Robust Methods in Data Mining

Mwitondi, K. S.

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds
Department of Statistics

October 2003

Acknowledgements

Many institutions and individuals contributed significantly, and in many different ways, to the accomplishment of this thesis. For their invaluable financial support, during the early stages of my research, I will always remain indebted to Messrs. Dilip Kesaria, Abdul Zakaria and Mohamed Mbaye of Dar Es Salaam, Northern Foods PLC, UNESCO, IFM and Sidney Perry Foundation for their contribution towards my first year tuition fees. My appreciations also go to the Churches Commission for International Students, African Educational Trust, the Department of Statistics and Welfare Services at the University of Leeds for the significant contribution they made towards my third year academic fees, at the time when almost all hope was lost.

I am deeply indebted to my supervisors Dr. Charles C. Taylor and Professor John T. Kent whose close supervision, assistance and advice helped me through different hurdles — both academic and non-academic. Many thanks to all those who, at different stages, discussed my problems and provided advice — in particular, Professor Kanti Mardia, Dr. Robert Aykroyd and Ms. Amina Mohamed Ali. I am particularly grateful to my colleagues — Yousef, Salem, Jamal and Fathi who, among other things, ensured that I worked comfortably during after-office hours, by guaranteeing transport to and from the University. Thanks are also due to all other colleagues and staff at the School of Mathematics for their co-operation. Some would make coffee or just help me put on a smiling face — I thank them all without exception.

My wife, Eyssa, and our children — Said, Mussa, Nasra and Ashura, deserve special thanks for their patience, love and encouragement that always kept me going. Thousands of miles away — in three continents — our parents, brothers, sisters, relatives and friends deeply missed us and constantly prayed for our safe return home. We thank them all for the inspirational moments which added a great sentimental value to our daily lives away from home. I am grateful to Sheikh Ahmed Islam of Dar Es Salaam, whose moral support has inspired me in many ways over the years. Last, but not least, all thanks are due to our God — the exalted, the creator and sustainer of the universe — who has urged us to remember him so that he can remember us.

Dedicated to my parents, Said Mwitondi and Taous Mousa

For the pangs they endured for not seeing me for four years.

Abstract

The thesis focuses on two problems in Data Mining, namely clustering, an exploratory technique to group observations in similar groups, and classification, a technique used to assign new observations to one of the known groups. A thorough study of the two problems, which are also known in the Machine Learning literature as *unsupervised* and *supervised* classification respectively, is central to decision-making in different fields — the thesis seeks to contribute towards that end.

In the first part of the thesis we consider whether robust methods can be applied to clustering — in particular, we perform clustering on fuzzy data using two methods originally developed for outlier-detection. The fuzzy data clusters are characterised by two intersecting lines such that points belonging to the same cluster lie close to the same line. This part of the thesis also investigates a new application of finite mixture of normals to the fuzzy data problem.

The second part of the thesis addresses issues relating to classification — in particular, classification trees and boosting. The boosting algorithm is a relative newcomer to the classification portfolio that seeks to enhance the performance of classifiers by iteratively re-weighting the data according to their previous classification status. We explore the performance of "boosted" trees (mainly stumps) based on 3 different models all characterised by a sine-wave boundary. We also carry out a thorough study of the factors that affect the boosting algorithm.

Other results include a new look at the concept of randomness in the classification context, particularly because the form of randomness in both training and testing data has directly affects the accuracy and reliability of domain-partitioning rules. Further, we provide statistical interpretations of some of the classification-related concepts, originally used in Computer Science, Machine Learning and Artificial Intelligence. This is important since there exists a need for a unified interpretation of some of the "landmark" concepts in various disciplines, as a step forward towards seeking the principles that can guide and strengthen practical applications.

Contents

Chapter 1

Introduction

1.1 An overview and scope of the thesis

Various classes of patterns that can be detected in data are known to present a potential source of useful information in business, commerce, medicine, industry and other social and scientific fields. Indeed, contemporary data analysis is largely concerned with detecting patterns in data, telling them apart and performing prediction. The thesis focuses on two main problems in *Data Mining* — clustering and classification, also known as *unsupervised* and *supervised* classification respectively.

In this work, we introduce some clustering and classification approaches and investigate robust alternative approaches. We also provide statistical interpretations of some of the classification-related concepts, originally used in different areas of Computer Science, such as Machine Learning and Artificial Intelligence, thus demonstrating the often "not too evident" link between statistics and those fields of study. In practice, most applications integrate the above listed disciplines and statistics. Some insight into the interface between Machine Learning and Statistics is provided by Nakhaeizadeh and Taylor (1997), focusing on classification — the main common area of intersection between the two disciplines. Preliminary versions of parts of the thesis appeared in Taylor and Mwitondi (2001) and Mwitondi *et al.* (2002).

1.1.1 Some of the basic concepts in the thesis

The following concepts present a terminological foundation for the thesis.

1. Robust Methods — The main idea of robustness is the construction of estimators that are approximately efficient, if the parametric model holds, but at the same time not grossly sensitive to small departures from the assumed model. The reduction of variation is central to the improvement of accuracy, reliability and efficiency.

2. Data Mining — Also known as Knowledge Discovery in Databases, Data Mining may be viewed as an extension of applied statistics relying on abundantly available computing power. For instance consider a commercial bank that needs to decide who should receive financial credit. The information operated on by the data mining process is contained in a historical database of previous interactions with customers and the features associated with the customers, such as annual income, age, marital status, sex, educational level and debt history. Such historical information can be used to build a model of customer behaviour that could be used to predict which customers are unlikely/likely to default on payments. Clearly, a primary problem in Data Mining is how to organise observed data into meaningful structures.

3. Learner — A fundamental concept in classification, referring to an algorithm that takes a dataset of examples, also called a *training* set, as input and uses it to transform the sample space into a partition. In the classification context, the main output of a learning algorithm is called a classifier. A classifier predicts class membership for each new vector of inputs. One way of distinguishing a learner from a classifier is by recognising that the former describes the path from the training data to a partition, while the latter describes the partition.

1.1.2 Clustering

The first part of the thesis considers two methods which we investigate as possible approaches to clustering. The first approach adapts methods originally designed for outlier detection to sequentially detect clusters in data. In particular, we use the EM algorithm (Dempster *et al.*, 1977) and Atkinson's Forward search algorithm (Atkinson, 1994) to detect clusters in data. The former fits a t-model, treating one of the clusters as "good" and the other as "bad", while the latter considers part of

the data as normally distributed and the remaining part as "outlying".

The second approach, in the first part of the thesis, uses the EM algorithm to fit a finite mixture of normals model, hence simultaneously detecting clusters in data. We use simulated data with known classes so that we can compare clustering results with the known classes in order to assess the performance of the algorithm.

1.1.3 Classification

The second part of the thesis mainly uses classification trees in conjunction with the boosting technique. Different tree sizes, ranging from a single split, also known as a "stump", to a full-grown tree, affect the final results of a classification problem. Typically, the number of misclassified examples is inversely related to the tree size — with larger trees yielding lower errors and vice versa. In practice, *training* and *test* data sets are used to produce and test the rule respectively.

A trade-off between size and error is usually made by, say, cross-validation — a method for estimating generalisation error based on re-sampling results. The resulting error estimates are often used for choosing the optimal size from the resulting set of different tree sizes. Generalisation error refers to a test set error which, typically, provides guidance in choosing a classifier of appropriate complexity.

Boosting

The boosting algorithm, due to Schapire (1990), is a classifier enhancement tool that works by iteratively re-weighting the data and combining the resulting classifiers to form the final consensus classifier. Its basic ideas originally appeared in the Computational Learning Theory literature through works of Schapire (1990), Freund (1995) and Freund and Schapire (1997). However, the method has its roots in the "Probably Approximately Correct" (PAC) model in Machine Learning, pioneered by Valiant (1984). Since its introduction, the boosting technique has been praised for not only yielding high classification accuracy, but also being resistant to over-fitting (Freund and Schapire, 1997; Friedman *et al.*, 2000). The thesis provides a detailed study of the method against the background of the existing literature.

This chapter is organised as follows. Section 1.2 provides the general background on clustering and introduces the methods adapted in the thesis. Chapter 1.3 gives an overview of classification and introduces a new look at the concept of randomness in classification performance assessment. Section 1.4 provides the general layout of the thesis - summarising it on a Chapter-by-Chapter basis, with Chapters 2 and 3 addressing clustering issues and 4 through 6 focusing on classification.

1.2 Background on clustering

Cluster analysis is a specialised branch of statistics that, typically, arises in the absence of *a priori* hypotheses. It is an exploratory technique that provides a description or a reduction in the dimension of the data, from n observations to k groups or clusters. Its main idea is to group data into k mutually exclusive and previously unknown subsets. It builds up the groups in such a way that the profiles of objects in the same groups are relatively homogeneous whereas the profiles of objects in different groups are relatively heterogeneous.

1.2.1 Selected standard clustering techniques

Distance-based clustering represents a family of popular clustering techniques. The most straightforward way of computing distances between objects in a multi-dimensional space is to use Euclidean distances, which are computed as follows

$$D(x_i, x_j) = \left\{ \sum_{k=1}^{p} (x_{ik} - x_{jk})^2 \right\}^{\frac{1}{2}}, \tag{1.1}$$

where $x_i = (x_{i1}, ..., x_{ip})$ and $x_j = (x_{j1}, ..., x_{jp})$ are $p-$dimensional vectors. The Euclidean distance is usually computed from raw data. One of the advantages of distance-based methods is that inter-object distances are not influenced by the addition of new objects, which may be outlying. Its major disadvantage is that it is affected by the measurement scales — that is, if one of the dimensions is in kilobytes and it is converted to bytes, the results in (1.1) computed from multiple dimensions are greatly affected. An easy solution would be to adopt a Mahalanobis distance.

Different clustering techniques have been studied. Some of the most popular standard clustering methods are the K-means (McQueen, 1967), histogram and the Kernel Density. The former appears in different variations, including the popular variant by Hartigan and Wong (1979), and can be summarised as follows. Given k assumed known clusters in a data set, assign each data item to the known clusters in such a way that the resulting cluster-means are as distinct as possible.

Central to the mechanics of the K-means algorithm are the measure of closeness, number of classes and the starting point. Exploratory approaches to clustering, such as histogram and Kernel Density Estimation are associated with smoothing parameters that affect the number of clusters detected. Using these methods entails choosing the appropriate smoothing parameter — a non-trivial issue which, in most applications, may result in either too many or too few clusters being detected. One way to tackle this problem is by using the method of cross-validation.

1.2.2 Our clustering techniques

The techniques described in Section 1.1, which we adapt for clustering, can be categorised as *sequential* and *simultaneous*. The former, collectively describing the EM algorithm for scaled mixtures of normals and Atkinson's forward search algorithm, seeks to detect the clusters one at a time, while the latter, referring to the EM algorithm for finite mixture models, aims at a simultaneous detection of the clusters. The sequential methods are similar in that in both cases they seek to find "good" data, whereas the simultaneous method tries to find the groups simultaneously.

The novelty in the sequential techniques lies in the application twist from outlier-detection to clustering. At the same time, the simultaneous method is a fairly standard clustering tool, but it is presented here with a new fuzzy example. One of the sequential methods and the simultaneous method are powered by the EM algorithm — a maximum-likelihood estimating tool, that is geared towards problems involving missing-data. The algorithm consists of two steps, namely, the EXPECTATION step and the MAXIMISATION step, thus accounting for its name. Basically, the E-step computes the log-likelihood of the complete-data problem, typically as its

conditional expectation given the observed data. The likelihood is then maximised in the M-step. The two steps are iterated until convergence.

The rationale for the scaled mixtures of normals is that the t-distribution provides a good extension of the *normal* distribution for longer-than-normal tailed errors, which enhances its capability in computing robust estimators. The EM algorithm iteratively computes ML estimators, particularly under incomplete-data situations and it is used as a *tool* to compute the MLEs for the t-distribution. Given degrees of freedom for the t-distribution, the EM algorithm reduces to a weighted least squares method by iteratively down-weighting outlying observations.

The second of the sequential methods is a graphically informative tool; its power derives from its ability to detect outlying observations in data sets. This property is particularly important in clustering as bad observations have a serious effect on most clustering techniques, including being the source of spurious clusters.

1.3 Background on classification

The classification problem is based on discriminant analysis, a technique used to determine how variables discriminate between the k known groups. Effected under different methods, classification provides a predictive tool for classifying previously unseen cases and can be viewed as the partitioning of the measurement space Ω. The ultimate goal of classification is to minimise the misclassification error. Another issue of concern is the rate at which the error is being driven down as $n \to \infty$.

If the densities and class priors were known, the classification problem would be trivial — simply apply Bayes' rule to compute the posterior probabilities of class membership for each new case. In practice, however, the densities are unknown and so the major issue in classification is to devise an accurate and reliable rule.

1.3.1 Selected standard classification techniques

A wide range of classification techniques have been studied in the literature, including Linear Discriminant, Classification Trees (Breiman *et al.*, 1984), K-Nearest Neighbour, Support Vector Machines (Vapnik, 1995) and Neural Networks (Bishop, 1995). The main concern arising from using each of the foregoing classifiers is over how the rule performs on previously unseen data. High variance of the estimated error and over-fitting are two of the main issues that arise in classification.

The boosting algorithm (Schapire, 1990), a relative newcomer to the classification portfolio, is a classifier enhancement tool and, in principle, can be used in conjunction with each of the above-mentioned classification techniques. Resistance to over-fitting has been singled out as one of its main distinctive properties.

1.3.2 Assessment of performance in classification

The general performance of any classifying algorithm is assessed by its misclassification error, which reflects its rate of *accuracy* and *efficiency*. Typically, the two properties are measured in terms of how close the algorithm gets to the Bayesian error and the amount of time and data it requires.

For the purposes of assessing the performance of classifiers, we propose a new look at the concept of randomness in the classification context. From a statistical perspective, we delve further into the features that affect performance, such as data distributions, size, structure and class boundaries. Our examples are based on data simulated to reflect the foregoing features. Classification examples in the thesis are confined to *categorical*, also known as *nominal* rather than ordered classes.

1.4 Layout of the thesis and the main issues

The thesis is presented in two parts, with chapters 2 and 3 focusing on clustering tools and the remaining three chapters addressing issues relating to classification. More precisely, Chapter 4 addresses two classical classification methods — LDA and Classification Trees (Breiman *et al.*, 1984). In chapters 5 and 6, we use boosting

with trees — mainly tree stumps, to classify data with a sine-wave boundary. The novelty in this part of the thesis is that we present a challenge to the linear methods of discrimination, by way of the sine-wave class boundary and vary the levels of overlap, under different models, to further challenge the boosting algorithm.

