of the bootstrap sample data. If all the training samples from different classes were put together to draw a bootstrap sample, the

bootstrap sample might contain no data from one or more classes. This would cause an even more biased estimate, or might even deduce a bootstrap sample, which could not be used to construct a classifier.

The weights calculated in the procedure are for the bootstrap estimate calculation. One may argue that the **GetWeights** procedures given in Chapter 3 can be used to substitute the **GetBootstrapSample** procedure, so that one could use different bootstrap schemes, basic bootstrap, Bayesian bootstrap and random weighting method. This is true in some cases. The key issue is to determine the kind of classifier used. If the Bayesian classifier is used, a **GetWeights** procedure could take the place of the **GetBootstrapSample** procedure in the algorithm. Then the weights $w[1, 1]$, $w[2, 1]$, ..., $w[n_c, c]$ would be passed to the **BuildClassifier** procedure to construct the classifier. However, a real bootstrap sample will be required, if a k -NN classifier is used. It is not difficult to see that the Bayesian bootstrap and the random weighting method can not be applied to a k -NN classifier.

The **BuildClassifier** procedure of this algorithm is the same as that for **Algorithm 4.1**, except that it uses the bootstrap sample to build the classifier. The **AppError** procedure only uses the **Algorithm 4.1** to calculate an apparent error estimate. Thus, the parameters (the number of classes, the training sample sizes of the classes, and the training sample data) have to be passed to the procedure, even though it has not been indicated in the algorithm.

4.2.4 E0 estimator

The **E0** estimator is another estimate based on the bootstrap technique. The difference between the basic bootstrap and the **E0** estimator is that the **E0** estimator tests only the training patterns not contained in a bootstrap sample, while the basic bootstrap tests all the patterns of the training sample. To carry out an **E0** estimate, after each bootstrap sample drawing, the error rate calculation has to be changed. Suppose B bootstrap samples are drawn, the b^{th} bootstrap sample is denoted as $\mathbf{X}^{*b}$, and the set of training patterns not in $\mathbf{X}^{*b}$ is denoted as $\mathbf{A}_b$. Then the **E0** error rate estimate is:

$$\text{Err}_{\text{E0}} = \frac{\sum_{b=1}^B \sum_{\mathbf{A}_b} Q(\omega, \eta(\mathbf{V}, \mathbf{X}^{*b}))}{\sum_{b=1}^B |\mathbf{A}_b|}, \quad (4.11)$$

where $|\mathbf{A}_b|$ denotes the cardinality of the set $\mathbf{A}_b$. The algorithm of the **E0** estimator is represented below.

Algorithm 4.4 E0 error estimator

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.

Output: The error estimate.

Method
BEGIN
 Total = 0
 Error = 0
 For $k = 1$ **to** B **do**
 GetBootstrapSample()
 BuildClassifier($\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$)
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 If $w[j, i] = 0$ **Then**
 Error = Error + **ClassifyingError**($\underline{x}[j, i]$)
 Total = Total + 1
 End-If
 End-For
 End-For
 End-For
 Error = Error / Total
 Return Error
END E0 error estimator

The procedures used in the above are the same as those used in **Algorithm 4.3**, and so they are not explained in any greater detail. The key point in this algorithm is to check the weight of a training pattern before it is classified. Thus, all the training patterns passed to the **ClassifyingError** procedure are excluded from the bootstrap sample. What should also be pointed out is that this algorithm yields an error rate estimate directly without adding the **AppError** to it. In other words, the **E0** estimator scheme is not an algorithm based on the bias correction.

4.2.5 Leave-one-out Bootstrap

Compared to the Cross-validation, the bootstrap provides an error estimate with a bigger downward bias (see [JDC87], [Ha86] and [ET97]). This is because the basic bootstrap does not explicitly separate the test pattern from the training patterns which are used to construct the classifier, like the Cross-validation does. So Efron and Tibshirani developed a scheme called the *Leave-one-out bootstrap* to deal with this separation.

Recall that in the Cross-validation algorithm, the training pattern set $\mathbf{X}_{(i)}$ was used to build the classifier when the test was done for training pattern $\mathbf{X}_i$. Thus, the testing pattern was not a part of the pattern set used to construct the classifier. So we define an empirical distribution $\hat{\mathbf{F}}_{(i)}$ on the training sample set $\mathbf{X}_{(i)}$,

$$\hat{\mathbf{F}}_{(i)} : \text{mass } \frac{1}{n-1} \text{ at } \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots, \mathbf{x}_n. \quad (4.13)$$

Using this, a bootstrap sample $\mathbf{X}_{(i)}^*$ of size n is drawn from $\hat{\mathbf{F}}_{(i)}$. Observe that $\mathbf{X}_{(i)}^*$ will not contain $\mathbf{X}_i$. For each i , there could be an error estimator given by:

$$\text{Err}_i^* = E_i^* \{ Q[\omega_i, \eta(\mathbf{Y}_n, \mathbf{X}_{(i)}^*)] \}, \quad (4.14)$$

where E_i^* denotes the expectation under the distribution $\hat{\mathbf{F}}_{(i)}$. The error rate estimate of the Leave-one-out bootstrap is:

$$\text{Err}_{\text{cv-boot}} = \frac{1}{n} \sum_{i=1}^n \text{Err}_i^*. \quad (4.15)$$

In the real algorithm, the Err_i^* in (4.14) can be obtained by the following: first, the bootstrap samples are drawn as usual (say B bootstrap samples, $\mathbf{X}^{*1}, \mathbf{X}^{*2}, \dots, \mathbf{X}^{*B}$).

Let N_i^b be the times of training pattern $\underline{\mathbf{X}}_i$ appearing in the b^{th} bootstrap sample. We define:

$$\mathbf{I}_i^b = \begin{cases} 1, & \text{if } N_i^b = 0 \\ 0, & \text{if } N_i^b > 0 \end{cases}, \quad (4.16)$$

and

$$\mathbf{Q}_i^b = \mathbf{Q}[\omega_i, \eta(\underline{\mathbf{V}}_i, \mathbf{X}^{*b})]. \quad (4.17)$$

Thus $\mathbf{Err}_i^*$ can be given by:

$$\mathbf{Err}_i^* = \frac{\sum_{b=1}^B \mathbf{I}_i^b \times \mathbf{Q}_i^b}{\sum_{b=1}^B \mathbf{I}_i^b}. \quad (4.18)$$

The algorithm of the Leave-one-out bootstrap is listed below.

Algorithm 4.5 Leave-one-out Bootstrap Error Estimator

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.

Output: The error estimate.

Method

BEGIN

 Total = 0

For $i = 1$ to c **do**

 Total = n_i

For $j = 1$ to n_i **do**

 Total[j, i] = 0

 Error[j, i] = 0

End-For

End-For

For $k = 1$ to B **do**

 GetBootstrapSample()

 BuildClassifier($\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$)

For $i = 1$ to c **do**

For $j = 1$ to n_i **do**

If $w[j, i] = 0$ **Then**

 Error[j, i] = Error[j, i] + ClassifyingError($\underline{x}[j, i]$)

 Total[j, i] = Total[j, i] + 1

End-If

End-For

End-For

End-For

 Error = 0

For $i = 1$ to c **do**

For $j = 1$ to n_i **do**

 Error = Error + Error[j, i] / Total[j, i]

End-For

End-For

 Error = Error / Total

Return Error

END Leave-one-out Bootstrap Error Estimator

The procedures used in this algorithm are the same as in **Algorithm 4.4**. It should be pointed out that this algorithm uses the same pattern set as that in **Algorithm 4.4** to calculate the error rate. The difference between the two algorithms is the calculations

themselves. In **Algorithm 4.4**, the error rate estimate is counted as the ratio of the total number of the misclassified training patterns to the total number of the training patterns tested. In **Algorithm 4.5**, however, the misclassification ratio for each training pattern within all the bootstrap samples is calculated first, and so that the error rate estimate turns out to be the average of the ratios of all the training patterns.

4.2.6 0.632 Estimators

The motivation of the 0.632 estimators is not strong. However, the simulation results showed that the 0.632 estimators have good performance (see [Ef83], [CMN85], [CMN86], and [ET97]). When a bootstrap sample of size m is drawn from a training sample of size n , the probability of a training pattern being excluded from the bootstrap sample is $(1 - \frac{1}{n})^m$ if the training patterns from all the classes are put together to draw a bootstrap sample. In general, since the value of m is set to n , as n goes to infinity, $(1 - \frac{1}{n})^n$ tends to $e^{-1} \approx 0.368$. So the value 0.632 is considered to be the contribution rate of the **E0** estimator to the bias correction [Ef83]. Thus, the **Bias** defined in (4.6) can be estimated by:

$$\mathbf{Bias}^{(0.632)} = 0.632 \times (\mathbf{E0} - \mathbf{AppErr}). \quad (4.19)$$

This gives the **Err** estimate

$$\mathbf{Err}_{(0.632)} = \mathbf{AppErr} + \mathbf{Bias}^{(0.632)} = 0.632 \times \mathbf{E0} + 0.368 \times \mathbf{AppErr}. \quad (4.28)$$

As the Leave-one-out bootstrap estimator has the same property as the **E0** estimator, (i.e. only the training patterns excluded from the bootstrap sample are counted to estimate the

error rate), an alternative way is to use the Leave-one-out bootstrap estimator $\text{Err}_{\text{cv-boot}}$ instead of E0 , which yields the Err estimate

$$\text{Err}_{(0.632+)} = 0.632 \times \text{Err}_{\text{cv-boot}} + 0.368 \times \text{AppErr}. \quad (4.29)$$

It is obvious that the algorithm $\text{Err}_{(0.632)}$ is a straightforward application of the algorithms E0 and AppErr , and formula (4.28). Similarly, the algorithm to yield $\text{Err}_{(0.632+)}$ is an application of the algorithms $\text{Err}_{\text{cv-boot}}$ and AppErr , and formula (4.29).

4.3. Simulation Results

Experiments were carried out for all the algorithms discussed in the last section. All of the experiments used the Data Set I with training sample sizes 8 and 24 for each class. As in Chapter 3, only two classes were involved in an experiment: one of them was the class A, and the other was chosen from the rest.

As the 1-NN classifier is meaningless for the Apparent error estimator, the 3-NN classifier was used in all the experiments. The true errors were obtained by using independent testing samples of size 1,000 for each class. Therefore, the true errors are the result of measuring 2,000 independent testing patterns. Each experiment was repeated 200 times. TABLE 4.1 and TABLE 4.2 give the results of the average out of 200 trails with training sample sizes 8 and 24 for each class respectively.

TABLE 4.1 Error Rate Estimates of 3-NN classifiers (sample size 8)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.99150	37.08750	33.37225	23.54025	14.33675	6.3225
	STD	4.05386	5.921621	5.755794	4.33367	3.139198	1.86351
Apparent Error	Mean	25.18750	20.00000	18.40625	13.81250	8.562500	3.56250
	STD	10.41915	9.519716	8.081555	7.944878	6.534464	4.780739
Leave-One-Out	Mean	52.46875	40.375	36.6875	25.53125	16.15625	7.000000
	STD	17.45415	16.49858	16.121	14.45329	10.70334	7.276890
Basic Bootstrap	Mean	28.98517	23.05344	21.22073	15.86291	9.877842	4.135117
	STD	11.12407	10.32956	8.820669	8.777943	7.191825	5.235515
E0	Mean	52.42875	42.58402	35.8724	25.25705	16.37028	7.157510
	STD	11.39010	12.51909	13.04464	12.00116	9.138354	5.826289
0.632	Mean	42.40397	34.2731	29.44485	21.04545	13.49702	5.834547
	STD	9.692471	10.48393	10.25504	9.919126	7.673558	5.008007
Leave-One-Out Bootstrap	Mean	49.3988	39.3107	36.14832	25.35199	16.46006	7.259035
	STD	11.37005	12.22582	12.85411	11.95885	9.016972	5.764418
0.632+	Mean	40.48904	32.20436	29.61924	21.10546	13.55376	5.89871
	STD	9.708022	10.32714	10.15273	9.912306	7.593661	4.987327

TABLE 4.2 Error Rate Estimates of 3-NN classifiers (sample size 24)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.49800	35.92875	32.34675	22.7975	13.83275	5.83675
	STD	2.340672	3.050293	2.773933	2.203314	1.729629	1.01312
Apparent Error	Mean	23.35417	19.54167	17.86458	12.38542	8.34375	3.604167
	STD	5.703884	5.18980	4.811722	4.593726	3.671029	2.689922
Leave-One-Out	Mean	48.68750	37.13542	33.58333	22.60417	13.52083	6.06250
	STD	9.122312	9.745987	8.719092	7.690718	5.705405	3.876288
Basic Bootstrap	Mean	24.74947	20.65822	18.88875	13.10459	8.808794	3.807138
	STD	5.76893	5.316619	4.917935	4.712453	3.768816	2.76579
E0	Mean	47.42198	36.83978	33.33931	22.82567	14.06874	6.106057
	STD	6.359907	8.131911	7.231284	6.814515	4.895951	3.440337
0.632	Mean	38.56503	30.47407	27.64461	18.98366	11.96194	5.185361
	STD	5.497819	6.538436	5.815562	5.577282	4.186173	3.007212
Leave-One-Out Bootstrap	Mean	47.6213	37.58765	34.23481	23.84802	15.17183	6.723717
	STD	4.549707	7.151611	6.241987	6.217412	4.565808	3.23101
0.632+	Mean	38.69099	30.94673	28.21056	19.62978	12.6591	5.575723
	STD	4.206508	5.834633	5.111546	5.13524	3.910966	2.831333

The results of the experiments show that the bootstrap technique works well in this case. Compared to the Leave-one-out estimator, all the bootstrap related algorithms

have relatively smaller standard deviations; although, some of them have larger biases than the Leave-one-out. The basic bootstrap algorithm improved the Apparent error rate estimator, but it still tends to under-estimate the error rates, while the E0 and Leave-one-out bootstrap algorithms generally over-estimate the error rates. The 0.632+ algorithm did enhance the performance of the 0.632 algorithm, which confirms the importance of the Leave-one-out approach. Although the Leave-one-out algorithm appears attractive, always yielding the best performance with all the experiment class pairs, it will be seen later in this chapter that, from some standpoints, the bootstrap related algorithms perform better than the Leave-one-out. It can also be seen that the results of these experiments are basically consistent with the research works done in [Ef83], [CMN85], [Ef86], [CMN86], [JDC87], and [ET97].

4.4. Pseudo-Sample Algorithms

4.4.1 Pseudo-Testing Algorithm

As pointed out at the beginning of the chapter, the algorithms previously mentioned use only the training patterns to estimate the error rate of the classifier. This means that the training patterns are used for both the construction of the classifier and for testing the classifier. Two lessons have been learned from the above results:

1. The training pattern used to test the classifier should be separated from those used to build the classifier. This is the key issue that makes the Leave-one-out error estimator perform well. This is also why the 0.632+ estimator performed better than the 0.632 estimator;

2. A larger training sample size would improve the error estimators. This can be seen by comparing the results in TABLE 4.1 and TABLE 4.2.

Separating the training patterns from the testing and building of the classifier can be done by merely applying the Cross-validation technique. However, in practice, training sample size is limited. The separating of the two implies the reduction of the number of the patterns used to build the classifier. Although there will generally not be a significant effect if only one training pattern is left out, this is not the case when the sample size is itself small. The examples of the 0.632 estimators have shown that a weak relationship between the patterns of testing and those of building the classifier will not make things worse. Therefore, it would be beneficial to devise a way to increase the training sample size and simultaneously retain the patterns used for testing the classifier in question.

This is what the SIMDAT algorithm given in Section 2.3.5 does. The algorithms provided here use a SIMDAT-like approach to generate pseudo-training patterns. Then the original and the pseudo training patterns are used separately as the testing patterns and the patterns to construct the classifier. To distinguish from the algorithms mentioned in the last section, the algorithms discussed here are referred to as *pseudo-sample* algorithms.

The main idea in the SIMDAT algorithm is that of combining the weighted m -nearest neighbours of a sample point to construct a pseudo-sample data. Here, the same approach is used to construct pseudo-training patterns. There is no restriction on the method to be used for setting the weights of the m -nearest neighbours of a training pattern. A generalized algorithm to construct a pseudo-training pattern is given below:

Algorithm 4.6 Generating A Pseudo-Training Pattern**Procedure GetPseudoPattern I****Input:** $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[m]$ the m -nearest neighbours of the training pattern $\underline{x}$ **Output:** A pseudo-training pattern.**Method****BEGIN****GetWeights(m)**

$$\underline{y} = \sum_{i=1}^m \underline{w}[i] \times \underline{x}[i]$$

Return $\underline{y}$ **END Procedure GetPseudoPattern I**

Observe that the m -nearest neighbours of the training pattern $\underline{x}$ include $\underline{x}$ itself.

The **GetWeights** procedure above can be any one of the **GetWeights** procedures presented in Chapter 3. Of course, the **GetWeights** procedure for the Bayesian bootstrap or the random weighting method are preferred because then the weights $w[1], w[2], \dots, w[m]$ are continuously distributed, and the pseudo-training pattern generated would not identify with any pattern in the training sample set. The main idea is that the pseudo-training pattern generated should oscillate around the mean of the m -nearest neighbours. So another way to generate a pseudo-training pattern is to apply the approach represented in the SIMDAT algorithm. The following algorithm uses this approach:

Algorithm 4.7 Generating A Pseudo-Training Pattern**Procedure GetPseudoPattern II****Input:** $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[m]$ the m -nearest neighbours of the training pattern $\underline{x}$ **Output:** A pseudo-training pattern.**Method****BEGIN****Mean** = 0**For** $i = 1$ **to** m **do****w**[i] = **RandomVariableGenerator**()**Mean** = **Mean** + $\underline{x}[i]$ **End-For****Mean** = **Mean** / m

$$\underline{y} = \sum_{i=1}^m \underline{w}[i] \times (\underline{x}[i] - \text{Mean})$$

y = **y** + **Mean****Return** **y****END Procedure GetPseudoPattern II**

The differences between **Algorithm 4.7** and **Algorithm 4.6** are that the weights in **Algorithm 4.7** are not standardized, and the weights are assigned to the centralized feature vectors $(\underline{x}[i] - \text{Mean})$. After the linear combination of the centralized feature vectors has been obtained, the mean of the m -nearest neighbours is added back to obtain the pseudo-training pattern. Thus, the **RandomVariableGenerator** procedure above should generate a random variable with zero as its conditional expectation. It is easy to see that **Algorithm 4.7** uses the approach used in the SIMDAT algorithm to obtain pseudo-patterns.