Chapter 2

We apply the EM algorithm and Atkinson's Forward search, originally designed for outlier-detection, to detect clusters in data. The underlying strategy is to use these two methods to sequentially identify clusters in data. Applications are based on two datasets — the first is a classical clustering problem in which there are well-separated groups with distinct means. The second dataset has clusters which are characterised by two intersecting lines such that points belonging to the same cluster lie close to the same line. In a Data Mining context, such clustering may be envisioned as a relationship between crop yield and temperature, such that one of the lines represents dry and the other wet, with the moisture being unobservable.

We produce some interesting results, for both the univariate and regression settings. The key result for clustering, using the first of the sequential methods, is the number of local maxima of the likelihood function as the relevant parameters are varied. Detection of the clusters in both cases happens to be dependent on these parameters as well as the data structure. As far as separation of the two clusters is concerned, the second of the sequential methods — Atkinson's Forward Search — turns out to be far more successful than the first — that is, the MLE for the t-distribution.

Chapter 3

Chapter 3 describes an alternative use of the EM algorithm. The strategy here is to fit a finite mixture of normals, whereby each component of the model corresponds to a cluster. More specifically, the goal in this chapter is to simultaneously detect the two lines of the line-clustered data used in Chapter 2, rather than sequentially detect one at the expense of the other.

Assumptions of both restricted and unrestricted group variances are made in each

case. Results are based on both univariate and bivariate data — the method works well if the variances are known to be equal. However, for the rather challenging example of intersecting clusters, the assumption of unrestricted variance may lead to undesirable results — implying that assumptions made about the distributions of the data would, typically, impinge on the performance of the algorithm. It seems reasonable, for clustering purposes involving similar problems, to make an assumption of equal group variance as the assumption adds stability to the algorithm.

Chapter 4

This chapter is mainly a review of some classical classification methods, with illustrations based on LDA and classification trees. Its main contribution is the proposed new look at the concept of randomness in the error estimation context. The relationship between the twin concepts of *learning* and *training* on the one hand and performance on the other is emphasised. We identify four ways to estimate the classification error and test the rule as

1. Test on population, i.e. using a notionally infinite test set.

2. Cross-validation.

3. Test data set.

4. Plug-in or re-substitution approach.

With the distinction between *theoretical* and *empirical* rules clearly highlighted, the randomness associated with the four empirical errors can arise from two sources — randomness due to the allocation rule depending on, typically random, training data and randomness due to basing the assessment on random test data. Using one of the four generalisation error estimates above, we give a statistical definition of the Machine Learning concept of *reliability* — as simply the difference between the expected population error estimate and the Bayesian error.

Chapter 5

The chapter is devoted to classification by boosting (Schapire, 1990), a classification enhancement tool used in conjunction with "poorly performing" algorithms, such

as tree stumps. Its underlying idea derives from early studies in Machine Learning by Valiant (1984), in which "rough" classifiers are coined *weak learners.* In this chapter, we provide a statistical definition and illustration of the Machine Learning concept of *weak learnability*, based on a notional discrete population.

Most classification rules use well-segmented and ambiguous data equally, which may lead to swamping and/or masking effects. We simulate data in which the Bayes' rule follows a sine-wave boundary as a natural challenge to the linear methods of discrimination. Using the data and "stump boosting", we then proceed to investigate the behaviour of boosting under various scenarios of overlap.

We track down the role of case weights close to the sine-wave boundary and farther from it and find that boosting may also be used as a diagnostic tool to detect ambiguously located observations. We investigate whether it is possible to determine the optimal complexity of the classifier to boost. Our results suggest that although boosting tree stumps works well, moderately large trees may work equally well. In both cases, data structure and size have a role to play.

Chapter 6

Built on the ideas developed earlier in chapters 4 and 5, the chapter investigates some of the factors affecting boosting, such as population densities, class priors, levels of overlap and sample size. We carry out a simulation study using sine-wave boundary data and use numerical integration to estimate the population error, which is then used to assess the impact of the various factors listed in Section 1.3.2.

We perform a thorough comparison of two probability models based on multiple simulations at different levels of overlap and show that, contrary to the assertion (Freund and Schapire, 1997; Friedman *et al.*, 2000), boosting can still over-fit under a non-overlap scenario.

Chapter 2

Outlier-resistant methods applied to clustering

2.1 Introduction

In this chapter we consider outlier-resistant methods applied to clustering. We adapt two methods, originally developed for detection of outliers in data sets, namely the MLE for the t-distribution (Lange *et al.*, 1989) and Atkinson's Forward Search algorithm (Atkinson, 1994) respectively, to clustering problems. We present the former as a long-tailed distribution represented as a scale of mixture normals and used to compute MLEs via the EM algorithm. The underlying idea is that long-tailed distributions accommodate the existence of outliers and can cope with the existence of multiple maxima. We use the latter as a diagnostic tool to determine cluster centres.

We mainly focus on the two concepts of "outliers" and "clusters", viewed here as closely related. Outliers are influential observations, the deletion of which can lead to dramatic changes in naive statistical estimates and data analysis conclusions. In particular, when different samples are taken, conclusions are likely to be unreliable mainly because the values and frequency of outliers are likely to vary with the samples. Such situations entail the development and application of robust procedures aimed at providing *resistant* outcomes in the presence or absence of outliers.

Detecting outliers in both univariate and multivariate data analysis is an important

aspect of robust statistical analysis. Inherent in the process of outlier detection are the twin problems of "masking" and "swamping". Masking occurs when bad observations go undetected because of the existence of another, usually adjacent, subset of observations and swamping occurs when good observations are detected as outliers because of the existence of another, usually remote, subset. On the other hand, clusters can generally be defined as different clumps of data within a given data set. Cluster analysis arises in the absence of *a priori* information and seeks to put data items in k distinct groups, where k is often unknown.

This chapter is organised as follows. An overview of clustering methods is given in Section 2.2, while Section 2.3 provides the general description of the problem to be tackled and the adopted methodology. Section 2.4 introduces the EM algorithm for the scaled mixture of normals, and gives the mechanics of the algorithm as well as some applications. Section 2.5 covers Atkinson's forward search algorithm, the mechanics of the method, its underlying theory, motivation and some applications.

2.2 Clustering - *A general overview*

As we noted earlier, clustering seeks to reduce the data into a small number of similar sub-groups. Typically, clustering problems will involve large and complex datasets, hence the need for automated tools in detecting structures inherent in data. Two main classes of clustering methods are known in the literature, namely *hierarchical* and *partitioning* clustering methods. Each of the two can be sub-divided into different groups, typically using different cluster-detection algorithms.

2.2.1 Hierarchical and Partitional clustering techniques

Hierarchical clustering techniques adopt both bottom-up and top-up approaches. In the former the algorithm proceeds successively merging smaller clusters into larger ones, while in the latter larger clusters are split into smaller ones. Different methods apply different rules in determining which two small clusters are merged or which large cluster is split. The end result of the algorithm is a tree of clusters called a dendrogram, which shows how the clusters are related. By cutting the dendro-

gram at a desired level a clustering of the data items into disjoint groups is obtained.

Partitional clustering techniques, on the other hand, adopt a domain-partitioning approach. The methods seek to perform a direct decomposition of the dataset into a set of disjoint and distinct clusters. The algorithm's criterion function to be minimised may emphasise the local structure of the data — as by allocating clusters to peaks in the probability density function — or the global structure. Typically the global criteria involve minimising some measure of dissimilarity in the samples within each cluster, while maximising the dissimilarity of different clusters.

One of the most widely used partitional clustering method is the K-means clustering method (McQueen, 1967). The method relies on a criterion function that is an average squared distance of the observations from their nearest cluster centroids. One possible algorithm for minimising the K-means criterion begins by initialising a set of K cluster centroids and adjusting their positions iteratively by first allocating observations to the nearest clusters and then recomputing them. The iteration terminates when changes in the criterion die away. An alternative algorithm considers each randomly chosen sample in succession, updating the nearest centroid.

2.3 Description of the problem

Consider data sets with multiple groups; in particular, a one-dimensional problem in which there are different groups. If the groups are well-behaved they can be described with, say, normal distributions. In the cluster analysis problem, we do not know how many groups exist and we look for statistical methods to separate these groups from one another. The LHS of Figure 2.1 provides a graphical bimodal form of the type of data structure we are trying to detect. In a regression setting a two-group data set is given on the RHS of the same figure. The choice of the dataset was motivated by the particularly interesting form of the clusters as compared to other forms of clusters such as the lunar-star and embedded circles type of clusters.

Our main strategy is to separate "good" from "bad" data, where the "good" data lie in one of the clusters and the "bad" data lie in the remaining clusters. Of course

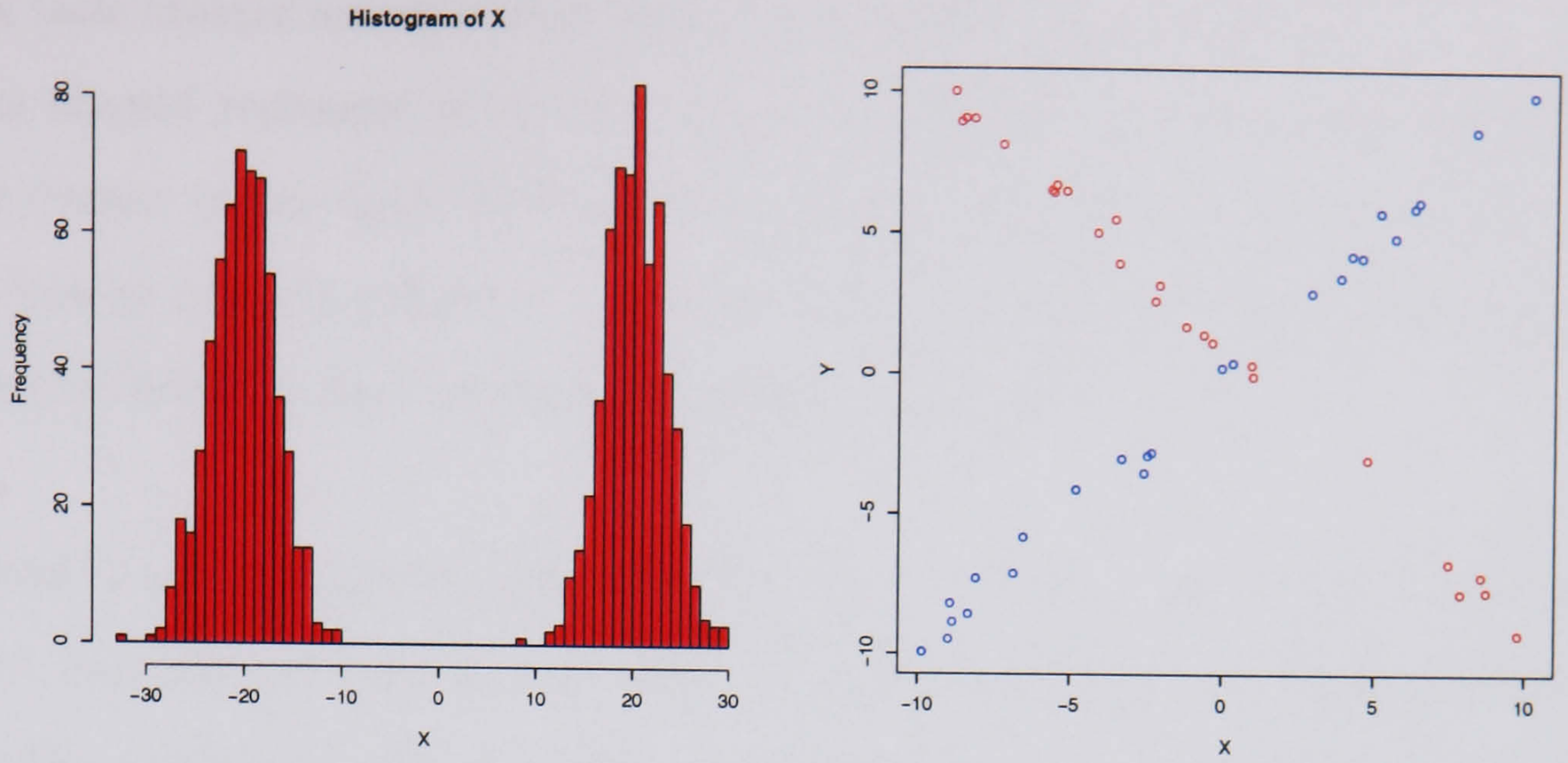


Figure 2.1: *The two panels show examples of data sets comprising two clusters. On the LHS is a univariate data set and on the RHS is a bivariate data set.*

depending on the initial choice of a main cluster, the definition of "good" and "bad" can change. The objective on the LHS of Figure 2.1 is to separate the two modes while on the RHS it is to separate the two arms of the data. Although we have adopted a two-class setting here, the setting can readily be extended to a $k-$class scenario by adopting a kind of step-wise partitioning of the data set. Ideally, one way of coping with more than two clusters is by removing the "good" subset from the experiment and examining the remaining "bad" subset for further structures.

2.3.1 A brief description of the methodology

As noted earlier, we consider the following methods in the general context of being potential tools in tackling multiple class data problems.

1. A long-tailed distribution represented as a scaled mixture of normals and used to compute MLEs via the EM algorithm.

2. Atkinson's Forward Search algorithm for outlier detection (Atkinson, 1994).

The two methods are similar in that they try to tell "good" and "bad" data apart. They typically find one cluster at a time, while treating the other points as outliers.

Method 1 was motivated by the fact that the t-distribution provides an extension of the normal distribution for thicker-than-normal tailed errors. The existence of

thick tails creates susceptibility to multiple local maxima. Ideally, each local maximum should represent a cluster centre and the global maximum should find the main cluster in the data. Fitting the model provides the idea as to what and where the clusters are. As a further separation step, the tails can be trimmed off and the model re-fitted to the remaining, hopefully clean, data.

Method 2 is a graphically informative tool; its power derives from its ability to detect bad observations in data sets. Although its mechanics fundamentally differ from the mechanics of the scaled mixture of normals, the overall objective of the two methods is quite similar. Given an example of a data set involving two groups of data, the forward search seeks to distinguish "good" from "bad" data and in so doing it is bound to reject one of the groups in favour of the other. In Sections 2.4.4 and 2.5.2, the two methods are applied on the bi-cluster problems in Figure 2.1.