After the pseudo-training patterns have been generated, they can be used in various ways, either as the testing patterns, or as the patterns to construct the classifier itself. The scheme is called a *Pseudo-Testing* algorithm, if the pseudo-training samples are used for testing purposes and a *Pseudo-Classifer* algorithm, if the pseudo-training

samples are used to construct the classifier. A Pseudo-Testing algorithm is described below:

Algorithm 4.8 Pseudo-Testing

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.
(v) m , the number of the nearest neighbours of a training pattern

Output: The error estimate.

Method

BEGIN

 Total = 0

 Error = 0

For $i = 1$ to c **do**

For $j = 1$ to n_i **do**

 GetNearestNeighboursOf($\underline{x}[j, i], m$)

End-For

End-For

For $k = 1$ to B **do**

 GeneratePseudoSample()

 BuildClassifier($\underline{x}[1, 1], \underline{x}[2, 1], \dots, \underline{x}[n_c, c]$)

For $i = 1$ to c **do**

For $j = 1$ to n_i **do**

 Error = Error + ClassifyingError($\underline{y}[j, i]$)

 Total = Total + 1

End-For

End-For

End-For

 Error = Error / Total

Return Error

END Pseudo-Testing

Procedure GeneratePseudoSample

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;

Output: The pseudo-training patterns $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$.

Method
BEGIN
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 $k = \text{random integer in } [1, n_i]$
 $\underline{y}[j, i] = \text{GetPseudoPattern}(m\text{-nearest neighbours of } \underline{x}[k, i])$
 End-For
 End-For
END GeneratePseudoSample

In this algorithm, $\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$ represent the pseudo-training samples generated. The pseudo-training samples have not only in total, but also for each class, the same size as the original training sample. After the classifier has been built with the original training sample, the pseudo-training sample is used to estimate the error rate of the classifier. The **GetNearestNeighboursOf**($\underline{x}[j, i], m$) procedure is used to calculate the m -nearest neighbours of the training pattern $\underline{x}[j, i]$. This function is basically used to calculate the Euclidean distances between $\underline{x}[j, i]$ and other patterns in the same class, and select m from them with the minimal distances. The **GetPseudoPattern** procedure, invoked in the **GeneratePseudoSample** procedure, can be either of those given in **Algorithm 4.6** and **4.7**. Unlike the SIMDAT algorithm, the **GetPseudoPattern** procedure does not generate a pseudo-training pattern for each pattern in the training sample. Instead, it uses the bootstrap technique to randomly select a training pattern first; then it uses the m -nearest neighbours of the pattern to generate a pseudo-training pattern.

Experiments on **Algorithm 4.8** were done with the Data Set I. Only two classes were involved in an experiment, each of them has a training sample of size eight. All of the experiments were carried out with a 3-NN classifier. Two different **GetPseudoPattern** procedures were used in the experiments: one of them used **Algorithm 4.6** with the **GetWeights** procedure for the Bayesian bootstrap, and the other used **Algorithm 4.7** with a **RandomVariableGenerator** procedure to generate the $U(\frac{1}{m} - \sqrt{\frac{3(m-1)}{m^2}}, \frac{1}{m} + \sqrt{\frac{3(m-1)}{m^2}})$ distributed random variables. The other parameters used in the experiments were: $B = 200$, $m = 3$. TABLE 4.3 lists the statistics from the 200 trials for each experiment.

TABLE 4.3 Error Rate Estimates of the Pseudo-Testing algorithm

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.99150	37.08750	33.37225	23.54025	14.33675	6.3225
	STD	4.05386	5.921621	5.755794	4.33367	3.139198	1.86351
SIMDAT	Mean	23.07175	15.26616	14.05301	8.634487	5.43765	2.126613
	STD	15.65927	13.33733	13.97576	11.33642	8.879793	4.58013
Bayesian Bootstrap	Mean	17.87747	12.83954	10.92097	7.108575	3.346563	0.635275
	STD	8.370604	7.583107	7.198176	6.424745	4.364733	1.493685

The results indicate that the estimates provided by the pseudo-testing algorithms are seriously biased downward. The cause of this is that the testing patterns are the linear combinations of the m -nearest neighbours. As these linear combinations tend to the local mean of the patterns in a class, their characteristics are more significant, and easier to distinguish from each other. So the performance of the Pseudo-Testing algorithm is not good.

4.4.2 The Pseudo-Classifier Algorithm

The Pseudo-Classifier algorithm exchanges the position of the original training patterns with that of the pseudo-training patterns. The pseudo-training sample is used to build the classifier, and the original training sample is used to test the classifier error rate.

The pseudo-classifier algorithm is represented here:

Algorithm 4.9 Pseudo-Classifier

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.
(v) m , the number of the nearest neighbours of a training pattern

Output: The error estimate.

Method
BEGIN
 Total = 0
 Error = 0
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 GetNearestNeighboursOf($\underline{x}[j, i], m$)
 End-For
 End-For
 For $k = 1$ **to** B **do**
 GeneratePseudoSample()
 BuildClassifier($\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$)
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 Error = Error + ClassifyingError($\underline{x}[j, i]$)
 Total = Total + 1
 End-For
 End-For
 End-For
 Error = Error / Total
 Return Error
END Pseudo-Classifier

The algorithm above is almost the same as Algorithm 4.8, except that the parameters passed to the BuildClassifier and ClassifyingError procedures have

changed. Although the change is small, the performance is highly improved. TABLE 4.4 – 4.5 list the statistics of the experiments on **Algorithm 4.9**. The experiments were done with the Data Set I.

All the experiment parameters are the same as those for the experiments on **Algorithm 4.8**, except that two training sample sizes 8 and 24 are used here. Additional experiments were carried out with the **GetWeights** procedure for the random weighting method.

TABLE 4.4 Error Rate Estimates of the Pseudo-Classifier algorithm (sample size 8)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.99150	37.08750	33.37225	23.54025	14.33675	6.3225
	STD	4.05386	5.921621	5.755794	4.33367	3.139198	1.86351
SIMDAT	Mean	35.97422	31.03078	29.49531	23.72187	19.60203	15.42453
	STD	6.815439	7.748732	8.117096	8.657076	7.226305	6.58984
Bayesian Bootstrap	Mean	27.98375	22.2825	20.64437	14.96141	9.709844	4.322344
	STD	6.924403	7.139102	6.855422	6.843363	5.653335	3.882771
Random Weighting	Mean	28.00937	22.23141	20.63219	14.89687	9.65375	4.266875
	STD	7.105689	7.231788	6.928664	6.933816	5.699131	3.830216

TABLE 4.5 Error Rate Estimates of the Pseudo-Classifier algorithm (sample size 24)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.49800	35.92875	32.34675	22.7975	13.83275	5.83675
	STD	2.340672	3.050293	2.773933	2.203314	1.729629	1.01312
SIMDAT	Mean	39.6638	33.11203	31.21865	24.37156	19.04755	13.31255
	STD	3.852234	6.084795	5.925159	5.912813	5.373447	5.069416
Bayesian Bootstrap	Mean	29.64885	23.6813	21.54167	15.04438	9.498958	3.994166
	STD	4.261607	4.985067	4.505514	4.428612	3.398046	2.433873
Random Weighting	Mean	29.78271	23.77761	21.66401	15.13792	9.474687	3.949583
	STD	4.366496	5.099502	4.572865	4.525221	3.423749	2.472508

For convenience, a pseudo-classifier algorithm is referred to as a *SIMDAT pseudo-classifier* algorithm (SPC), if the SIMDAT-like algorithm was used to generate the pseudo-training sample. A pseudo-classifier algorithm is referred to as a *Bayesian Bootstrap pseudo-classifier* algorithm (BBPC), if the Bayesian bootstrap is used to

generate the pseudo-training sample. A pseudo-classifier algorithm is referred to as a *Random Weighting pseudo-classifier* algorithm (RWPC), if the random weighting method is used to generate the pseudo-training sample.

Although the SPC algorithm did not perform well enough in the experiments, the BBPC and RWPC performed better than the basic bootstrap. They have smaller biases and standard deviations. The experiments show also that, for the purpose of error rate estimating, the pseudo-training pattern generating algorithm, **Algorithm 4.6** is better than **Algorithm 4.7**.

4.4.3 Leave-one-out Pseudo-Classfier Algorithm

A further improvement on the pseudo-classifier algorithm is to introduce the Leave-one-out approach. As we have already seen that the bootstrap algorithm is sharpened by the Leave-one-out, it is natural to think that the pseudo-classifier algorithm might benefit from it as well. The Leave-one-out pseudo-classifier algorithm (LPC) is given below.

Algorithm 4.10 Leave-One-Out Pseudo-Classifier

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.
(v) m , the number of the nearest neighbours of a training pattern

Output: The error estimate.

Method**BEGIN**

Total = 0

Error = 0

For $i = 1$ to c **do** **For** $j = 1$ to n_i **do** GetNearestNeighboursOf($\underline{x}[j, i], m$) **End-For****End-For****For** $k = 1$ to B **do** **For** $i = 1$ to c **do** **For** $j = 1$ to n_i **do** GeneratePseudoSample($\underline{x}[j, i]$) BuildClassifier($\underline{y}[1, 1], \underline{y}[2, 1], \dots, \underline{y}[n_c, c]$) Error = Error + ClassifyingError($\underline{x}[j, i]$)

Total = Total + 1

End-For **End-For****End-For**

Error = Error / Total

Return Error**END Pseudo-Testing**

Procedure GeneratePseudoSample

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(v) $\underline{x}[q, p]$, the training pattern should be left out

Output: The pseudo-training patterns $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$.