2.4 EM algorithm for scaled mixture of normals

In this section we show how the t-distribution can be represented as a scale of mixture of normal distributions and how the EM algorithm can be used to estimate location and/or scale parameters. Using the t-distribution in modelling contaminated data is plausible not only because it is a parametric model deriving from the normal, but also because it yields robust estimators. To compute these robust estimators, we can apply a simple and comprehensive tool such as the EM algorithm.

The EM algorithm for the t-distribution allows weights to be incorporated in the iterative procedure. We investigate the properties of the EM algorithm for the t-distribution in handling *bad-looking* data. Our main focus will be on the performance of the algorithm, given fixed degrees of freedom and bimodal or multi-modal data. We wish to investigate how much information about the data can be recovered or lost — expressed in terms of the number of detected modes in the data.

2.4.1 The EM algorithm and the problem of missing data

The EM algorithm (Dempster *et al.*, 1977) is an iterative method that is widely applied in computing maximum likelihood estimates, especially in cases of *incomplete data.* Although in its conventional definition the EM algorithm seems to be confined to cases of *incomplete data*, the concept of *incompleteness* is not used in its absolute sense. A wide range of cases of *incomplete data* exist, in which the computation of MLEs becomes complicated — for instance, models with *contaminated* data.

The underlying mechanics of ML estimation are favourable to complete-data problems from an exponential family distribution. Particularly for the normal distribution, ML estimation is made easier since both the mean and the variance can be computed explicitly. The exponential family distributions possess properties that are favourable to the computation of MLEs, hence convenient for making inferences.

The EM algorithm involves two steps: **E**xpectation and **M**aximisation, thus accounting for its name. The **E**xpectation step generates data for the *complete-data* problem based on the *incomplete-data* problem and the **M**aximisation step is subsequently applied to the *complete-data* problem. In other words, the *log-likelihoods* are generated by the E-step and maximised by the M-step. The steps **E** and **M** are then repeated several times until convergence. Based on the mechanics of ML estimation, the EM algorithm can be thought of as a mechanism that transforms an *incomplete-data* problem into a *complete-data* problem, so that the computation of the MLEs is made possible in the M-step of the algorithm. A detailed account of the EM algorithm is provided in McLachlan and Krishnan (1996). We illustrate the mechanics of the algorithm in the subsequent pages as well as in Chapter 3.

2.4.2 The t-distribution

The t-distribution is defined as follows. Let z be a standard random variable, i.e. $z \sim \mathrm{N}(0,1)$ and let w be a chi-square random variable with ν degrees of freedom, i.e. $w \sim \chi^2_\nu$. Then if z and w are independent, $t = z/\sqrt{w/\nu}$ is said to follow a t-distribution with ν degrees of freedom and the *pdf*

$$f(t) = c\left(1 + \frac{t^2}{\nu}\right)^{-\frac{(\nu+1)}{2}} \tag{2.1}$$

where c is the normalising constant depending on ν, the specific value of which being of no interest. Note that $t|w \sim N(0, \nu/w)$ and so t follows a mixture (over w) of normals. The density function, (2.1), is centred about $\mu = 0$. If we introduce location and scale parameters by setting $y = \mu + \sigma t$, then y has *pdf*

$$f_\nu(y) = \frac{c}{\sigma}\left(1 + \frac{\left(\frac{y-\mu}{\sigma}\right)^2}{\nu}\right)^{-\frac{(\nu+1)}{2}}. \tag{2.2}$$

Small values of ν imply thicker tails and as $\nu \to \infty$, the t-distribution tends towards the normal. Lange *et al.* (1989) propose to replace the normal by the t-distribution as the error term in regression model, thus fitting the parameters of the model in an outlier-resistant fashion. Allowing ν to vary between small and large values, makes it possible to use it as a tuning parameter for outlier detection.

Theoretical foundations of the steps of EM algorithm for the t-distribution

Let y_i $(i = 1, ..., n)$ be a random sample such that $(y_i - \mu)|\xi_i \sim \mathrm{N}(0, \sigma^2/\xi_i)$, where ξ_i is an *iid* sample from the density $f(\xi_i)$. If, for instance, $\xi_i \sim \chi^2_\nu/\nu$, the marginal distribution of the random sample $(y_i - \mu)/\sigma$ follows a t-distribution with ν degrees of freedom $[y_i \sim t_\nu(\mu, \sigma)]$, say. If all the parameters of ξ_i are known, then the density of the complete data function will form an exponential family with sufficient statistics $\sum_{i=1}^n y_i^2\xi_i$, $\sum_{i=1}^n y_i\xi_i$, and $\sum_{i=1}^n \xi_i$. For observed ξ_i, the MLEs are given as

$$\hat{\mu} = \frac{\sum_{i=1}^n y_i^2\xi_i}{\sum_{i=1}^n \xi_i} \quad \text{and} \quad \hat{\sigma}^2 = \frac{\sum_{i=1}^n (y_i - \hat{\mu})^2\xi_i}{n}. \tag{2.3}$$

If the ξ_i are unobserved, they need to be estimated by the expected value of ξ_i given y_i and the initial values, $\mu^{(m)}$ and $\sigma^2_{(m)}$. We know that $y_i|\xi_i \sim \mathrm{N}(\mu, \sigma^2/\xi_i)$ and since the situation we have is $\xi_i \sim \chi^2_\nu/\nu$, and not $\xi_i \sim \chi^2_\nu$, we need a transformation in which to express the distribution of ξ_i with the appropriate expected value and variance. To that end, we let $\xi_i = W/\nu$, where $W \sim \Gamma(\nu/2, 1/2)$.

Note that $\xi_i = W/\nu$, which implies that $W = \nu\xi_i \implies \partial W/\partial\xi_i = \nu$. This means that $f(\xi_i) = f_w(\xi_i\nu)\partial W/\partial\xi_i$, which affects the variance for the distribution of ξ_i, thus becoming $\xi_i \sim \Gamma(\nu/2, \nu/2)$. Further, note that $f(\xi_i|y_i) = f(y_i|\xi_i)f(\xi_i)/f(y_i)$,

and that if we can recognise the distribution of the numerator we will know the denominator. Working only with the numerator we obtain

$$f(\xi_i|y_i) \propto \frac{(\nu\xi_i)^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{1}{2}}}\exp\Big\{-\frac{1}{2\sigma^2}(y_i-\mu)^2\nu\xi_i\Big\}\nu\xi_i^{\frac{\nu}{2}-1}e^{-\frac{\nu\xi_i}{2}} \tag{2.4}$$

$$f(\xi_i|y_i) \propto (\nu\xi_i)^{\frac{\nu-1}{2}}\exp\Big\{-\frac{1}{2\sigma^2}\nu\xi_i\Big[(y_i-\mu)^2+\sigma^2\Big]\Big\}, \tag{2.5}$$

which can be recognised as the density of $\Gamma[(\nu+1)/2, (s_i+\nu)/2] = \Gamma(\alpha,\beta)$ distribution, say, where $s_i = (y_i-\mu)^2/\sigma^2$ denotes the scaled residual with the current parameter estimates. We can obtain α from the equation $(\nu-1)/2 = \alpha-1 \Longrightarrow \alpha = (\nu+1)/2$. Hence we obtain a Gamma density with $\alpha = (\nu+1)/2$ and $\beta = [(y_i-\mu)^2+\nu\sigma^2]/2\sigma^2$, which can be simplified further to $\beta = [(y_i-\mu)^2/\sigma^2+\nu]/2$, giving the expected value

$$\mathrm{E}(\xi_i|y_i) = \frac{\alpha}{\beta} = \frac{\nu+1}{\frac{(y_i-\mu)^2}{\sigma^2}+\nu} = \frac{\nu+1}{s_i+\nu} = w_i. \tag{2.6}$$

Note that the expression in (2.6) represents a weight criterion that is a non-increasing function of the squared residuals, s_i, and hence down-weights outlying observations.

The EM algorithm involves two main steps — the **E**-step computes the log-likelihoods and the **M**-step maximises them. From the initial estimates, $\mu^{(0)}$ and $\sigma^{(0)}$, and with m denoting the current iteration; the EM algorithm proceeds as follows.

1. The **E**-Step: Set $w_i^{(m)}$ as in (2.6)

2. The **M**-Step: Compute the estimates $\mu^{(m+1)}$ and $\sigma^2_{(m+1)}$ as

   - $\mu^{(m+1)} = \frac{\sum_{i=1}^n y_i w_i}{\sum_{i=1}^n w_i}$
   - $\sigma^2_{(m+1)} = \frac{\sum_{i=1}^n w_i[y_i-\mu^{(m+1)}]^2}{n}$

The two steps are iteratively repeated until convergence. As $m \to \infty$, the estimates $\mu^{(m)}$ and $\sigma^2_{(m)}$ converge to a local maximum of the likelihood for $\hat{\mu}$ and $\hat{\sigma}^2$. It is sometimes necessary to try different starting points to locate all the local maxima. More details of the EM algorithm can be found in McLachlan and Krishnan (1996).

The above algorithm is presented for unknown σ^2. A similar algorithm works for known σ^2, i.e. by holding the parameter constant and just updating μ. Kent *et al.* (1994) show that if σ^2 is known, the shape of the log-likelihood function, hence the

MLE values, will depend just on the composite parameter $\nu\sigma^2$, with large and small values of $\nu\sigma^2$ yielding unimodal and multi-modal likelihood functions respectively.

If σ^2 is unknown, the shape of the log-likelihood will depend on ν, with small values of ν, typically, yielding more local maxima than large values of ν. Indeed, for $\nu \geq 1$ in the univariate case, the MLE is unique. That is, the log-likelihood function has a unique local maximum. On the other hand, as $\nu \longrightarrow 0$, the number of modes in the log-likelihood tends to n, i.e. the number of data points.

2.4.3 The EM algorithm for linear regression with t-errors

We now consider the regression case based on the model with t-errors, defined as

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \text{where} \quad \epsilon_i \sim t_v. \tag{2.7}$$

In this case, we can summarise the EM algorithm for the t-distribution as follows:

1. Initialise the parameters $\hat{\beta}_0^{(m=0)}$, $\hat{\beta}_1^{(m=0)}$ and $\hat{\sigma}^2_{(m=0)}$.

2. The **E**-Step: Using the current values of $\hat{\beta}_0^{(m)}$, $\hat{\beta}_1^{(m)}$ and $\hat{\sigma}^2_{(m)}$, compute/update the scaled squared residuals and the weights as follows

$$s_i^{(m)} = \frac{(y_i - \hat{y}_i^{(m)})}{\hat{\sigma}^2_{(m)}}, \tag{2.8}$$

$$w_i^{(m)} = \frac{(\nu + 1)}{(\nu + s_i^{(m)})}. \tag{2.9}$$

   where $\hat{y}_i^{(m)}$) denotes fitted y_i at the m^{th} step and ν are the degrees of freedom.

3. The **M**-Step Using the current weights generated in step 2, update the parameters by weighted least squares as follows

$$\hat{\beta}_1^{(m+1)} = \frac{\sum_{i=1}^n w_i(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^n w_i(x_i - \overline{x})^2}. \tag{2.10}$$

$$\hat{\beta}_0^{(m+1)} = \frac{\sum_{i=1}^n y_i w_i - \sum_{i=1}^n \beta_1^{(m+1)} x_i w_i}{\sum_{i=1}^n w_i} = \overline{y}_w^{(m)} - \beta_1^{(m+1)} \overline{x}_w^{(m)}. \tag{2.11}$$

$$\hat{\sigma}^2_{(m+1)} = \frac{\sum_{i=1}^n (y_i - \beta_0^{(m)} - \beta_1^{(m)} x_i)^2 w_i}{n} = \frac{\sum_{i=1}^n (y_i - \hat{y})^2 w_i}{n}. \tag{2.12}$$

4. Repeat steps 2 and 3 until convergence.

2.4.4 Local maxima and the convergence behaviour of the EM algorithm

In this section the EM algorithm for the t-distribution is applied to the two data sets illustrated in Figure 2.1. In the univariate case on the LHS of the figure, the objective is to locate the two modes, while in the bivariate case on the RHS we seek to separate the two arms of the data. In both cases, each mode of the log likelihood is to be interpreted as a cluster. In general terms, estimating the group means and the regression parameters in Figure 2.1 is tantamount to detecting the two clusters in the data. It is in this context that we use the local MLEs, i.e. local maximum of likelihoods, for the t-distribution as a clustering method.

When two clusters exist, as is the case here, we refer to the occurrence of multiple local maxima and global maximum in the log-likelihood function as *swamping* and *masking* respectively. The effect of masking, in this context, refers to the loss of information in the data, as the two modes are fused together into one mode. The effect of swamping refers to the phenomenon in which spurious clusters are detected.

Univariate bimodal data

We consider the problem of estimating the mean in a simple univariate case of a mixture of two normals as given on the LHS of Figure 2.1. The data are generated as follows: Simulate $n_1 \sim B(n, \pi_1)$ and set $n_2 = n - n_1$. Then, given n_1, simulate the first random variable $X_i \sim N(\mu_1, \sigma^2),\ i = 1, ..., n_1$ and the second, $X_i \sim N(\mu_2, \sigma^2),\ i = n_1 + 1, ..., n_1 + n_2$ and define the mixture of two normals

$$\pi_1 N(\mu_1, \sigma^2) + \pi_2 N(\mu_2, \sigma^2). \tag{2.13}$$

For the mixed model in (2.13), the Mahalanobis distance, $D = |\mu_1 - \mu_2|/\sigma$, describes the separation between the two clusters, with a large D indicating that the two modes are well separated and a small D indicating that they are not.