Method**BEGIN**

For $i = 1$ to c **do**

For $j = 1$ to n_i **do**

do

$k =$ random integer in $[1, n_i]$

While $(\underline{x}[k, i] = \underline{x}[q, p])$

$\underline{y}[j, i] = \text{GetPseudoPattern}(m\text{-nearest neighbours of } \underline{x}[k, i])$

End-For

End-For

END GeneratePseudoSample

The reader will observe that there are some changes in **Algorithm 4.10** compared to **Algorithm 4.9**. First, the **GeneratePseudoSample** and **BuildClassifier** procedures have moved to the innermost **For** loop, so different pseudo-training samples are used to test different training patterns. Secondly, when the **GeneratePseudoSample** procedure is invoked, a training pattern $\underline{x}[q, p]$ is passed to it. When executing the **GeneratePseudoSample** procedure, there is a **While** loop to ensure that the patterns drawn by the bootstrap sampling will exclude the pattern $\underline{x}[q, p]$. This means that no pseudo-training pattern would be a linear combination of the m -nearest neighbours of $\underline{x}[q, p]$. Although it is still possible for $\underline{x}[q, p]$ to be involved in the pseudo-training sample as one of the m -nearest neighbours of another pattern, the relationship between $\underline{x}[q, p]$ and the pseudo-training sample is relatively weak.

TABLE 4.6 – 4.7 list the statistics of the experiments using **Algorithm 4.10**. The experiments were done with the Data Set I. All the parameters used are the same as those used for the experiments involving **Algorithm 4.9**.

TABLE 4.6 Error Rate Estimates of the Leave-One-Out Pseudo-Classifier Algorithm (sample size 8)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.99150	37.08750	33.37225	23.54025	14.33675	6.3225
	STD	4.05386	5.921621	5.755794	4.33367	3.139198	1.86351
SIMDAT	Mean	43.205	36.33188	34.32563	27.09	22.01375	16.91
	STD	8.497344	8.877637	9.745623	9.643984	7.871366	6.760757
Bayesian Bootstrap	Mean	37.97937	29.78687	27.2975	19.78188	12.82187	5.785625
	STD	9.943817	9.613514	9.734568	9.240409	7.257156	4.866056
Random Weighting	Mean	38.72188	30.02	27.51187	19.78188	12.77125	5.8825
	STD	10.02002	9.850337	9.737521	9.240409	7.291842	4.873968

TABLE 4.7 Error Rate Estimates of the Leave-One-Out Pseudo-Classifier Algorithm (sample size 24)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
True	Mean	46.49800	35.92875	32.34675	22.7975	13.83275	5.83675
	STD	2.340672	3.050293	2.773933	2.203314	1.729629	1.01312
SIMDAT	Mean	44.1575	36.39396	34.11125	26.30104	20.3325	14.21229
	STD	4.463164	6.851486	6.660264	6.302611	5.643442	5.190987
Bayesian Bootstrap	Mean	38.38708	30.42146	27.82375	19.53833	12.32875	5.453958
	STD	5.290009	6.350641	5.711829	5.485231	4.136333	3.003775
Random Weighting	Mean	38.06104	29.48875	26.63542	18.10437	11.28021	4.773958
	STD	5.783742	6.84088	5.992693	5.634306	4.102531	2.778053

As in the previous section, a pseudo-classifier algorithm is referred as to a *Leave-One-Out SIMDAT pseudo-classifier* (CVSPC), *Leave-One-Out Bayesian Bootstrap pseudo-classifier* (CVBBPC), or a *Leave-One-Out Random Weighting pseudo-classifier* (CVRWPC) Algorithm according to which approach is used for generating the pseudo-training sample. Here, the ‘CV’ stands for Cross-Validation.

Although, the CVSPC algorithm does not perform well here, the CVBBPC and CVRWPC algorithms achieved great progress compared with the BBPC and RWPC

algorithms. Thus, for instance, for class pair (A, E), the means of BBPC and RWPC on 200 trials are 14.9611% and 14.89687% given in TABLE 4.4, while that of CVBBPC and CVRWPC are both 19.78188 given in TABLE 4.6. As the mean of the true error rate is 23.54025, it is obvious that CVBBPC and CVRWPC provided much better estimates than BBPC and RWPC did. This is true not only for class pair (A, E) but also for all of the other class pairs. Although, they are not considered to be the best among all the algorithms discussed this chapter, in the next section we will see that the CVBBPC and CVRWPC algorithms are definitely among the top level performers.

4.5. Summary

In this chapter, two types of algorithms for error rate estimating have been discussed. Algorithms of the first type are referred to as real-sample algorithms because they only use the real training samples to estimate the error rate. As opposed to these, algorithms of the other type are referred to as pseudo-sample algorithms because they generate a set of pseudo-training samples to estimate the error rate.

The conclusions of the experiments on the algorithms are:

- 1) The apparent error estimator is obviously downward biased, and the bootstrap techniques are helpful for correcting the bias.
- 2) The Leave-one-out estimator, on average, has a smaller bias while its standard deviation is larger than those related to the bootstrap techniques. A notable fact is that the Leave-one-out estimator seems robust for all of the class pairs involved in the experiments.

- 3) The bootstrap-related estimators have smaller standard deviations and their biases are also smaller. However, it seems that their performance is effected by the changes of the class pair.
- 4) The pseudo-training sample algorithms work well, if the pseudo-training patterns are used to construct the classifier. To generate pseudo-training patterns, the Bayesian bootstrap or random weighting-like approaches are better than the SIMDAT-like approach.

According to the statistics on the averages and standard deviations of the experiments, it is hard to say which algorithm performs the best. We explain below an alternate method for comparing these algorithms, which may be of great interest.

In each experiment, an algorithm was repeatedly carried out for 200 trials; correspondingly, there were also 200 true errors obtained by using independent testing samples. We now compute whether the estimate given by an algorithm is close to the true error when a training sample is given. This would be an important measurement of the performance of an algorithm. For a large number of the algorithms, the absolute difference between its estimate and the true error was calculated for each trial. TABLE 4.8 – 4.9 give the statistics of the absolute difference for 200 trials.

TABLE 4.8 Absolute difference between estimates and true errors (sample size 8)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
0.632	Mean	8.20128	7.644637	7.331174	6.880708	5.406926	4.018858
	STD	5.993444	5.646829	5.650957	4.997281	4.05843	2.781746
0.632+	Mean	9.034845	8.351023	7.191383	6.928247	5.351249	3.99628
	STD	6.547045	5.921737	5.587428	4.880659	4.032969	2.789631
Basic Bootstrap	Mean	18.65208	15.01745	12.90989	9.028082	6.790447	4.874094
	STD	9.776027	8.314804	7.756232	6.036461	4.399029	3.117815
E0	Mean	9.886876	9.943626	8.54768	8.212337	6.637759	4.362711
	STD	6.63802	6.836052	6.627127	6.023431	4.897164	3.512784
Leave-one-out	Mean	14.32125	12.1795	11.34075	10.287	7.7125	5.2685
	STD	10.26207	8.768119	8.730435	7.685018	6.194523	4.802235
Leave-one-out Bootstrap	Mean	8.67498	8.65594	8.693553	8.112915	6.611603	4.313201
	STD	6.478595	6.405682	6.24546	6.092953	4.827036	3.526287
BBPC	Mean	19.00775	14.80881	12.72788	8.884844	5.686156	3.684469
	STD	6.748866	6.724637	6.059126	5.234548	3.981853	2.367959
RWPC	Mean	18.98213	14.89222	12.74006	8.993563	5.751187	3.665437
	STD	7.01119	6.782092	6.142415	5.221314	4.007166	2.379201
CVBBPC	Mean	10.82138	9.081875	7.9405	6.752125	5.265625	3.897875
	STD	7.09268	6.31021	5.84634	4.988222	4.036345	2.713429
CVRWPC	Mean	10.37188	9.05675	7.951875	6.752125	5.31175	3.9125
	STD	6.962961	6.344934	5.647328	4.988222	4.049173	2.718217

TABLE 4.9 Absolute difference between estimates and true errors (sample size 24)

Method	Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
0.632	Mean	8.181462	6.059487	5.76184	4.984816	3.249423	2.374014
	STD	4.982886	4.531387	3.9619	3.428988	2.412928	1.536563
0.632+	Mean	7.838429	5.299544	4.84954	4.347153	2.828926	2.096018
	STD	3.973984	4.031601	3.559655	2.989113	2.093496	1.440829
Basic Bootstrap	Mean	21.74853	15.27053	13.458	9.803956	5.23203	2.843432
	STD	6.012867	5.016574	4.994424	4.225781	3.191326	1.716013
E0	Mean	4.965949	5.431661	5.152211	4.614903	3.26912	2.609235
	STD	3.552329	3.825483	3.85294	3.437708	2.633243	1.773858
Leave-one-out	Mean	7.3285	6.8905	6.363583	5.5355	3.956084	2.941583
	STD	5.400913	4.938355	5.017844	4.029313	3.216421	2.17678
Leave-one-out Bootstrap	Mean	3.397437	4.616175	4.453939	4.082082	3.211287	2.415943
	STD	2.523541	3.304807	3.284122	3.243581	2.546535	1.790734
BBPC	Mean	16.84915	12.24745	10.80508	7.789479	4.459375	2.352667
	STD	4.289556	4.109218	4.154	3.620973	2.614111	1.619874
RWPC	Mean	16.71529	12.15114	10.69126	7.695417	4.497938	2.404375
	STD	4.378507	4.235016	4.253706	3.726431	2.624747	1.631616
CVBBPC	Mean	8.20425	6.143125	5.553833	4.598667	3.091917	2.311292
	STD	4.979079	4.408269	4.04529	3.313884	2.332204	1.514652
CVRWPC	Mean	8.529791	6.885083	6.474917	5.513208	3.469709	2.226958
	STD	5.427395	4.898712	4.373805	3.693069	2.474363	1.556952

Of course, it is understood that the smaller its absolute difference from the true error, the better the performance of an estimator. From this point of view, the Leave-one-out bootstrap algorithm is definitely better than just the Leave-one-out. It is almost true for the CVBBPC and CVRWPC algorithms. TABLE 4.10 – 4.11 range the performances of the algorithms according to the mean value given in TABLE 4.8 – 4.9.