Shapes of the t log-likelihood for different values of $\nu\sigma^2$, for an assumed known $\sigma^2 = 10$, corresponding to the data in the LH panel of Figure 2.1, are given in Figure 2.2. Varying the composite parameter, $\nu\sigma^2$, determines the behaviour of the log-likelihood. The plots are informative and their main purpose is to see if the log-likelihood function has a local maximum or maxima near any of the two clusters

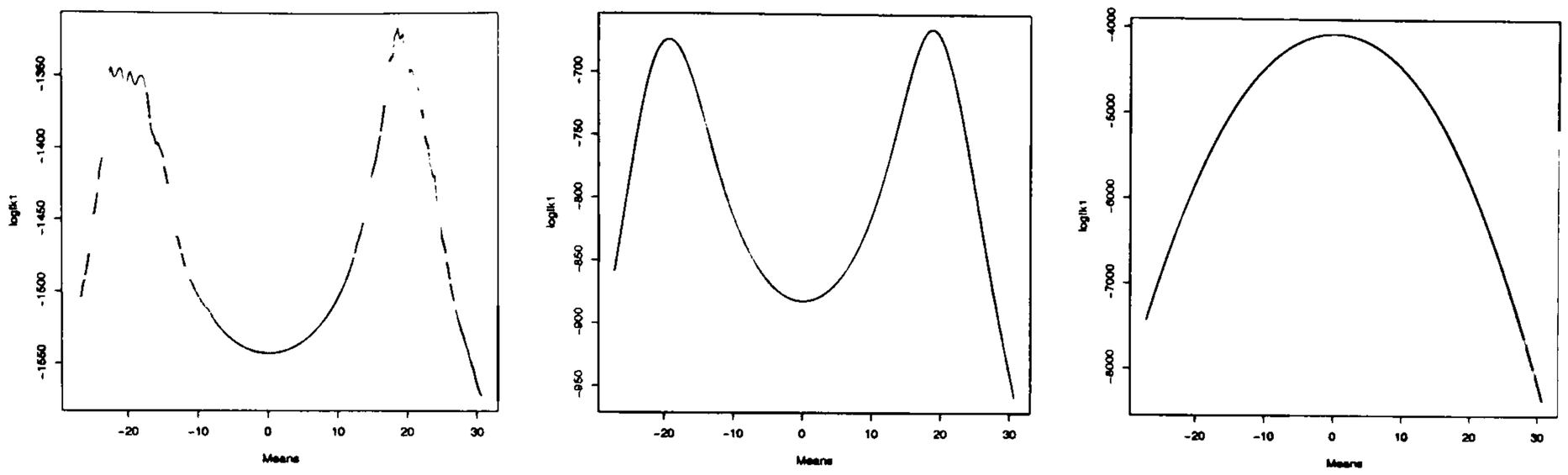


Figure 2.2: *Plots of the t log-likelihood against μ for known σ^2, based on the univariate data in Figure 2.1 and different values of $\nu\sigma^2$. The LH panel corresponds to $\nu\sigma^2 = 0.001$ and reveals multiple local maxima near both modes. The bimodal middle plot corresponds to $\nu\sigma^2 = 10$ and appears to best capture the two modes of the data. The unimodal RH plot is based on very large $\nu\sigma^2$ — resulting into masking effects.*

of the data. The LHS panel in Figure 2.2 exhibits multiple local maxima about the modes, which may be viewed as spurious clusters. Although swamping is hardly evident here, setting $\nu\sigma^2$ much smaller will clearly show it. The single global maximum on the RHS panel implies loss of information about the data.

Figure 2.3 plots the composite parameter, $\nu\sigma^2$, on a log-scale against estimated means, and summarises the performance of the algorithm for the LHS data in Figure 2.1. The plot is based on $n_1 = 317$, $n_2 = 283$, $\mu_1 = -20, \mu_2 = 20$ and two different values of D. The outer red lines correspond to $D = 12.65$ while the inner blue lines corresponds to $D = 4.62$. In both cases, multi-modality is evident for smaller values of $\nu\sigma^2$, but more pronounced in the inner plot, based on small D. In our other examples, choices of grossly imbalanced n_ks led to unimodality, while widely separated clusters bore features of masking and swamping for extreme values of $\nu\sigma^2$.

Note that the plot corresponding to a smaller D is almost fully nested within the plot corresponding to a larger D. The range of bi-modality over $\nu\sigma^2$ is shorter for the plot with a smaller D but its unimodality and multi-modality ranges are longer than those generated by a larger D. Unimodality occurs at about $\log \nu\sigma^2 = 5$ and about $\log \nu\sigma^2 = 8$ for the smaller and larger D respectively. As $D \to 0$, we expect to see a diminishing separation range, with an increasing inner-nesting of the plots.

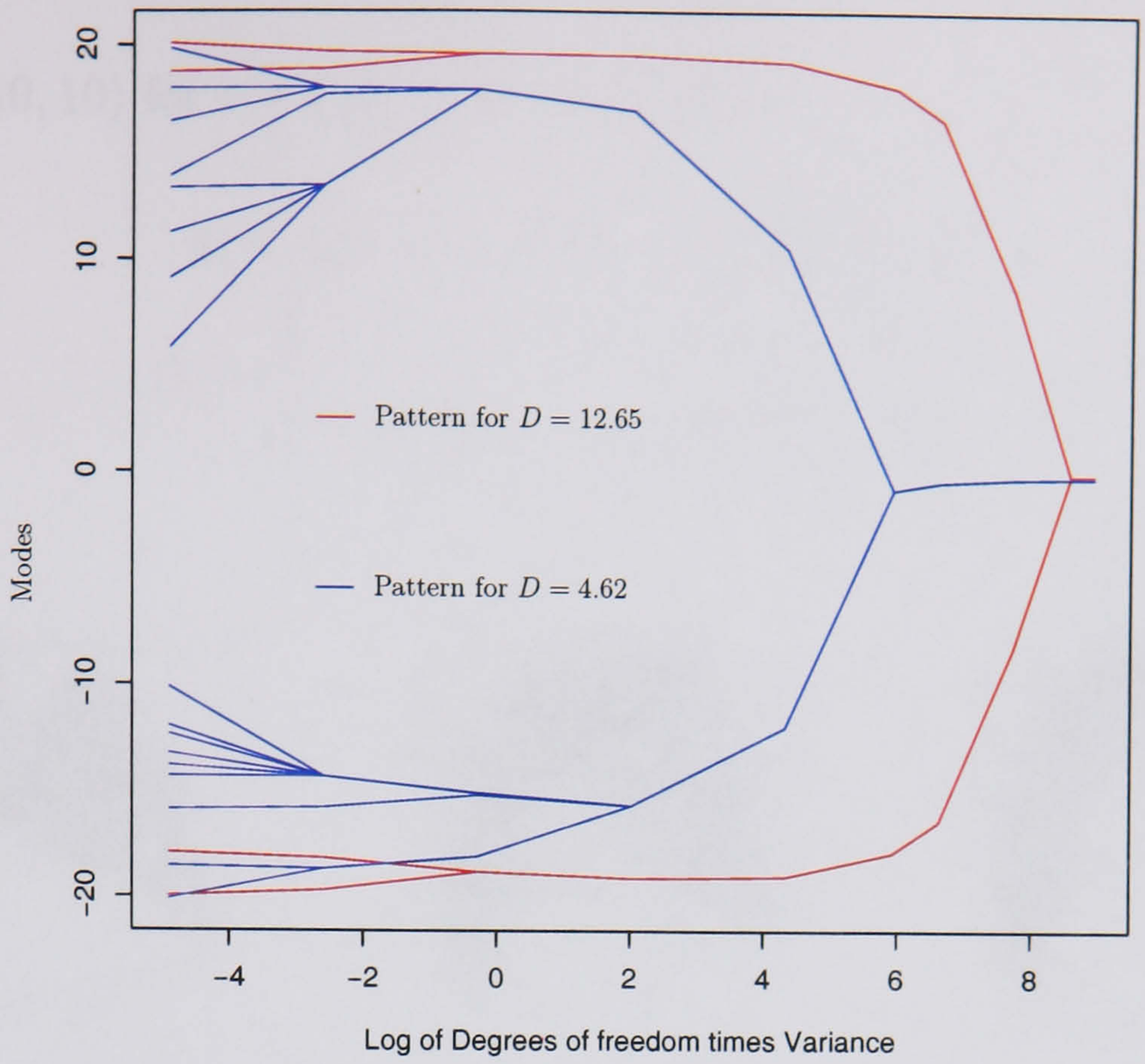


Figure 2.3: *Plots of the composite parameter, $\nu\sigma^2$ on a log-scale against local MLEs for the means are given for two different values of D. Two features are immediately detectable from the plot, i.e. multi-modality is more pronounced in the smaller D scenario than in the case of large D and, although the separation of the two modes is clear in both cases, for some values of $\nu\sigma^2$, the bimodal pattern disappears faster in the case of a smaller D than when D is large. Generally, as $D \to 0$, the lines merge towards the sample mean, causing a masking effect.*

The key result for clustering is the number of local maxima of the likelihood function as the relevant parameters are varied. The general message is that the quantities D and $\nu\sigma^2$ influence the number of the local maxima. Typically, any clustering algorithm should do well if D is large and poorly if D is small. In cluster detection, $\nu\sigma^2$, is a smoothing parameter — similar to the bandwidth in Kernel Density Estimation — an "appropriate" level of which will provide an optimal choice of the number of clusters. Having identified the clusters, cases could then be assigned to each of the detected clusters by using standard methods such as the K-means.

The X-shaped bivariate data analysed by regression

The second example involves a set of the $X-$shape data set in Figure 2.1. The data can be said to be "ambiguous" in that they do not follow a single linear regression, but instead follow a mixture model. Each of the two arms of the data set contain data points which are seemingly outlying from the perspective of the other.

To simulate the data, we generated a random variable, uniform over [-10,10], that

is, $x_i \sim U(-10, 10)$ for $i = 1, 2, ..., 49, 50$ and set

$$y_i = \beta_i x_i + \epsilon_i, \text{ where } \epsilon \sim \mathrm{N}(0, 1) \text{ and} \tag{2.14}$$

$$\beta_i = \begin{cases} +1 & \text{for} \quad i = 1, 2, \ldots, 25 \\ -1 & \text{for} \quad i = 26, 27, \ldots, 50 \end{cases} \tag{2.15}$$

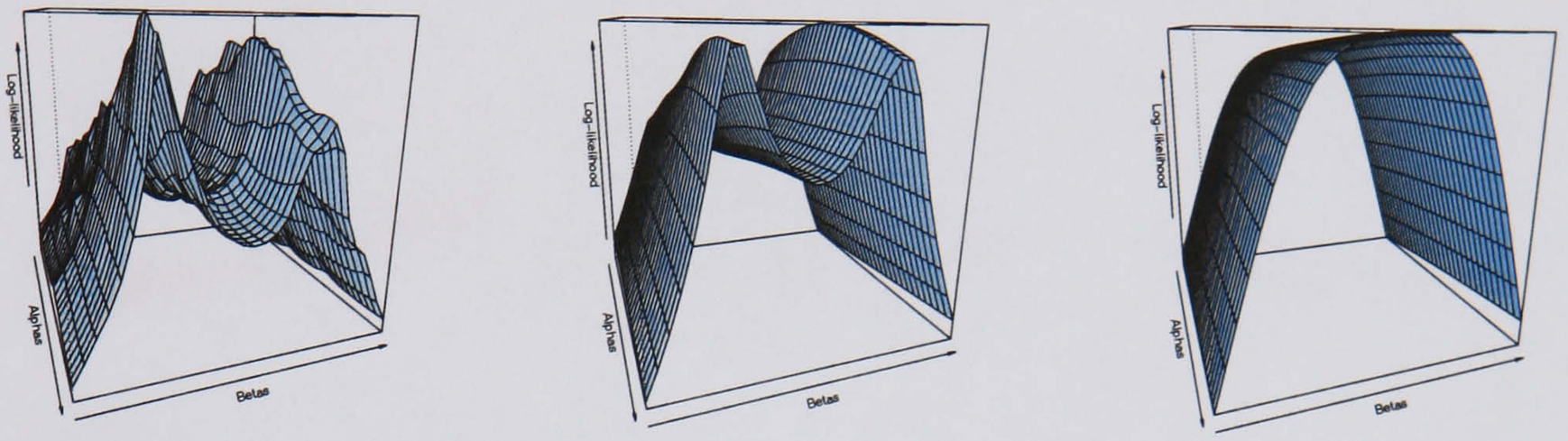


Figure 2.4: *Plots of the t log-likelihood in 3-D for the $X-$shape data, on the RHS of Figure 2.1, at different degrees of freedom. The LHS panel is based on $\nu = 0.001$ and reveals multiple local maxima. The bimodal middle panel is based on $\nu = 1$ and the clearly unimodal RHS panel corresponds to $\nu = 50$. The plots show that extremely low and high values of ν lead to swamping and masking effects respectively.*

The log-likelihood features are illustrated in 3-D plots in Figure 2.4 at three different values of ν. For smaller values of ν, the log-likelihood exhibits features of multiple local maxima, which tend to unimodality as $\nu \to \infty$. From the LHS, the plots correspond to the degrees of freedom $\nu = 0.001$, $\nu = 1$ and $\nu = 50$. Clearly, swamping and masking effects become evident as $\nu \to 0$ and $\nu \to \infty$ respectively.

The randomly-generated 50 data points form two data subsets of 25 data points each, intersecting at right angles. Standard normal noise is then added to the combined two data subsets. The two intersecting lines of data are considered to be different clusters. As in the previous example, the EM algorithm for the t-distribution is applied with different parameters to try and separate the two arms of the data.

Figure 2.5 shows the progress of the EM algorithm from the initial $\hat{\beta}_0 = 0.65504$ and $\hat{\beta}_1 = 0.03822$, for different values of ν and an unknown σ^2. The top row-panels, left-to-right correspond to the degrees of freedom $\nu = 0.001$, $\nu = 0.01$ and $\nu = 0.1$ respectively, while the bottom row-panels correspond to $\nu = 1$, $\nu = 2$ and $\nu = 4$ in the same order. Note that there is a tendency towards the OLS fit as $\nu \to \infty$.