TABLE 4.10 Performance ranges according to the absolute difference from the true error (sample size 8)

Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
0.632	1	1	2	3	4	6
0.632+	3	2	1	4	3	5
Basic Bootstrap	8	10	10	9	9	9
E0	4	6	5	6	8	8
Leave-One-Out	7	7	7	10	10	10
Leave-One-Out Bootstrap	2	3	6	5	7	7
BBPC	10	8	8	7	5	2
RWPC	9	9	9	8	6	1
CVBBPC	6	5	3	1	1	3
CVRWPC	5	4	4	1	2	4

TABLE 4.11 Performance ranges according to the absolute difference from the true error (sample size 24)

Class Pair	(A, B)	(A, C)	(A, D)	(A, E)	(A, F)	(A, G)
0.632	5	4	5	5	4	5
0.632+	4	2	2	2	1	1
Basic Bootstrap	10	10	10	10	10	9
E0	2	3	3	4	5	8
Leave-One-Out	3	7	6	7	7	10
Leave-One-Out Bootstrap	1	1	1	1	3	7
BBPC	9	9	9	9	8	4
RWPC	8	8	8	8	9	6
CVBBPC	6	5	4	3	2	3
CVRWPC	7	6	7	6	6	2

The statistics above definitely convince us that the bootstrap related algorithms are generally better than the Leave-one-out. It also suggested that the 0.632, 0.632+, CVBBPC or CVRWPC should be first considered, if the training sample size is small; the 0.632, 0.632+, E0 and Leave-One-Out should be considered if the training sample size is relatively large.

CHAPTER 5

BOOTSTRAP-BASED CLASSIFIER DESIGN

5.1 Introduction

Both the Bhattachayya bound and the error rate of a classifier discussed in the previous chapters have similar problems – that of estimating an unknown quantity. In these problems, a question of bias correction arose, and the bootstrap techniques were applied to correct it. Although the details are different, the basic strategies are similar when the bootstrap technique is applied to this kind of problem: it involves sampling from the original training set, and using the bias deduced from the bootstrap sample to estimate the true bias. Generally, the bootstrap technique works well in such a case.

As opposed to this the classifier design problem is a totally different issue as it is not a problem of bias correction. What is needed here is a technique to sharpen the classifier within a given training sample. Therefore, the way to apply the bootstrap technique to the problem of classifier design is different from what was discussed before. Here, the research will focus on how to generate an artificial training sample set from the original one. The known work was done by Hamamoto, Uchimura and Tomita [HUT97] who proposed bootstrap schemes to achieve this. The details of their work makeup the content of Section 5.2. In Section 5.3, some new schemes are introduced, which are mainly based on the pseudo-sample algorithms mentioned in Chapter 4. The experimental results obtained by using these schemes will be discussed in Section 5.4.

5.2 Previous Work

As mentioned above, the question addressed here is one of generating an artificial training sample set when applying the bootstrap technique to the problem of classifier design. Therefore, the schemes provided by Hamamoto [HUT97] are for generating the synthetic training samples used to build the classifier. Below are the four algorithms given in Hamamoto's paper.

Algorithm 5.1 Bootstrapping I

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(v) m , the number of nearest neighbors of a training pattern

Output: $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$, the artificial training sample data.

Method
BEGIN

For $i = 1$ to c do

For $j = 1$ to n_i do

GetNearestNeighboursOf($\underline{x}[j, i], m$)

End-For

End-For

For $i = 1$ to c do

For $j = 1$ to n_i do

$k = \text{random integer in } [1, n_i]$

$\underline{y}[j, i] = \text{GenerateArtificialPattern}(m\text{--nearest neighbors of } \underline{x}[k, i])$

End-For

End-For

Return $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$

END Bootstrapping I

Procedure GenerateArtificialPattern**Input:** $\underline{x}[1], \underline{x}[2], \dots, \underline{x}[m]$ the m -nearest neighbors of the training pattern $\underline{x}$ **Output:** a pseudo-training pattern.**Method****BEGIN** **GetWeights**(m)

$$\underline{y} = \sum_{i=1}^m \underline{w}[i] \times \underline{x}[i]$$

Return $\underline{y}$ **END Procedure GenerateArtificialPattern****Procedure GetWeights****Input:** m , the number of the neighbors of a training pattern;**Output:** $w[1], w[2], \dots, w[m]$, the weights**Method****BEGIN**

sum = 0

For ($j = 1$ to m) $w[j] = \text{random uniform } U[0, 1]$ sum = sum + $w[j]$ **End-For** **For** ($j = 1$ to m) $w[j] = w[j] / \text{sum}$ **End-For** **Return** $w[1], w[2], \dots, w[m]$ **END Procedure GetWeights**

The **GetNearestNeighboursOf** procedure is used to obtain the m -nearest neighbors of a training pattern. Its task is to calculate the Euclidean distances between the given pattern and the others in the same class, sort the distances calculated, and collect the first m patterns having the shortest distances. Note that a pattern itself is included in its m -nearest neighbors. The main procedure in this algorithm is the **GenerateArtificialPattern** procedure that provides the artificial training sample set to construct the classifier. The key point of the algorithm is to use a weighted average of the m -nearest neighbors of a training pattern as an artificial training pattern. Observe that the

weights given by the **GetWeights** procedure is just what was provided by the random weighting method.

Algorithm 5.2 Bootstrapping II

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) m , the number of the nearest neighbours of a training pattern

Output: $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$, the artificial training sample data.

Method
BEGIN
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 GetNearestNeighboursOf($\underline{x}[j, i], m$)
 End-For
 End-For
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 $\underline{y}[j, i] = \text{GenerateArtificialPattern}(m\text{-nearest neighbors of } \underline{x}[j, i])$
 End-For
 End-For
 Return $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$
END Bootstrapping II

The only change between **Algorithm 5.1** and **Algorithm 5.2** is that the step

$k = \text{random integer in } [1, n_i]$

in **Algorithm 5.1** is eliminated. This means that the artificial training sample is just to replace each pattern in the original sample with the weighted average of its m -nearest neighbors.

Algorithm 5.3 Bootstrapping III

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(v) m , the number of the nearest neighbors of a training pattern

Output: $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$, the artificial training sample data.

Method
BEGIN
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 GetNearestNeighboursOf($\underline{x}[j, i], m$)
 End-For
 End-For
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 k = random integer in $[1, n_i]$
 $\underline{y}[j, i]$ = the average of the m -nearest neighbors of $\underline{x}[k, i]$
 End-For
 End-For
 Return $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$
END Bootstrapping III

Unlike **Algorithm 5.2**, this algorithm modifies **Algorithm 5.1** in another way. It replaces an original training pattern directly with the arithmetic average of its m -nearest neighbors; meanwhile, the original patterns are randomly selected so as to give the averages of their m -nearest neighbors.

Algorithm 5.4 Bootstrapping IV

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(v) m , the number of the nearest neighbours of a training pattern

Output: $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$, the artificial training sample data.

Method
BEGIN
 For $i = 1$ to c **do**
 For $j = 1$ to n_i **do**
 GetNearestNeighboursOf($\underline{x}[j, i], m$)
 End-For
 End-For
 For $i = 1$ to c **do**
 For $j = 1$ to n_i **do**
 $\underline{y}[j, i]$ = the average of the m -nearest neighbors of $\underline{x}[j, i]$
 End-For
 End-For
 Return $\underline{y}[1, 1], \dots, \underline{y}[n_1, 1], \underline{y}[1, 2], \dots, \underline{y}[n_c, c]$
END Bootstrapping IV

Algorithm 5.4 changed from Algorithm 5.2 the same way as Algorithm 5.3 changed from Algorithm 5.1. The artificial training sample consists of the arithmetic averages of the m -nearest neighbors for all the patterns in the original sample.

The comparison of the above algorithms was done in the paper by Hamamoto [HUT97] on the bases of the experiments with k -NN classifiers ($k = 1, 3, 5$). The experiments were designed to inspect the performances of the above algorithms in various situations, as well as the different dimensions, training sample sizes, sizes of the nearest neighbors, and distributions. The main conclusions of their work are:

- 1) The algorithms outperform the conventional k -NN classifiers as well as the edited 1-NN classifier.
- 2) The advantage of the algorithms comes from removing the outliers by smoothing the training patterns, e.g. using local means to replace a training pattern.

3) The number of the nearest neighbors chosen has an effect on the result. An algorithm was introduced to optimize the selection of this size.

The details of the experiments and the algorithm for optimizing the size of the nearest neighbor set can be found in [HUT97].