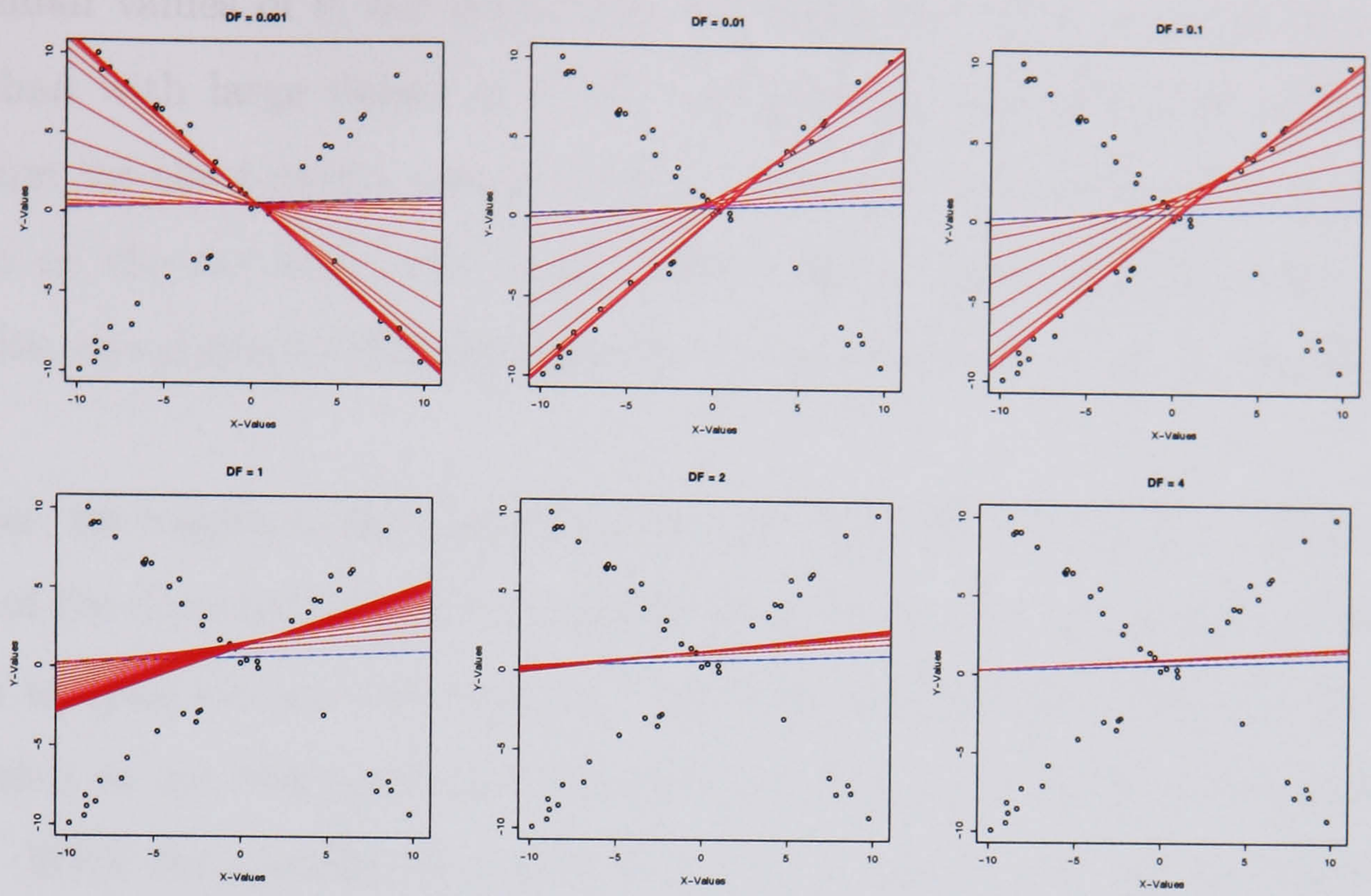


Figure 2.5: *The top three panels, left-to-right correspond to smaller degrees of freedom,* 0.001, 0.01 *and* 0.1 *respectively. At these levels of* ν*, and the starting point fixed at the OLS estimates (the horizontal-most blue line), the EM algorithm is able to detect each of the two arms depending on* ν*. In the top LH panel, the algorithm moves in a clock-wise direction to converge to the negative-slope arm, while in the remaining two, it moves in the anti-clockwise direction to converge to the other arm. The bottom panels, left-to-right correspond to larger values of* ν*,* 1, 2 *and* 4*. In this case, the algorithm considers the two arms as equally likely with only a negligible improvement to the OLS initial estimate as* $\nu \to \infty$*.*

The performance of the EM algorithm for the t-distribution depends heavily on ν. At higher degrees of freedom the bimodal nature of the data set is totally masked. For large values of ν, the starting point has no influence on the location of convergence, as the algorithm would always converge to approximately the OLS line. We need very small values of ν, (≤ 0.1), to discernibly separate the two arms of the data.

2.4.5 Concluding remarks

In both the univariate and regression examples we used the thick-tailed t-model with the EM algorithm to detect clusters. In other words, we identified the data clusters with local maxima of the likelihood function in (2.2). With σ^2 assumed known, the EM algorithm applied to the data on the LHS of Figure 2.1 yielded the clearly undesirable single and multiple clusters for large and small values of $\nu\sigma^2$ respectively. For the data on the RHS and estimated σ^2, the quantity ν was crucial.

With small values of ν, the algorithm, was more successful in separating the two arms than with large values of ν. We can generally conclude that using the EM algorithm for the t-model entails striking a balance between small ν and multiple clusters on the one hand and large ν and a single cluster on the other. In both examples, the degrees of freedom have to be small enough to show the two modes.

Consider, for instance, the univariate data on the LHS of Figure 2.1. Since the two halves of the data are roughly of equal size and shape, we will have $\hat{\mu} \approx 0$ and σ big enough to span across both clusters. Since any probability region is a function of σ, whether or not the parameter of interest is within the range will depend on the model. With the t-model we needed $\nu\sigma^2 \ll 1$ to stand any chance, implying that the method is useless in this case for picking up individual clusters.

In the regression case with unknown σ^2, for instance, convergence of the algorithm depends on ν and the starting point, which are both user-based choices. By allowing ν to vary, we transformed it into a tuning parameter; an important tool in robust regression. However, the starting point for our method can only make sense if ν is appropriately defined. Thus, when dealing with bimodal or multi-modal data the EM algorithm for the t-distribution may not be very effective with fixed degrees of freedom, for the residual variance remains a key factor in fixing ν.

A major drawback is that the method can not be described as a full algorithm because it has no well-defined way to specify the smoothing parameter $\nu\sigma^2$, which is critical. At the present there is no automatic choice of the parameter and applications on different data sets are purely exploratory. It is fine if the clusters are well-separated. In the case of appreciable overlap, there will be problems. Note, for instance, even for the no-overlap LH plot in Figure 2.1, the separation in Figure 2.2 is not perfect. The t-distribution has a big drawback here, largely because it is not thick enough in the tails. In the light of this evidence, cross-validation-based methods, such as the Kernel Density Estimation may be preferable. A brief discussion of these methods is provided in Chapter 4.

2.5 Atkinson's Forward Search algorithm

The second method discussed in this chapter is Atkinson's Forward Search Algorithm (Atkinson, 1994). Although the method's mechanics are very different from those of the scaled mixture of normals of the previous section, the overall objective of the two methods is quite similar. The forward search seeks to distinguish "good" from "bad" data. When there are two well-separated groups in the data, it rejects one of the groups in favour of the other.

The algorithm starts with a small, hopefully *clean* sample intended to be *outlier-free* and incrementally adds to the sample in such a way that outliers are unlikely to be included. It is applicable in both univariate and multiple regression settings. Although there are no very strict rules as to what sample size the algorithm should start with, the minimum initial sample size is typically $m_0 = p$, where p is the dimension of the data. At each iteration, the algorithm brings together the two subsets, one with m data points to be used in the fit and the other containing $n-m$ data points outside the fit.

The algorithm searches forward through the data by systematically incrementing the data subset used in the fit, extracting residuals from the fit and performing *outlier detection* through graphical outputs called *stalactite* plots. This process is repeated many times from different starting points. The concept of *stalactite* is borrowed from physical geography and refers to deposits of calcium carbonate hanging like icicles from the roof of a cave; a similar concept, *stalagmites* refers to deposits, like stalactites, but standing like a pillar on the floor of a cave. Atkinson (1994) uses the former concept in defining the graphical representations of outliers identified by the forward search. We adopt his terminology, although our plots run bottom up.

The algorithm is built on the philosophy of the Least Median of Squares criterion which estimates β by minimising the median of the squared residuals over i, i.e,

$$\min_{\beta} \underset{i}{\text{median}} \left|y_i - \hat{y}_i\right|^2 . \tag{2.16}$$

As the algorithm moves forward through the data, it stores the smallest value of the LMS criterion. This is the value that determines the performance of the algorithm

at each particular search. Each one of the plots obtained at the minimum value of the search yields an informative stalactite plot. The most informative stalactite plot, however, will be the one obtained at the lowest value of the criterion.

The intuition behind the algorithm lies in the contention that outlier detection methods applied to the entire data set, such as the single diagnostic methods, will be affected by the observations they are supposed to detect, especially when there are many outliers. A single diagnostic method may work well with one or two outliers. When there are many outlying observations, single diagnostic methods will result in masking and swamping — that is, with outlying observations slipping through as good data and good data being mistakenly identified as outliers. Starting with a small sample of observations and proceeding forward in an outlier-avoidance search creates a clear pattern of outliers, through the stalactite plots, for instance, whereas some of them could remain hidden from a single diagnostic method.

2.5.1 The underlying philosophy and mechanics of the method

The original model by Hadi and Simonoff (1993) and Atkinson (1994) is given as

$$Y = X\beta + \epsilon, \quad \epsilon \sim N(0, \sigma^2), \tag{2.17}$$

where Y is an $n \times 1$ vector, X is an $n \times p$ design matrix, with 1s in the first column. The formulation in (2.17) yields a univariate case when $p = 1$ and a simple regression case when $p = 2$. Further,

$$\hat{Y} = X\hat{\beta} \Rightarrow \hat{Y} = X(X^TX)^{-1}X^TY = HY. \tag{2.18}$$

The matrix $H = X(X^TX)^{-1}X^T$ is called *hat matrix* because it puts "hats" on Y. The diagonal values of the hat matrix, measure the distance of the i^{th} observation from the remaining $n - 1$ observations in the space of the hat matrix. Further,

$$\text{Var}(\hat{Y}) = \sigma^2 HIH^T = \sigma^2 H, \tag{2.19}$$

where I is an $n \times n$ identity matrix. Consequently, $\text{Var}(\hat{Y}_i) = \sigma^2 h_i$, where h_i are the diagonal values of the hat matrix, H. The variance of the residuals are given as

$$\text{Var}(Y - \hat{Y}) = (I - H)\sigma^2 I(I - H) = (I - H)\sigma^2 \rightarrow \text{Var}(Y - \hat{Y})_i = \sigma^2[1 - h_i] \tag{2.20}$$

For a two carrier model, $E(y_i) = \beta_0 + \beta_1 x_i$ the diagonal values of the *hat matrix* are

$$h_i = \frac{1}{n} + \frac{(x_i - \overline{x})^2}{\Sigma(x_i - \overline{x})^2} \tag{2.21}$$

The measure, h_i, is particularly important in that for an observation with high leverage, h_i is near 1. This means that the observed value virtually determines the fitted value. If $x_i = \overline{x}$, h_i assumes its lowest value, $1/n$. As the distance between x_i and $\overline{x}$ increases, the magnitude of h_i goes up with it. Atkinson (1985) and Atkinson and Riani (2000) provide useful details of the *hat* matrix in influence statistics.

The Forward Search algorithm starts with a small random sample of size $m_0 \in M_m$, where M_m is a subset containing m_0. As the algorithm moves forward, the elements of M_m are systematically incremented from $m = m_0$ to $m = n$. Typically, the OLS fit and the predictions are initially made based on $m_0 = p$. For a sample of size m we get the following estimates

$$\beta_m = (X_m^T X_m)^{-1} X_m^T Y_m \tag{2.22}$$

$$\hat{Y}_{(m)} = X\beta_m \tag{2.23}$$

$$e_{(m)} = [Y - \hat{Y}_{(m)}]^2, \tag{2.24}$$

where X_m is an $m \times p$ matrix, Y_m is of length m, $\hat{Y}_{(m)}$ is an $n \times 1$ vector of predicted values on the whole data set and $e_{(m)}$ is an $n \times 1$ vector, pooling together fitted and predicted residuals — that is, residuals from the m fitted and from the $n - m$ observations outside the fit. The variances of the residuals in (2.24) are proportional to

$$1 - x_i(X_m^T X_m)^{-1} x_i^T = 1 - h_i \quad \text{for } i \in M_m \tag{2.25}$$

$$1 + x_i(X_m^T X_m)^{-1} x_i^T = 1 + d_i \quad \text{for } i \notin M_m \tag{2.26}$$

The following are the main ingredients of Atkinson's Forward Search algorithm

1. The number of observations, m_0, with which to initialise the search.

2. The residuals in (2.24), m of which are OLS residuals and $n-m$ are predicted.

3. The dynamic quantities (2.25) and (2.26) for the fitted and predicted values.

4. Sample distributions of the fitted and predicted normalised-adjusted residuals.

5. To detect outliers, we first run multiple simulations for calibration of residuals. This is done by running a similar algorithm on an artificial dataset with the dependent variable simulated from a standard normal distribution and the same design matrix. Each simulation calculates the estimated scale parameter

$$\tilde{\sigma}_m^2 = e^2_{[\text{median}]}(\tilde{\beta}_m), \tag{2.27}$$

where $e^2_{[median]}$ is the median of the raw residuals. The algorithm yields $\overline{\sigma}_m$ as (2.27) averaged over the starting points, for each m, which provides an adjustment for some unusual estimates of sigma, thus normalising the stalactites.

The basic steps of Atkinson's Forward Search algorithm

The Atkinson's Forward Search algorithm can be summarised by the following steps.

1. Randomly pick m_0 data points.

2. Fit the linear regression model for the picked data points.

3. Extract the raw residuals (2.24) for the fitted and predicted data points.

4. Arrange all raw residuals in ascending order to obtain (2.27).

5. Adjust fitted and predicted residuals to get "normalised-adjusted" residuals

$$t_i = \frac{e_i \overline{\sigma}_m}{\tilde{\sigma}_m \sqrt{(1-h_i)}} \quad i \in M_m \tag{2.28}$$

$$t_i = \frac{e_i \overline{\sigma}_m}{\tilde{\sigma}_m \sqrt{(1+d_i)}} \quad i \notin M_m \tag{2.29}$$

   Both (2.28) and (2.29) are obtained through dividing the raw residuals by the square root of (2.25) and (2.26) respectively as well as multiplying them by $\overline{\sigma}_m$. This adjustment is necessary so as to eliminate the sample dependence.

6. Pick the $m+1$ observations corresponding to the smallest $m+1$ "normalised-adjusted" residuals in the foregoing order.

7. Update $m \leftarrow m+1$.

8. Repeat steps 2 through 7 as long as $m \leq n$.

9. Record the minimum value for the scale parameter, $\tilde{\sigma}_{\text{min}}$, over m. This value defines the performance of this particular search.

10. Repeat the algorithm, say, 120 times — the larger this number is the better, as it helps explore a large number of starting points.

11. Pick the best performance, based on the smallest estimated scale parameter, $\tilde{\sigma}_{\text{min}}$, and plot the corresponding stalactite plot.