5.3 Mixed-sample Classifier Design

In Chapter 4, we presented the pseudo-classifier algorithms, which used a pseudo-training sample to test the accuracy of the classifier. It is generally accepted that the larger a training sample, the greater the accuracy of a classifier. Consequently, we believe that using the pseudo-training sample set to enlarge the training sample size is a good idea. The question that we face is one of knowing how to generate suitable pseudo-training patterns and blending them together with the original training samples to build a classifier. Such a classifier design scheme is referred to as a *Mixed-sample* classifier design because it uses the mixed training sample to construct the classifier. Generally, this kind of algorithm will build a classifier in two steps: first, it will generate a pseudo-training sample set; then it will build a classifier with all the training patterns using both of the original and the pseudo-training samples.

The second step will be executed based on the classifier selected. This can involve any standard classifier design algorithm based on the given training patterns. We will thus focus on how to carry out the first step. Earlier, in Chapter 4 we presented a **GeneratePseudoSample** procedure. We shall now marginally modify an existing

procedure and make it suitable for the purposes of the classifier design. Below is a typical algorithm for generating a pseudo-training sample for the classifier design.

Algorithm 5.1 Generating Pseudo-training sample

Input: (i) c , the number of classes;
(ii) n_i , the size of the training sample of class i ;
(iii) $\underline{x}[1, 1], \dots, \underline{x}[n_1, 1], \underline{x}[1, 2], \dots, \underline{x}[n_c, c]$, the training sample data;
(iv) B , the repeated times of drawing bootstrap samples.
(v) m , the number of the nearest neighbours of a training pattern
(vi) t_i , the size of the pseudo-training sample of class i ;
Output: $\underline{y}[1, 1], \dots, \underline{y}[t_1, 1], \underline{y}[1, 2], \dots, \underline{y}[t_c, c]$, the pseudo-training sample.
Method
BEGIN
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** n_i **do**
 $\text{GetNearestNeighboursOf}(\underline{x}[j, i], m)$
 End-For
 End-For
 For $i = 1$ **to** c **do**
 For $j = 1$ **to** t_i **do**
 $k = \text{random integer in } [1, n_i]$
 $\underline{y}[j, i] = \text{GetPseudoPattern}(m\text{-nearest neighbours of } \underline{x}[k, i])$
 End-For
 End-For
 Return $\underline{y}[1, 1], \dots, \underline{y}[t_1, 1], \underline{y}[1, 2], \dots, \underline{y}[t_c, c]$
END Generating Pseudo-training sample

The **GetPseudoPattern** procedure in the above algorithm can be either the **GetPseudoPattern I** procedure given in **Algorithm 4.6** or the **GetPseudoPattern II** procedure given in **Algorithm 4.7**. Therefore, it is possible to have different weighting schemes for the **GetPseudoPattern** procedure. In our experiments, we used three different schemes: the SIMDAT algorithm, the Bayesian bootstrap and the random weighting methods.

There are other parameters that have to be decided as well in this algorithm. These are the number of the nearest neighbors and the size of the pseudo-training sample

set of each class. It is preferred to give a fixed rate between the pseudo-training sample size and the original training sample size for each class. As expected, the pseudo-training sample size affects the performance of the classifier. In our experiments, several pseudo-training sample sizes were used to ascertain their effect on the performance of a classifier. The size of the nearest neighbor set can also affect the performance of a classifier. Two different sizes of the nearest neighbor set were used in our experiments to show how they effect the classifier's performance.

5.4 Simulation Results

Experiments for the Mixed-sample algorithms were done with the Data Set I. Two classes each with a training sample of size 8 were involved in each experiment. Six different sizes: 4, 8, 12, 16, 20 and 24, were used to generate the pseudo-training sample sets. With the size of the nearest neighbors $m = 3$, three schemes namely, the SIMDAT algorithm, the Bayesian bootstrap and the random weighting method were used to generate the pseudo-training samples. To study the effect of the size of the nearest neighbors, the experiments were also carried out with the size of the nearest neighbors $m = 8$. Note that the training sample size of a class is 8, so $m = 8$ means that a pseudo-training pattern is a combination of all the patterns in the training sample of a class. The Bayesian bootstrap and the random weighting method were used in the experiments with $m = 8$. To distinguish between the Bayesian bootstrap and the random weighting method used in two different sizes of the nearest neighbors, the previous ones are referred to as the Local-Bayesian bootstrap and the Local-random weighting method while the latter

are simply called the Bayesain bootstrap and the Random weighting method. After constructing a classifier with the mixed-sample, an independent testing sample of size 1,000 for each class was used to estimate the error rate of the 3-NN classifier. Figure 5.1 – 5.6 provide the results of the experiments on an average of 200 trials.

Figure 5.1 Testing Error of Mixed-Sample Classifier-Class Pair (A, B)

Figure 5.2 Testing Error of Mixed-Sample Classifier-Class Pair (A, C)

Figure 5.3 Testing Error of Mixed-Sample Classifier-Class Pair (A, D)

Figure 5.4 Testing Error of Mixed-Sample Classifier-Class Pair (A, E)

Figure 5.5 Testing Error of Mixed-Sample Classifier-Class Pair (A, F)

Figure 5.6 Testing Error of Mixed-Sample Classifier-Class Pair (A, G)

The X-axis in each chart represents the pseudo-training sample size of a class, while the Y-axis represents the error rate of the classifier estimated by the independent testing sample. Each chart gives the experiment's results on one class pair. Although,

different class pairs have a different error rate; the experiment results showed consistent tendencies, which are

1. All the three algorithms for pseudo-training sample generation seem to make things worse. These are the SIMDAT algorithm, the Local-Bayesian bootstrap and the Local-random weighting method.
2. The Bayesian bootstrap and random weighting method apparently improve the performance of the classifier.
3. The effect of the pseudo-training sample increases with its size increment.

For instance, without a pseudo-training sample mixed into the original training sample for the classifier construction, the error rate of the 3-NN classifier would be 33.37% for class pair (A, D). When the size of the pseudo-sample increased to 16, twice of the training sample size, the classifier's error rates increased to 34.41%, 33.82% and 34.15% for the Local-Bayesian Bootstrap, the Local-random weighting method and the SIMDAT algorithm respectively. However, with the same pseudo-training sample size, the error rates decreased to 31.08% and 31.14% for the Bayesian bootstrap and the random weighting method. This is also true for class pair (A, E). Without the pseudo-training sample, the error rate of the 3-NN classifier is 23.54%. The error rates rose to 24.14%, 23.59% and 25.14% for the Local-Bayesian Bootstrap, the Local-random weighting method and the SIMDAT algorithm respectively. At the same time, the error rates decreased to 21.61% and 21.70% for the Bayesian bootstrap and the random weighting method.

The failure of the Local-Bayesian Bootstrap, the Local-random weighting method and the SIMDAT algorithm indicates that the size of the nearest neighbors should not be too small. Although, a combination of m -nearest neighbors smoothed the training patterns in some way; it might still be an outlier if it is the combination of m -nearest neighbors of an outlier and the size, m , is small. Thus, the mixed-sample classifier might have more outliers than that of the usual classifier, which makes things even worse. It is noted that the SIMDAT algorithm is the worst one of all the schemes for class pairs (A, E), (A, F) and (A, G). The cause for this is that the SIMDAT algorithm uses the uniform distribution $\mathcal{U}[\frac{1}{m} - \sqrt{\frac{3(m-1)}{m^2}}, \frac{1}{m} + \sqrt{\frac{3(m-1)}{m^2}}]$ to generate the weighting vectors, which allows for a greater chance for an outlier point to be generated.

The algorithms of the Bayesian bootstrap and the random weighting method performed very well as they use combinations of whole training patterns to produce the pseudo-training sample. This confirms that the size of the nearest neighbors should not be too small. The effect of the pseudo-training sample size is also significant. It can be seen from Figure 5.1 – 5.6 that, for all class pairs, the larger the pseudo-training sample's size, the lower the error rate. The evidence show that the rate of decrease slows down when the pseudo-training sample's size increases. Generally, it is enough to take the size of the pseudo-training sample as 1.5 times (in our case it is $m = 12$) as that of the original training sample. The results of the experiments also showed that there is no significant performance difference between the Bayesian bootstrap and the random weighting method.

5.5 Discussions and Conclusions

In this chapter, the classifier design problem has been discussed. Previously, Hamamoto and the others constructed a classifier by using the weighting local means in place of the original training patterns. Their experiments demonstrated that their strategy performed favorably. With the local means strategy, they also discussed how to find the optimized size of the nearest neighbors.

What introduced by us is an alternative approach for constructing a classifier with the mixed-sample. The motivation for using the mixed-sample classifier design was to have a larger sample size. This approach works when the pseudo-training patterns are obtained from the weighting averages of all the original training patterns together. However, it fails if the size of the nearest neighbors is small.

Comparing the results of the mixed-sample classifier design to the previous results of Hamamoto's work, it may be seen that there are still interesting problems to be discussed.

First, in terms of the mixed-sample classifier design, the classifier designs suggested by Hamamoto can be referred to as pseudo-training sample designs as they use only the pseudo-training samples – the local means – to construct a classifier. The difference between the mixed-sample and the pseudo-training sample classifier designs is that the latter uses the pseudo-training sample to replace the original one while the former mixes the two samples together.

Second, the local means function well in the pseudo-training sample classifier design because the local means removed the outliers by smoothing the patterns. However,

they perform badly in the mixed-sample classifier design when the size of the nearest neighbors is small, as a bad local mean might be an extra outlier. On the other hand, a large size of the nearest neighbors, such as the whole set of the training sample, works in the mixed-sample classifier design.