Because the algorithm yields as many stalactite plots as there are simulations, outlying observations are visually identified by looking at the "best" plot. This is the plot with the minimal scale parameter, $\tilde{\sigma}_{\text{min}}$ and, like other stalactite plots, it represents an $n \times n$ graphical presentation of the data with the data labels on the horizontal axis and the sequentially fitted samples, m, on the vertical axis. Each case for which $t_i \geq 3$ is identified as outlying and marked by an asterisk on the plot.

Cluster detection based on the Forward Search Algorithm will rely only on the resulting visual pattern. Basically, assessment will be based on the concentration of asterisks (i.e., observation for which $t_i \geq 3$). One of the possible problems is that the stalactite pattern may be quite indiscernible if the dataset is very small.

2.5.2 Application of the algorithm

In this section we apply the Atkinson's Forward Search algorithm to the univariate and bivariate data in Figure 2.1. Again, we seek to separate the two modes of the data from each other. For easy visibility and interpretability of the stalactite plots, we use a smaller sample, of size 100, for the univariate data but we retain all the parameters. The same bivariate dataset used in Section 2.4.4 is used here.

The smallest initial values for running the algorithm are $m_0 = 1$ and $m_0 = 2$ for the univariate and bivariate data respectively. To detect outliers we try all possible starting values for the univariate data and run the algorithm from random starting points for the bivariate data. We then use stalactite plots to graphically capture the observations identified as outlying. The final interpretation of the forward search is made by visually looking at the pattern yielded by the stalactite plot.

The univariate bimodal data

The univariate case is an instance of the general regression formulation in Section 2.5.1, when $p = 1$. There are 100 observations in all, 57 in the first cluster and 43 in the other. With X_m now given as a vector of 1s, the precision matrix $(X_m^T X_m)^{-1}$ contains only the constant $1/m$. We exhaustively ran 100 simulations for the calibration of residuals — results are given in Table 2.1, with an average value of 0.847.

1	0.995	0.991	0.986	0.981	0.975	0.970	0.961	0.952	0.942	0.932
11	0.922	0.914	0.904	0.890	0.878	0.864	0.856	0.850	0.844	0.837
21	0.827	0.814	0.803	0.793	0.783	0.773	0.766	0.760	0.754	0.750
31	0.741	0.735	0.731	0.722	0.716	0.714	0.714	0.713	0.711	0.706
41	0.703	0.701	0.697	0.693	0.687	0.687	0.688	0.686	0.680	0.677
51	0.671	0.666	0.666	0.658	0.653	0.654	0.658	0.660	0.661	0.665
61	0.665	0.665	0.664	0.663	0.666	0.668	0.669	0.670	0.673	0.678
71	0.685	0.684	0.684	0.686	0.689	0.696	0.699	0.697	0.697	0.696
81	0.695	0.691	0.693	0.690	0.683	0.684	0.676	0.665	0.660	0.6566
91	0.654	0.661	0.659	0.663	0.670	0.663	0.668	0.666	0.671	0.667

Table 2.1: *Simulated $\overline{\sigma}_m$ values for each sample size m, in a dataset of size 100. The values are median-based estimates of the scale parameter, obtained from ordered residuals of an artificial dataset simulated from a standard normal distribution, averaged over 25 simulations for each of the values of m. They provide adjustment in the criteria (2.28) and (2.29) for some unusual estimates of the variance, thus normalising the stalactite plots by removing dependence on the sample size.*

Results of the application of the method on univariate data are graphically summarised in Figure 2.6. The LHS and the middle panels represent the univariate data-points and the best solution for the forward search, obtained at $\tilde{\sigma}_{\text{min}} = 3.844$, with the algorithm initialised with observation 53 from the larger of the two groups. This is a plausible solution in relation to the way the data were simulated, as it is a good approximation of the cluster's $\sigma = 3.162$. The RHS panel shows a typical example of initialising the algorithm with a within-group extreme value, in this case observation 44. The smallest scale estimate corresponding to this starting point is $\tilde{\sigma}_{\text{min}} = 5.376$ which also applies to the starting points 24, 29, 38, 39 and 46.

Note that all the foregoing starting points yield large estimates of the scale parameter, as a result of which, initially, no observations are picked up as outlying. That at lower values of m nothing is flagged as outlying is due to the fact that at this stage

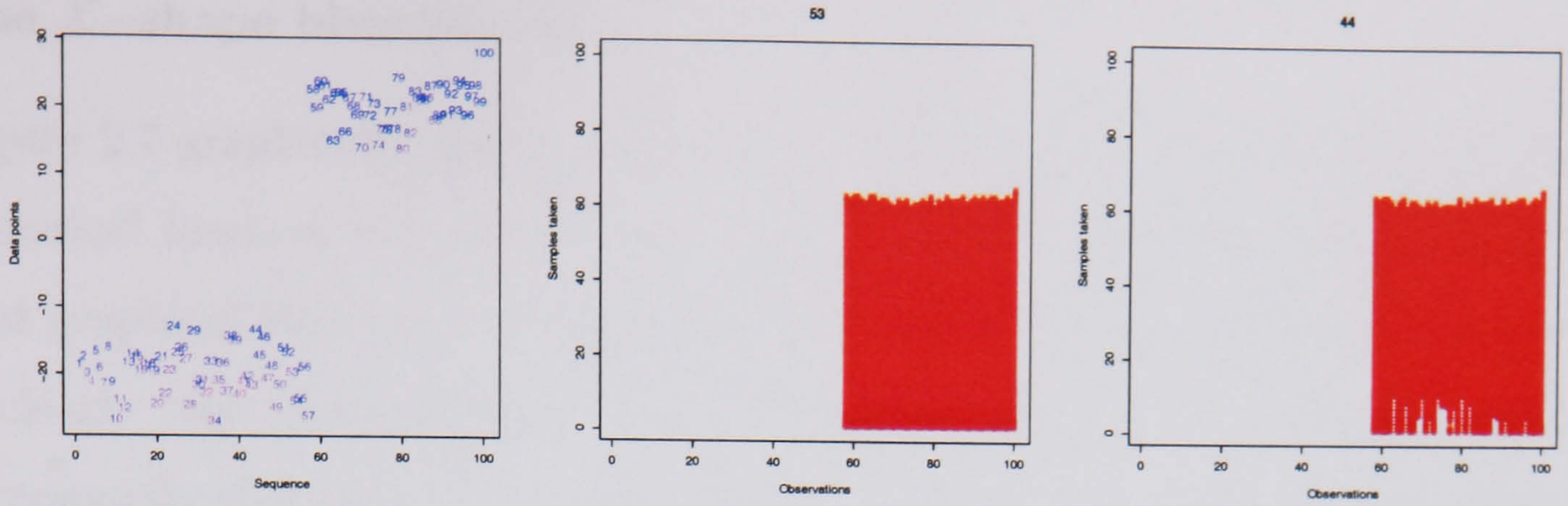


Figure 2.6: *The LH and middle panels show the univariate data-points and the algorithm's best clustering solution respectively. The best solution was obtained at* $\tilde{\sigma}_{min} = 3.844$ *running initially from observation 53. The RHS panel shows the pattern for an initial point in observation 44. This starting point, alongside those from observations 24, 29, 38, 39 and 46, yield a relatively large scale parameter, as a result of which at lower values of* m*, some "bad" observations slip through as "good". The observations have been ordered in the way they were simulated, i.e. with one group all to the left and the other to the right. In all three panels, the horizontal axis represents data labels, the vertical axis, i.e. each row, in the LHS panel, represents data values. In both the middle and RHS panels, the vertical axis corresponds to a value of* m*. Each observation, flagged as outlying, is marked by an asterisk.*

the algorithm treats the whole data as a homogeneous group and $\tilde{\sigma}_m$ is estimated as a total variance rather than group variance. The algorithm's average scale estimate based on the larger group is 4.686, which goes up to 10.328 as the smaller group joins in. The scale estimate is a crucial parameter in detecting the clusters, as small and large values may lead to swamping and masking effects respectively. At small values of the scale estimate, the algorithm is able to distinguish between the two clusters and as $m \to n$, everything is "good".

Masking and, to some extent, swamping effects not withstanding, the overall performance of the algorithm in the univariate case appears to be good. The algorithm quite successfully separates the two modes from each other. Note that for such linearly separable groups, the median-based choice of the best solution inevitably favours the larger group. In the unlikely event of perfectly equal groups, the resulting conventional median will be the average of the lower and the upper median, and for well-separated groups, this measure will be too large for the lower median and too small for the upper median, as a result of which none of the groups may be detected as outlying. A simple solution would be to adopt the lower median. The lower median is defined here to mean the lower of the two group medians.

The $X-$shape bivariate data

Figure 2.7 graphically illustrates the forward search. The LHS panel gives the geometrical location of the simulated $X-$shape data. The RH panel provides the best graphical clustering solution from the forward search algorithm. The method is clearly very successful here in separating the two arms of the data. Each row corresponds to a value of m. The observations have been ordered in the way they were simulated, i.e. with one group all to the left and the other to the right.

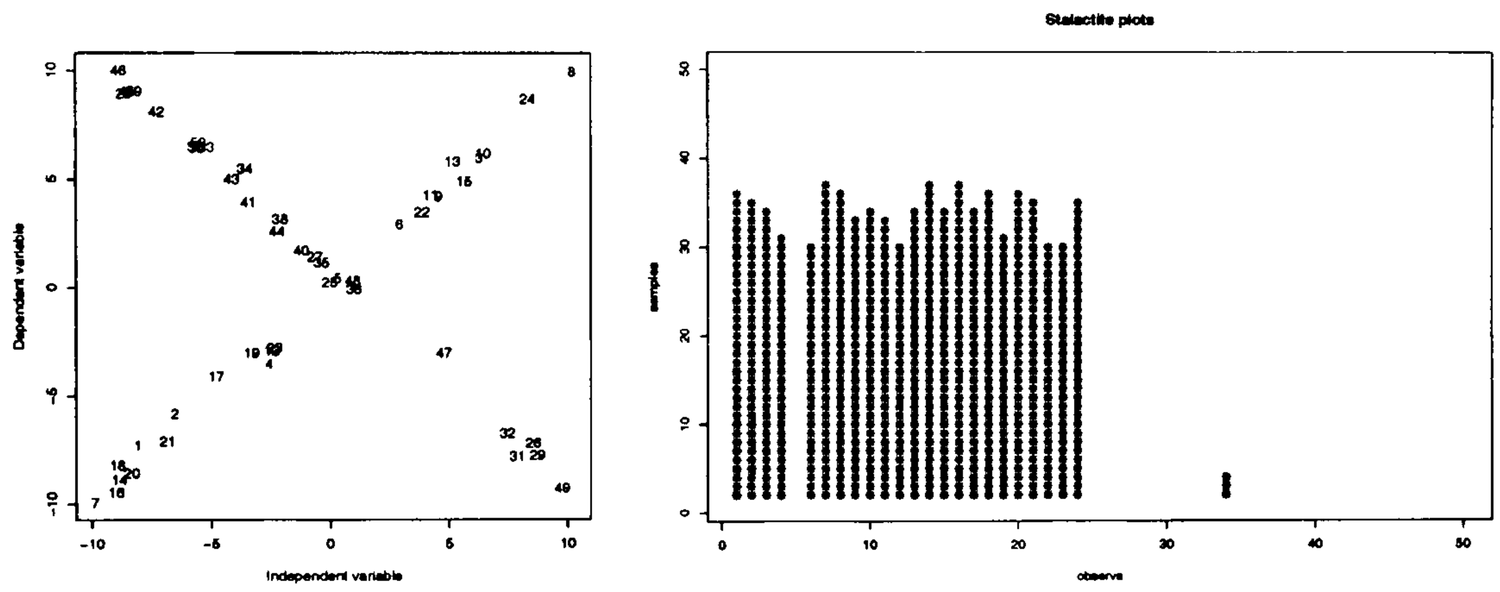


Figure 2.7: *The RHS panels shows the best stalactite plot for the $X-$shape data, on the LHS panel. The algorithm's best solution was obtained at the 115^{th} search with $\tilde{\sigma}_{min} = 0.281$. The forward search is clearly very successful here in separating the two arms of the data. Each row corresponds to a value of m. The observations have been ordered in the way they were simulated, i.e. with one group all to the left and the other to the right. The points marked by an asterisk denote outlying observations.*

Note that observation 5, in particular, despite lying in the largely outlying cluster, is not identified as such because of its ambiguous location. Further, large values of m lead to masking effects. Our best solution presented in Figure 2.7 comes from the 115^{th} search with the performance of $\tilde{\sigma}_{\text{min}} = 0.281$. Closest to this performance are searches number 17, 24, 61, 74, and 108 with corresponding $\tilde{\sigma}^{\text{s}}_{\text{min}}$ 0.329, 0.480, 0.330, 0.479 and 0.472 respectively. Each of the searches unambiguously picked up either one of the two arms of the data, with less unambiguous solutions yielded at much larger values of $\tilde{\sigma}_{\text{min}}$.

Values of m and the impact of the initial fits

Depending on the slope of the initial fit, the Forward Search may either unambiguously pick up one cluster and reject the other, or lead to an ambiguous solution. For

instance, starting with observations from the same data arm, regardless of the initial value of m, has an almost straightforward effect of identifying the observations on the other arm as outliers. Which observations will ultimately be confirmed as outlying will depend on the geometrical locations of the remaining 48 observations. Apparently this affects the outcome of this particular search, although repeated searches can prove to be a remedy.

Particularly for the $X-$shape data, starting at $m_0 = 2$, the stalactite plots clearly show the outlying nature of either of the two arms. As the sample size increases, so does the effect of masking. For instance, one would be very lucky to identify any outliers if the algorithm is started at $m_0 = 25$. In fact, not only starting at large values of m, but even when the algorithm is started with small values of m masking becomes increasingly pronounced as m grows to about size 35. Why does this happen? The behaviour derives from the internal mechanics of the algorithm. The algorithm starts with a set of data intended to be clean and outlier-free and we can not therefore expect any of the observations in the fit to be flagged as outlying. As the forward search progresses through the data, incrementing the sample size as it moves, bad observations, if any, are inevitably brought into the fit. Beyond a particular large m, once observations are brought into the fit, they can't be flagged as outlying, which explains the lack of outliers at values of m larger than 35.

It is also possible for some observations to be flagged as outlying at small values of m. This might sound fallacious, especially in cases when the fitted observations are from the same arm of the data set. This situation is explained by the location of the data points used in the fit. Two data points, for instance, can form a fitted line in the direction forming, say 45^o with the arm to which they belong. This means observations on the arm can then easily be flagged as outlying, although this can not go on for long as the next *good* observations will apparently come from the arm to which the initial points belong. This explains why at small values of m some observations are identified as outliers but cease to be so as m grows.

In fact, by the definition of the algorithm there aren't long enough patterns at smaller values of m to record a swamping effect. For instance, if a line is fitted

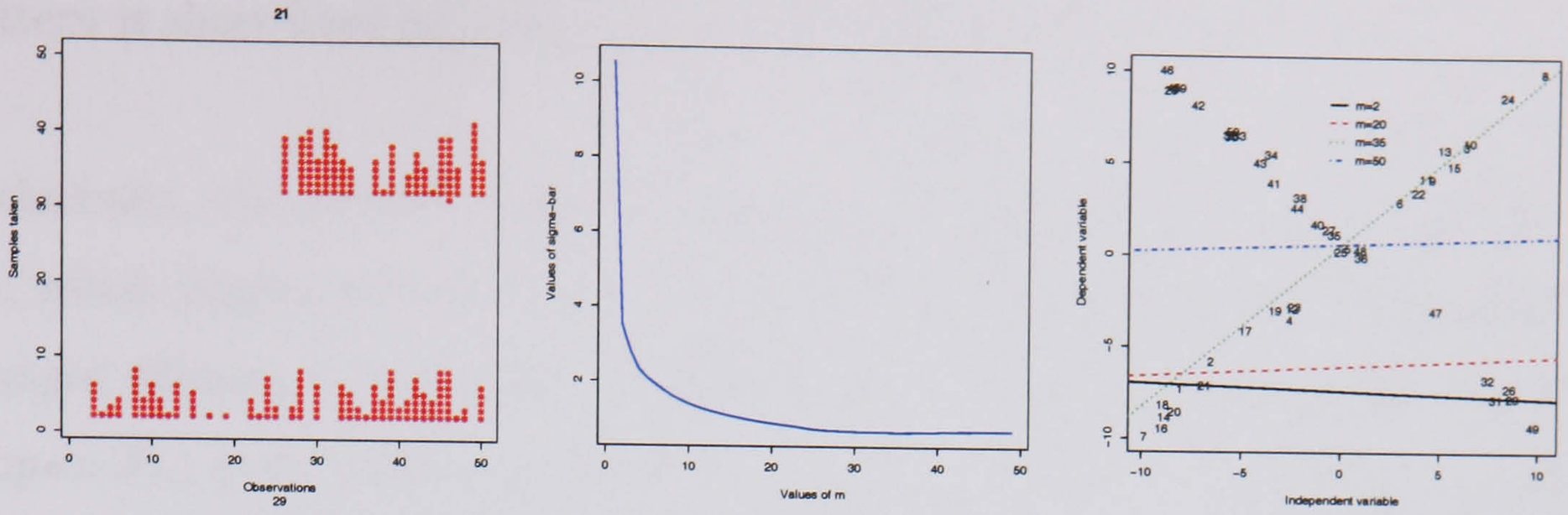


Figure 2.8: *A typical example of an ambiguous solution, on the LHS, resulting from a "bad" starting point, thus leading to gross swamping and masking effects. The RHS panel provides a graphical description of what happens on the LHS. The algorithm is initialised with observations 21 and 29, yielding an almost zero slope fit. The initial line means that at both small and large values of m, fitted observations will come from either arms of the data, hence the flagging of outliers on both arms in the early stages of the algorithm, as exhibited on the LHS panel. At intermediate values of m, "nothing is outlying" but outlyingness re-appears at large values of m before disappearing as $m \to n$. The dramatic change of $\overline{\sigma}_m$ as m increases, shown in the middle panel, provides a technical explanation of the behaviour of the algorithm on the LHS. Indeed, the behaviour justifies the need for choosing the best solution from a number of stalactite plots, in order to avoid ambiguous solutions.*

initially to observations 29 and 31 (See LHS of Figure 2.7), most observations from the same arm will be flagged as outliers as would most from the other arm. However, this behaviour soon dies away with increasing m.

The LHS and RHS panels in Figure 2.8 graphically demonstrate the impact of the initial points selected for the fit. This scenario, with $m_0 = 2$, may be considered as one of the "worst cases", as the algorithm starts with the points 21 and 29 picked from either of the two arms. The panels provide a graphical description of how the algorithm proceeds from its initial choice of data points. The initial, almost horizontal, slope means somehow both at small and large values of m the algorithm will draw into M_m observations from both arms, thus leading to swamping and masking.

At small values of m the algorithm sequentially draws into M_m observations from either half of the two arms, thus flagging as outlying the remaining observations from the other halves. This proceeds to somewhere about the OLS line, where nothing is flagged as outlying, as exhibited by the empty space at about $10 < m < 30$. At $m = 35$, for instance, one of the arms is unambiguously flagged as outlying, but the

pattern is short-lived as masking effects fast creep in as $m \longrightarrow n$.

Technically, the behaviour of the algorithm in Figure 2.8 is due to the fact that $\overline{\sigma}_m$, which plays a crucial role in the outlier-detection criteria in (2.28) and (2.29), changes dramatically with m. The middle panel in Figure 2.8 exhibits the drastic drop in $\overline{\sigma}_m$, from a value larger than 10 at $\overline{\sigma}_{m=2}$ to below 1.5 at $\overline{\sigma}_{m=10}$. Patterns such as this underline the need for choosing the best solution from multiple stalactite plots, hence avoiding such ambiguous solutions.

2.5.3 Concluding remarks

There is a wide range of choices of clustering algorithms in the literature and many researchers are faced with the hard task of having to decide on which one to use. Generally, clustering methods are prone to two major problems, i.e., the choice of the number of clusters and the difficulty in interpreting them. Most clustering algorithms prefer certain cluster shapes, and the algorithms will always assign the data to clusters of such shapes even if there were no clusters in the data. Indeed, if we seek not only to reduce data dimensionality but also to make inferences about the structure of the data, it is important to analyse whether the data set exhibits a clustering tendency. This part of the thesis considered aspects of the nature of the data as well as algorithmic initialisation relating to specific data structure.

Built on the philosophy of the Least Median of Squares mechanics, Atkinson's Forward Search algorithm falls in the same domain of LMS application as the *elemental* and *exhaustive* searches. In fact, the Forward Search is a version of elemental search. The exhaustive search would, typically, deliver reliable results, but, although it may sound plausible its successful application remains a function of the size of the data set. Our $X-$shape data set of 50 observations would require 1225 combinations for only $m = 2!$ As m increases, so does the total number of combinations. By searching forward from different random points, the Forward Search provides a viable alternative. The nature of its construction implies that the algorithm will not detect outliers in "well-behaving" data.

For well-separated, two-cluster, examples as used in this chapter, every search, i.e. each solution yielded by the stalactite plots, can be placed into one of the following three categories, namely, accepting cluster one, accepting cluster two or yielding an ambiguous solution. Typically, in both the univariate and bivariate cases discussed, starting with an m choice from only one of the two clusters, the algorithm flags all observations from the other cluster as outlying. It then keeps on incrementing the sample size, m, until the observations in the current cluster are exhausted and as the algorithm starts drawing cases from the other cluster into the subset m, masking effects start to emerge. Indeed, both Figure 2.6 and Figure 2.7 exhibit that at low values of $\tilde{\sigma}_m$, the algorithm treats one of the clusters as good data and the other as bad, but as $m \rightarrow n$, $\tilde{\sigma}_m$ gets larger and all the data are treated as good.

Particularly for the bivariate case, if the algorithm initially picks observations from both arms of the data, filtering-out "bad" observations is harder than if the observations were from the same arm. Typically, masking and swamping will affect the results although it will also depend on the particular initial points. The presence of masking and swamping in both examples suggest that more structures could still be uncovered in the data by removing the outlying bit from the scene and re-applying the algorithm to the remaining data.

Although both the univariate and bivariate examples presented in this chapter may look weird, Atkinson's Forward Search algorithm will usually work pretty well in practice, with unordered data. Typically, large n and clumps of outlying observations will combine to create clearly discernible patterns. Generally, individual observations standing out as outliers will be earmarked and visually judged by the continuity and length of the asterisk marks.

The main drawback of the method is that there is no formal method to transform the stalactite plots into clusters. Plots can only be judged visually by looking for a large band in the vertical direction where nothing changes. This approach may work well for large data sets involving multiple clusters. Indeed, for our well-clustered and separated datasets, visual-based judgement appeared to be fine. However, it is very likely that different datasets may yield unclear patterns that may be hard to inter-

pret. A small sample with a single outlying observation, will not yield an elongated pattern of asterisks. It is therefore important to consider the sample size in interpreting the emerging patterns returned by the algorithm.

In comparison with the long-tailed method of Section 2.4, for which we needed an exploratory analysis of the crucial parameter ν, the Forward Search algorithm appears to be more successful in separating the two clusters of the data. However, in the long-tailed method we have straightforward ways of fitting the data, i.e. normal, multivariate, OLS, as distinct from the almost heuristic nature of Atkinson's Forward Search algorithm. Although common sense may dictate that a *formal method* would perform better than a *heuristic approach,* Atkinson's forward search, heuristic though it may be, does appear to be an intelligent method in detecting outlier patterns, especially in cases that involve many outliers.

Standard clustering approaches would not usually work well for our example because of the nature of the boundary between the groups. Any method based on a single rule will need to be highly problem-specific to do well. For example, the K-means algorithm will typically converge to a single global mean for the two groups, thus failing to separate them. As part of our envisioned future work, we shall be comparing the performance of these methods with standard clustering methods on different forms of clusters such as the lunar-star and embedded circles cluster forms.

Chapter 3

Finite mixture of normals for clustering

3.1 Introduction

In this chapter we consider the application of mixture models for clustering via the EM algorithm. In Chapter 2 we adopted a sequential approach in separating the two clusters from each other. This chapter adopts a simultaneous clustering approach, based on fitting finite mixture models using the EM algorithm. While the two methods in Chapter 2 require no *a priori* knowledge of the number of clusters in the data, clustering by mixture models typically stipulates that the number of clusters in the data be known in advance.

Studying patterns in data using mixture models is crucial in data-based decision making. For instance, in many practical applications we produce measurements from experimental units that are known to belong to one of a set of classes, but whose individual class memberships are unknown. For instance, in medicine, we may have clinical measurements for a set of patients whose disease classifications are unknown. An early step towards the disease classification would be to place patients with similar clinical measurements into the same group. Such situations have made the application of mixture models very popular.

Consider a situation where the data take the form of a random sample of obser-

vations where each observation is described by a parametric finite mixture density. Direct maximisation of the likelihood function is generally very awful. However, if we treat the group membership as missing data, the EM algorithm can be used to estimate the MLEs. We address this problem from location-scatter and regression perspectives. Detailed discussions of finite mixture models are provided in McLachlan and Peel (2000) and Titterington *et al.* (1985).

This chapter is organised as follows. Section 3.2 introduces the general methodology, basic considerations, some notation and the general mechanics of the EM algorithm. The section also provides a brief discussion on choosing the initial values for the algorithm for the mixtures. A univariate application under various scenarios appears in Section 3.3, followed by the regression application in Section 3.4. Concluding remarks are given in Section 3.5.

3.2 The EM algorithm for mixtures

Consider the problem where the number of clusters is known *a priori*. An entity of interest can belong to only one of the known clusters and our interest lies in the cluster membership of the entity and the parameters of the model used to determine its membership. The cluster membership of an entity is unknown, and will have to be estimated based on available data and the chosen model. We start by introducing some basic notation and conventions.

3.2.1 Basic notation and considerations

We have n entities, each lying in one of the known classes $k = 1, ..., K$. Let $C_i \in \{1, ..., K\}$ label the class of the i^{th} entity. Class membership can also be coded by a dummy vector $z_i \in \{0, 1\}^K = (z_{i1}, ..., z_{iK})$, where $z_{ik} = 1$ if and only if $C_i = k$. Given the class membership, the observations have some probability density function $\phi_k(x)$. The vector x_i contains the measurement of the feature of the entity $i \in \{1, ..., n\}$. The class memberships are unobserved and the densities $\phi_k(.)$ contain some unknown parameters, θ_k. We assume that $\phi_k(x)$ corresponds to the normal density with k components and $\theta_k = \{\mu_k, \sigma_k^2\}$. If the classes have prior probabilities π_k, then the

marginal density of x is a mixture of normals defined as

$$\phi(x;\psi) = \sum_{k=1}^{K} \frac{\pi_k}{\sqrt{2\pi\sigma_k^2}} \exp\Big\{ - \frac{(x-\mu_k)^2}{2\sigma_k^2} \Big\}, \tag{3.1}$$

where $\psi = \{\theta_k, \pi_k,\ k = 1, ..., K\}$, where the class priors, π_k, are non-negative and $\sum_{k=1}^{K} \pi_k = 1$. The expression in (3.1) denotes a finite mixture model density with K normal components. The complete data set is therefore given as $y_i = (x_i, z_i)$, that is, the observed x_i augmented by the *missing data*, z_i. Given the data $x_1, ..., x_n$, the objective is to estimate ψ.

If the class membership were also known, the problem would be trivial — just take the sample mean and variance within each group of the observations. But since the class labels are unknown, we shall use the EM algorithm to iteratively estimate the parameters. One of the most notable properties of the EM algorithm is that it depends critically on the initial values of the parameters. We address this issue in sections 3.2.2, 3.3 and 3.4.

The categorical variable and the probability of class membership

With the categorical variable $z_i = (z_{i1}, ..., z_{iK})$ denoting class membership of the entity, i, our interest then is focused on the relationship between the class membership z_i and the feature vector x_i. Note that the indicator variable, $\{z_{ik}\}$, is an $(n \times K)$ matrix of binary values. The k^{th} component of $\{z_{ik}\}$ is 1 if entity $C_i = k$ and zero otherwise. That is,

$$z_{ik} = \begin{cases} 1 & \text{if entity } C_i = k \\ 0 & \text{if entity } C_i \neq k \end{cases} \tag{3.2}$$

The posterior membership probability that an observation x_i belongs to class k is

$$P\{C_i = k|x_i\} = P\{z_{ik} = 1|x_i\} = \frac{\pi_k \phi_k(x_i)}{\sum_{k=1}^{K} \pi_k \phi_k(x_i)} = \hat{\pi}_{ik},\ \text{say}, \tag{3.3}$$

and as such, $\hat{\pi}_{ik}$ is a function of x_i. The foregoing expression can be given a Bayesian description in that it represents the probability of our entity of interest belonging to class k, given that x_i was observed. The numerator represents the model component in which π_k arose from the i^{th} component of the mixture depending on the value initially returned by z_{ik}. The denominator represents the marginal density of x_i.