Third, Hamamoto's work proved that a large size for the set of nearest neighbors used does not imply that it yields the best results [HUT97]. That is why an algorithm to optimize the size of the nearest neighbors was suggested. Hence, it appears as if the problem of optimizing the size of the nearest neighbors for the mixed-sample classifier design is unanswered. This could be a problem for further study.

Fourth, our experiments have proved that using a pseudo-training sample to enlarge the sample size is a good strategy. As no work has been done to compare the mixed-sample classifier design with the pseudo-training sample classifier design, it is difficult to say which one is better. It is reasonable, though, to use the original training patterns in the construction of a classifier if they are beneficial.

Finally, from all of the above discussions, it is confirmed that the classifier design can be improved on by using a pseudo-training sample. The problem is how to avoid the contamination of bad pseudo-patterns. There are two possible solutions to the problem: one is to exclude the outliers from the original training sample set before generating the pseudo-training samples. The other is to replace the outliers in the original training sample with their local means. Although it is believed that these two approaches are able to enhance the classifier design, further studies are needed to prove the assumption.

CHAPTER 6

SUMMARY AND DISCUSSION

6.1 Introduction

This thesis has represented an in-depth study on Efron's bootstrap techniques. It also discussed how the technique should be applied to problems in statistical pattern recognition, namely those of estimating the Bhattacharyya error bounds, estimating classifier error rates and designing classifiers. All the works done in the area, including the work presented here, demonstrate that the bootstrap technique is applicable to and works well for problems in the field of statistical pattern recognition. Throughout our discussion, we have proposed the use of three main strategies for the application of the bootstrap technique, namely, bias correction, cross-validation and pseudo-pattern generation. We now discuss how these strategies were applicable in statistical pattern recognition.

6.2 Bias Correction

As explained in Chapter 2, Efron's bootstrap scheme was initially proposed to estimate the bias of an estimator. The simulation of the distribution $\hat{F}^*$ given by the bootstrap sample, to experimentally represent the empirical distribution $\hat{F}$ is used to provide us with the information about the way $\hat{F}$ simulates the true distribution F . Thus the bias of a parameter θ defined in (2.1) can be estimated as:

$$\text{Bias} = E^* [\hat{\theta}^* - \hat{\theta}] = E^* [\theta(\hat{F}^*) - \theta(\hat{F})].$$

A bootstrap estimate of θ is obtained by using the bias estimate to correct the original estimate,

$$\hat{\theta}_{\text{BOOT}} = \hat{\theta} - \mathbf{Bias} . \quad (6.1)$$

Bias correction is obtained by merely applying (6.1).

There are two different ways of applying the bias correction strategy: direct or indirect. The direct bias correction is to apply (6.1) directly on the parameter estimated. This approach works well for estimating the Bhattacharyya bound. As shown in Chapter 3, it improved the estimates for all of the six class pairs. Direct bias correction also works for estimating the error rate of a classifier. To do this, the basic bootstrap, 0.632, and 0.632+ estimators use the direct bias correction, even though their methods to estimate the bias are different. The performance of a direct-bias-correction estimator depends largely on how well the bias estimate does. For instance, in estimating the error rate of a classifier, all of the schemes, namely, the basic bootstrap, 0.632 and 0.632+ algorithms are based on the bias correction of the apparent error estimate. The performances of the three algorithms vary because they estimate the bias in different ways. Therefore, it is very important to find a good bias estimator. Unfortunately, however, there are no general rules that can be followed so as to yield a superior bias estimator.

When the parameter θ is a function of another parameter ξ : $\theta = f(\xi)$, it is possible to correct ξ 's bias first, and then to obtain the estimate of θ via the Taylor's expansion of the function $\theta = f(\xi)$. This approach is called the indirect bias correction. In Chapter 3, this approach was used to estimate the Bhattacharyya bound. Although the FAMS algorithm with the bootstrap scheme improves the estimate significantly, the standard

variance of the estimate is very large. For example, in TABLE 3.7, the mean and the standard deviation for the 200 trials are 20.63062% and 23.34433% for class pair (A, G), while, in TABLE 3.10, they are 33.01038% and 43.78814% for the same class pair, implying that the results are not very satisfactory. On the other hand, the FAMS algorithms with the Bayesian bootstrap and the random weighting method schemes have a relatively small standard deviation, but they do not correct the bias significantly. In TABLE 3.8 with the Bayesian bootstrap scheme, for example, the mean and the standard deviation of the class pair (A, G) are 9.294273% and 4.665902%. The fact is that there is a large gap between the basic bootstrap schemes and the Bayesian bootstrap scheme, as well as between the basic bootstrap and the random weighting method. There seems to also be a trade-off between the bias correction and the standard deviation. The question of whether we can devise a scheme that can have the advantages of both the basic bootstrap and the Bayesian bootstrap remains open.

Comparing the indirect bias correction with the direct bias correction, it is not easy to state which is superior. The experimental results given in Chapter 3 show that the direct bias correction performs better for class pairs (A, B), (A, C), (A, D) and (A, E), while the indirect bias correction performs better for class pairs (A, F) and (A, G). The cause for this is that the Bhattacharyya bound ϵ_u is an exponential function of the Bhattacharyya distance $\mu(1/2)$, $\epsilon_u = \frac{1}{2} e^{-\mu(1/2)}$. It must also be noted that the Bhattacharyya distance $\mu(1/2)$ is usually over-estimated. Hence the estimate bias of the Bhattacharyya distance will contribute more to the estimate bias of the Bhattacharyya bound when the

Bhattacharyya distance is large. Therefore, the larger the Bhattacharyya distance is, the larger its effect on the estimate bias of the Bhattacharyya bound will be.

6.3 Cross-validation

The cross-validation approach is mainly concerned with the error rate estimation of a classifier. The term “cross-validation” as used by us in this thesis does not have the same meaning as in machine learning. The key idea in the Cross-validation algorithm is to separate the testing pattern from the training patterns. It might, therefore, be better to describe the scheme using the phrase “Training and testing pattern separating approach” instead of using the term “Cross-validation”. All of the methods, namely E0, 0.632, Leave-one-out bootstrap, 0.632+ estimators, CVBBPC and CVRWPC use the Cross-validation to estimate the error rate. Compared with the other error rate estimation algorithms, which do not separate the training and testing samples, these algorithms obviously come out as the winners.

In small sample cases, however, an explicit separation does not necessarily mean the best. TABLE 4.8 and TABLE 4.10 indicate that when the training sample size is 8, the impure separating estimators: 0.632, 0.632+, CVBBPC and CVRWPC estimators performed apparently better than the pure separating estimators: the E0 and Leave-one-out bootstrap estimators. On the other hand, TABLE 4.9 and TABLE 4.11 show that the performance of the E0 and the Leave-one-out bootstrap estimators improved significantly. These facts demonstrate us that the size of the training sample set is an important factor. When some patterns were excluded from the training sample for testing,

it effectively reduced the size of the training sample set to construct a classifier, and so the error rate of the classifier increased. This explains why the error rate estimates given by Leave-one-out, E0 and Leave-one-out bootstrap tend to over-estimate the true error rate.

When the size of the training sample set is large, the result will not be so effected when we merely exclude one pattern each time from the training sample for testing. TABLE 4.9 and TABLE 4.11 clearly demonstrate this fact. However, when the size of a training sample is small, it is hard to design, test and train the classifier with separate patterns. One of the solutions is to use a pseudo-classifier algorithm, such as CVBBPC or CVRWPC. Of course, the Leave-one-out bootstrap performs very well in the small training size case, but a pseudo-classifier algorithm is preferred for the following reasons:

1. A pseudo-classifier algorithm cannot be worse than the Leave-one-out bootstrap. Thus, for example, the performance of CVBBPC and CVRWPC are comparable to the Leave-one-out bootstrap.
2. A pseudo-classifier algorithm is more flexible than the Leave-one-out bootstrap as it is possible to improve its performance by changing either the method of generating a weighting vector or by changing the number of the nearest neighbors of a training pattern used in designing the algorithm.

The first reason has been supported by our experiments. The details of the second reason will be studied in the next section.

6.4 Pseudo-pattern Generation

Pseudo-classifier algorithms are used for estimating the error rate of a classifier, and the pseudo-training sample can be used in classifier design. In spite of the difference between their purpose, both the pseudo-classifier algorithms and the pseudo-training sample algorithms are based on the same strategy, namely, that of pseudo-pattern generating. Of course, the strategies for generating the pseudo-patterns vary depending on the purpose for which pseudo-patterns are generated.

Two issues are important when generating pseudo-patterns. The first is the schemes used to produce the weighting vectors. In our experiments, three schemes were tested: the SIMDAT algorithm, the Bayesian bootstrap and the random weighting method. Generally, the latter two perform better although, in error rate estimation, the SIMDAT algorithm has a better performance for the class pairs (A, B), (A, C) and (A, D) (see TABLEs 4.4 - 4.7). It is therefore clear, that a suitable scheme must be chosen based on the situation at hand. Unfortunately, at this time, there are no rules available for such a selection. For the error rate estimation, our experiments showed that the SIMDAT algorithm is better when the error rate is higher, and the Bayesian bootstrap and the random weighting method are better with a lower error rate.

The second issue of importance is number of the nearest neighbors of a pattern used in generating the pseudo-patterns. In error rate estimation, pseudo-patterns are used to simulate the behavior of the training patterns. So it is better to keep the size of the nearest neighbors small. In this way, the distribution of the pseudo-patterns can be kept

close to that of the original training sample. As opposed to this, in classifier design, the purpose of adding the pseudo-patterns is to decrease the classifier's error. So the size of the set of nearest neighbors of a pattern should be kept larger. In this way, the pseudo-patterns will tend to appear more frequently in the central vicinity of each class, and so bring about more smoothness to the training samples.

In the estimation of error rates, it is unlikely that we will be able to design an algorithm that can optimize the size of the set of nearest neighbors because of the lack of practical criterion function. The goal of the exercise is to compute an estimate that is as close to the true error rate as possible. However, this problem is complex because the true error rate is unknown in practice. A possible solution for the optimization is to choose the size of the set of nearest neighbors depending on the results of simulation experiments.

In the classifier design, however, the Leave-one-out error estimation can be implemented to optimize the size of the nearest neighbors. The algorithm is briefly described below:

1. For a fixed size of nearest neighbors, generate pseudo-patterns and construct the classifier with the Mixed-sample classifier design discussed in Chapter 5.
2. Use the Leave-one-out algorithm to estimate the error rate of the classifier built in Step 1.
3. Do Steps 1 and 2 for three different sizes of nearest neighbors, $m_1 = 2 < m_2 = (m_1 + m_3) / 2 < m_3 =$ the training sample size of a class. Compare the estimated error rates.

4. Let Err_i be the estimated error rate related to size m_i . If $\text{Err}_1 < \text{Err}_2 < \text{Err}_3$, then select a $m_0 = (m_1 + m_2) / 2$; if $\text{Err}_1 > \text{Err}_2 > \text{Err}_3$, then select a $m_0 = (m_2 + m_3) / 2$; if $\text{Err}_1 > \text{Err}_2$ and $\text{Err}_2 < \text{Err}_3$, then select $m_{01} = (m_1 + m_2) / 2$ and $m_{02} = (m_2 + m_3) / 2$.
5. Do Steps 1 and 2 for the newer selected size m_0 , and compare the newer estimated error rate Err_0 with the two others (Err_1 and Err_2 if $m_0 = (m_1 + m_2) / 2$; Err_2 and Err_3 if $m_0 = (m_2 + m_3) / 2$). If there are two newer sizes m_{01} and m_{02} selected in Step 4, then two new error rates Err_{01} and Err_{02} will be estimated, and the comparison would be carried out with Err_2 , Err_{01} and Err_{02} .
6. Repeat Steps 3 – 5, until the three compared sizes converge. The m with the minimum Cross-validation error would be selected as the size of the nearest neighbors.

Although we cannot guarantee that the best size for the set of the nearest neighbors will result, this algorithm can, at least, provide a reasonably good estimate for the cardinality of the set.

6.5 Conclusion

In this thesis we have demonstrated, conclusively, that the bootstrap technique is useful for solving problems in statistical pattern recognition. Generally, algorithms using the bootstrap technique take more space and more time for calculations, which are the main disadvantages of the bootstrap technique. However, space and time are not serious problems in estimating the Bhattacharyya bound, and in estimating the error rate, as the

size of the training sample is not generally big, and the estimates are only calculated once. There are also no problems of space and time in the classifier design. Once the classifier has been constructed with the mixed-samples, it will be used the same way as the classifier designed by the usual approach.

The benefits of the bootstrap technique are obvious:

1. It corrects the estimate bias of the Bhattacharyya bound and classifier error rate, and
2. It improves the performance of a classifier.

Therefore, the question now is not “Could the bootstrap technique be applied to statistical pattern recognition?” but rather, “Which is the best bootstrap algorithm applicable?” and “How much can the bootstrap technique achieve?”.

Based on our research, the following algorithms for the bootstrap technique are suggested for the three problems in statistical pattern recognition, which we have researched.

To estimate the Bhattacharyya bound, the bias correction approach with the basic bootstrap scheme is preferred. Either the direct or indirect bias correction approach can be used depending on the Bhattacharyya distance associated with the class pair. If the Bhattacharyya distance is small, it is preferred that the direct bias correction be used. Otherwise the indirect bias correction would be better.

To estimate the error rate of a classifier, a combined Cross-validation and bootstrap approach is recommended, such as the 0.632+, the Leave-one-out bootstrap, the CVBBPC or the CVRWPC. For the pseudo-classifier algorithms CVBBPC and

CVRWPC, the size of the nearest neighbors can be selected as $m = 3$, as demonstrated by our experiments. As mentioned in the previous section, the pseudo-classifier algorithm with the SIMDAT scheme can also be a good candidate, if the true error rate is high.

To design a classifier, a mixed-sample algorithm with the Bayesian bootstrap or the random weighting method is preferred. The size of the set of nearest neighbors can be the same as the size for the training samples of a class. Our experiments proved that this size works well for the classifier design.

It is difficult to say just how much advantages a bootstrap technique can give. Although the methods recommended above have achieved high performances, there is still potential to achieve more progress by applying the idea of pseudo-pattern generation and the bootstrap technique. Two kinds of improvements seem possible. The first is to seek a better scheme to generate the weighting vectors for a specific problem. The other is to optimize the size of the set of nearest neighbors of a pattern. Both of these avenues are areas where much research can be done. The question of using parametrical bootstrap techniques for the three problems studied remains open.

REFERENCE

- [BD84] Bunke, O. and Droge, B. (1984), "Bootstrap and cross-validation estimates of the prediction error for linear regression models", *Annals of Statistics*, 12, pp.1400-1424.
- [Bu89] Burman, P. (1989), "A comparative study of ordinary cross-validation, v -fold cross-validation and the repeated learning-testing methods", *Biometrika*, 76, pp.503 - 541.
- [CMN85] Chernick, M.R., Murthy, V.K. and Nealy, C.D. (1985), "Application of bootstrap and other resampling techniques: evaluation of classifier performance", *Pattern Recognition Letters*, 3, pp.167 - 178.
- [CMN86] Chernick, M.R., Murthy, V.K. and Nealy, C.D. (1986), "Correction to 'application of bootstrap and other resampling techniques: evaluation of classifier performance'", *Pattern Recognition Letters*, 4, pp.133 - 142.
- [DH92] Davison, A. C. and Hall, P. (1992), "On the bias and variability of bootstrap and cross-validation estimates of error rate in discrimination problems", *Biometrika*, 79, pp.279 - 284.
- [DH97] Davison, A. C. and Hinkley, D. V. (1997), *Bootstrap methods and Their Application*, Cambridge University Press, Cambridge.
- [Ef79] Efron, B. (1979), "Bootstrap Method: Another Look at the Jackknife", *Annals of Statistics*, 7, pp.1 - 26.
- [Ef82] Efron, B. (1982), *The Jackknife, the Bootstrap and Other Resampling Plans*, CBMS-NSF, Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics.
- [Ef83] Efron, B. (1983), "Estimating the error rate of a prediction rule: improvment on cross-validation", *Journal of the American Statistical Association*, 78, pp.316 - 331.
- [Ef86] Efron, B. (1986), "How biased is the apparent error rate of a prediction rule?", *Journal of the American Statistical Association*, 81, pp.461 - 470.
- [ET93] Efron, B. and Tibshirani, R. J. (1993), *An introduction to the Bootstrap*, Chapman & Hall, Inc.

- [ET97] Efron, B. and Tibshirani, R. J. (1997), "Improvement on cross-validation: the .632 + bootstrap method", *Journal of the American Statistical Association*, 92, pp.548 - 560.
- [Fu90] Fukunaga, K. (1990), *Introduction to Statistical Pattern Recognition, Second Edition*, Academic Press, Inc.
- [Ha86] Hand, J.D., (1986), "Recent advances in error rate estimation", *Pattern Recognition Letters*, 4, pp.335 - 346.
- [HUT97] Hamamoto, Y., Uchimura, S. and Tomita, S. (1997), "A bootstrap technique for nearest neighbor classifier design", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19, pp.73 - 79.
- [JDC87] Jain, A.K., Dubes, R.C. and Chen, C. (1987), "Bootstrap techniques for error estimation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9, 628 - 633.
- [Li78] Li, K.-C (1978), "Asymptotic optimality of C_p , C_l , cross-validation and generalized cross-validation: discrete index set", *Annals of Statistics*, 15, pp.958 -975.
- [Ma92] Mammen, E., (1992), *When Does Bootstrap Work? - Asymptotic Results and Simulations*, *Lecture Note in Statistics* 77, Springer - Verlag, New York, Inc.
- [Po94] Port, Sidney C., (1994), *Theoretical Probability for Applications*, John Wiley & Sons, Inc.
- [ST95] Shao, J. and Tu, D., (1995), *The Jackknife and Bootstrap*, Springer - Verlag, New York, Inc.
- [TT92] Taylor, M. S. and Thompson, J. R., (1992), "A nonparametric density estimation based resampling algorithm", *Exploring the Limits of Bootstrap*, John Wiley & Sons, Inc., pp.397 - 404.
- [WJ95] Wand, M. P. and Jones, M. C. (1995), *Kernel Smoothing*, Chapman & Hall, London.
- [We91] Weiss, S.M. (1991), "Small sample error rate estimation for k-NN classifiers", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13, pp.285 - 289.

-
- [Zh87] Zhen, Z. (1987), "Random weighting methods", *Acta Math. Appl. Sinica*, 10, pp.247 - 253.