The complete data case

What we do in a *complete-data* setting determines how we proceed in an *incomplete-data* setting. If z_{ik} were observable, we would estimate the parameters by maximum likelihood as follows.

$$\hat{\pi}_k = \frac{\sum_{i=1}^{n} z_{ik}}{n}, \; k = 1, ..., K. \tag{3.4}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{n} z_{ik} x_i}{\sum_{i=1}^{n} z_{ik}}, \; k = 1, ..., K. \tag{3.5}$$

$$\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{n} z_{ik} (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{n} z_{ik}}, \; k = 1, ..., K. \tag{3.6}$$

It is also possible to assume a common variance in (3.6), in which case the population variance can be estimated by the pooled variance, based on the K samples.

3.2.2 Estimation of the MLEs

In the subsequent sections we apply the EM algorithm in estimating model parameters for both the univariate and regression models. In both cases, the problem is to fit the mixture model $\phi(x; \psi)$ and since we do not observe z_i, we must replace them by their posterior expectations given the data, x_i. The result is virtually (3.3), which effectively represents the **E**-step of the algorithm. The computations at the **M**-step of the algorithm are equivalent to those of the MLEs above, with the unobservable z_{ik} replaced by $\hat{\pi}_{ik}$ from (3.3).

The EM algorithm is initialised with chosen $\mu_k^{(m)}$ and $\sigma_k^{2(m)}$, where $m = 0$. At the **E**-step of the algorithm, we replace each z_i by its expectation conditional on x_i and maximise the parameters at the **M**-step. More specifically, the EM algorithm proceeds as follows, where m denotes the iteration.

1. The **E**-step.

$$\hat{\pi}_{ik}^{(m+1)} = \mathrm{E}[z_{ik} | x_i, \hat{\mu}_k^{(m)}, \sigma_k^{2(m)}, \hat{\pi}_k^{(m)}] = \frac{\phi_k[x_i; \hat{\mu}_k^{(m)}, \hat{\sigma}_k^{2(m)}, \hat{\pi}_k^{(m)}]}{\phi[x_i, \hat{\psi}^{(m)}]}, \tag{3.7}$$

where $\phi_k(x; \mu_k, \sigma_k^2)$ denotes the density for the k^{th} component and the denominator is a mixture density as in (3.1). The **M**-step maximises the parameters.

2. The M-step.

$$\hat{\mu_k}^{(m+1)} = \frac{\sum_{i=1}^{n} \hat{\pi}_{ik}^{(m+1)} x_i}{\sum_{i=1}^{n} \hat{\pi}_{ik}^{(m+1)}}. \tag{3.8}$$

Depending on the assumption made, the variance can be estimated as either

$$\hat{\sigma}^{2(m+1)} = \frac{\sum_{i=1}^{n} \hat{\pi}_{i1}^{(m+1)} [x_i - \hat{\mu}_1^{(m+1)}]^2 + \sum_{i=1}^{n} [1 - \hat{\pi}_{i1}^{(m+1)}][x_i - \hat{\mu}_2^{(m+1)}]^2}{n} \tag{3.9}$$

for $K = 2$ or more generally

$$\hat{\sigma}_k^{2(m+1)} = \frac{\sum_{i=1}^{n} \hat{\pi}_{ik}^{(m+1)} [x_i - \hat{\mu}_k^{(m+1)}]^2}{\sum_{i=1}^{n} \pi_{ik}^{m+1}}. \tag{3.10}$$

Increment m and iterate the **E** and **M** steps until convergence.

Choosing the initial parameters for the EM algorithm

A crucial factor of the EM algorithm is the choice of the initial parameters. For instance, in a univariate setting, as to be expected, choosing $\mu_1^{(0)} \approx \mu_2^{(0)}$ might result into the two parameters converging to the same value, thus leading to masking effects. More generally, given equal priors and variance, for $K = 2$, the Mahalanobis distance, $D = |\mu_1 - \mu_2|/\sigma$, measures the separation of the two groups. For well-separated groups, with a large D, initialising the μ_ks within each group will typically yield nice and unambiguous converging patterns.

In typical applications of finite mixture of normals, different initial values for the EM algorithm may lead to different final estimates. One conventional approach in choosing the initial parameters is to try different random starting points. A close alternative would be to randomly generate *iid* initial μ_ks with mean and variance equal to those of the data. Another issue of great concern is the assumption made about the group variance. For instance, if, under the assumption of $\sigma_1^2 \neq \sigma_2^2$, the initial $\mu_1^{(0)}$ is set to equal x_1, and the initial σ_1^2 is large, then the likelihood would tend to infinity (Kiefer and Wolfowitz, 1956). We shall roughly be guided by the foregoing issues in considering the choice of the initial parameters.

3.3 Application under the univariate setting

In this section the EM algorithm is applied to iteratively estimate the parameters of a mixture model, with $K = 2$ normal components, for the univariate case. A

univariate dataset, x_i, of size $n = 1981$ (size resulting from the simulation interval), dimension $p = 1$, equal class priors $\pi_1 = \pi_2$, equal group variance $\sigma_1^2 = \sigma_2^2 = \sigma^2$ and unobservable class labels, z_i, is generated from the mixture of normals given in (3.1). Our objective is to estimate the parameters of the density (3.1), i.e. $\psi = (\mu_1, \mu_2, \sigma^2, \pi_1, \pi_2) = (35, 65, 100, 1/2, 1/2)$, using the simulated data $\{x_i\}$ and the EM algorithm for mixture models.

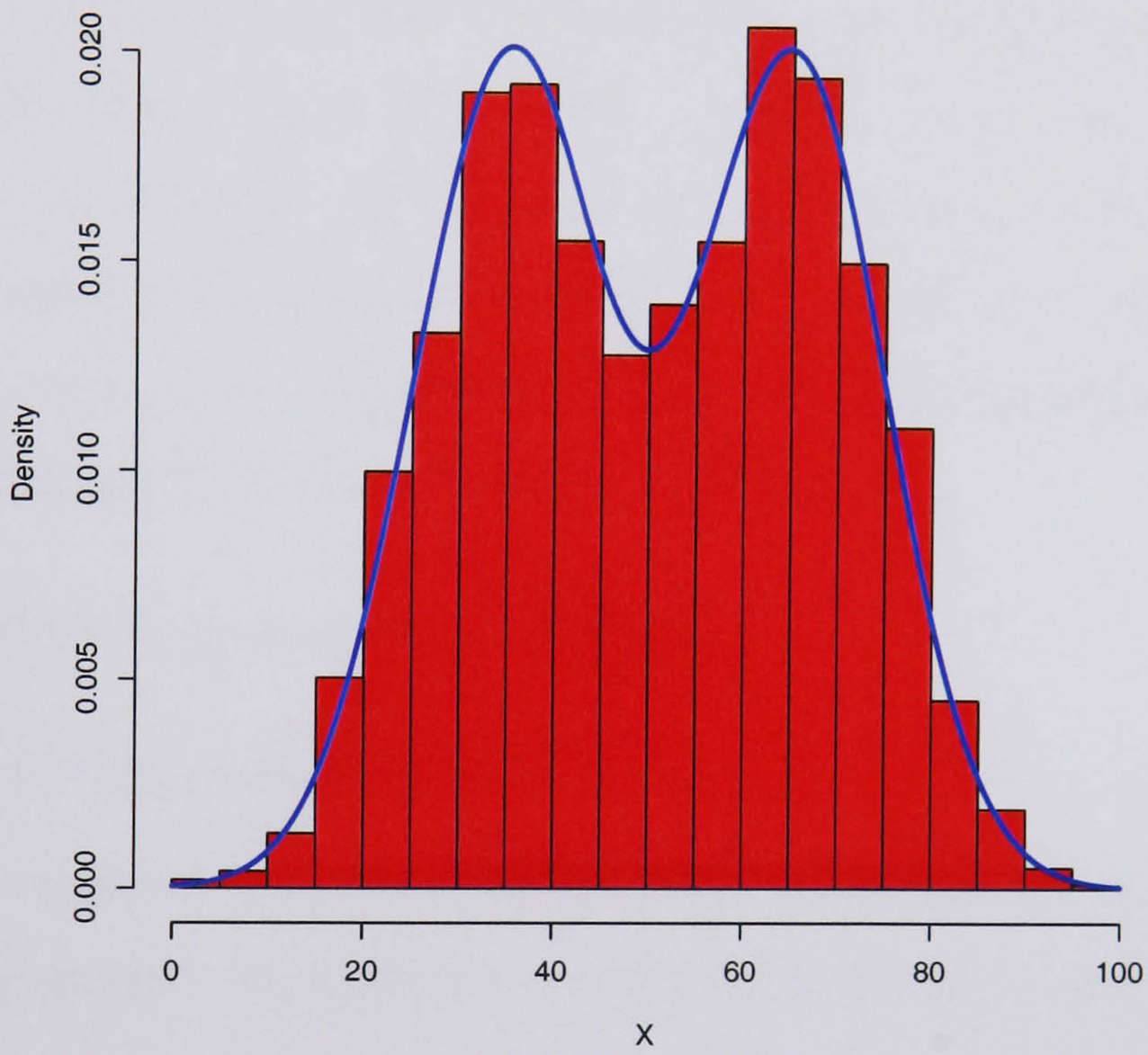


Figure 3.1: *The plot exhibits a histogram of the simulated data for* $n = 1981$. *The superimposed blue line represents the true density with the true parameters given as* $\psi = (\mu_1, \mu_2, \sigma^2, \pi_1, \pi_2) = (35, 65, 100, 1/2, 1/2)$.

Figure 3.1 provides the histogram of the data with a super-imposed *probability density function* plot. We can use this data plot as a starting point in estimating ψ. The example is characterised by three nice features, i.e. the clearly standing out two modes providing an insight into the underlying structure in the data, equal priors and equal variance. The last two features may guarantee stability of our clustering algorithm. In this univariate example, our choice of the initial parameters is guided by both the information about the data and the considerations mentioned in Section 3.2.2. With various initial choices of means, we consider both the restricted and unrestricted variances in estimating ψ. The main objective is to establish if the

estimated parameters converge to their MLEs. The estimate of ψ, ψ_∞, is obtained by running the EM algorithm initialised with the parameters ψ_0.

3.3.1 Equal group variance

In this example we restrict the group variances to be equal and consider a wide range of initial parameter choices; chosen to cover a range of plausible values, i.e., various μ_ks are used in conjunction with a set of small and large σ^2s. The main result, given $\sigma_1^2 = \sigma_2^2$, is that regardless of the initial parameters the algorithm always converged to the same locations, about the MLEs, differing only in the number of iterations required to get there. There are, however, exceptions in as far as convergence is concerned. As an example, consider the restricted variance scenario, $\sigma^2 = 2$, with the two non-negative μ_ks lying on both sides of the smaller of the two true means, e.g. $\mu_1 = 20$ and $\mu_2 = 40$. With these settings, the EM algorithm breaks down, as the computation of (3.7) fails due to the small variance.

1. The problem may be resolved by either
   - 1.1 Increasing μ_2 appreciably or
   - 1.2 Increasing σ^2 to some "feasible" level. Although this approach guarantees convergence, it entails an enormous increase in m as $\sigma^2 \to \infty$.
2. Decreasing μ_1 infinitely, with μ_2 and σ^2 unchanged, is unhelpful.
3. Increasing μ_1, requires $|\mu_1 - \mu_2| \gg 0$ and leads to a cross-over effect.

The foregoing features apply to the situation when the larger of the two true means is flanked by the chosen initial means, for instance $\mu_1 = 60$ and $\mu_2 = 70$. in this case, it is the infinite increase of the larger mean estimate, μ_2, that won't be helpful, so either σ^2 is inflated, μ_1 is decreased or μ_2 is decreased to attain $|\mu_1 - \mu_2| \gg 0$.

Table 3.1 provides a numerical summary of the performance of the algorithm for selected initial parameter choices. The final estimates are given in the middle set of columns. The last column of the table shows the number of iterations needed to attain convergence. Note, in particular, the relationship between the closeness of the μ_ks and the number of iterations, m, with m increasing inversely with the gap

Initial parameters - ψ_0				**Final estimates - ψ_∞**				**Iterations**
μ_1	μ_2	σ^2	π_1	μ_1	μ_2	σ^2	π_1	m
2.00	120.00	100.00	0.5	35.03	65.48	101.03	0.4913	22
18.00	25.00	100.00	0.5	35.03	65.48	101.03	0.4913	29
20.00	22.00	100.00	0.5	35.03	65.48	101.03	0.4913	74
25.00	85.00	10.00	0.5	35.03	65.48	101.03	0.4913	20
32.00	62.00	10.00	0.5	35.03	65.48	101.03	0.4913	17
90.00	115.00	10.00	0.5	35.03	65.48	101.03	0.4913	26

Table 3.1: *Summary of the EM algorithm-generated parameter estimates for six different initial parameters, based on the model with balanced class priors and equal group variance. The left and middle sets of columns provide the initial and final parameter estimates respectively, while the extreme right column gives the number of iterations. Note that running from all the initial points, ψ_0, the algorithm appears to accurately approximate the true parameters, $\psi = [\mu_1 = 35, \mu_2 = 65, \sigma^2 = 100]$. The same can be said about the mixing proportions, $\pi_1 = \pi_2 = 0.5$. A notable result is that all the ψ_0 converge to the same values regardless of the starting point. Note also the relationship between the closeness of the μ_ks and m.*

Figure 3.2: *The plots graphically summarise Table 3.1. The top row, left-to-right, panels correspond to the choices $[\mu_1 = 2,\ \mu_2 = 120]$, $[\mu_1 = 18,\ \mu_2 = 25]$, and $[\mu_1 = 20,\ \mu_2 = 22]$ respectively. The bottom row panels, from left to right, correspond to the initial parameters $[\mu_1 = 25, \mu_2 = 85]$, $[\mu_1 = 32, \mu_2 = 62]$ and $[\mu_1 = 90, \mu_2 = 115]$ in that order. The true parameters in each case are $(\mu_1 = 35, \mu_2 = 65, \sigma^2 = 100)$. In each panel, the two means are plotted against each other to show the convergence locations for each of the two means.*

between the means. For the same choices of μ_ks, an increasing σ^2 will lead to an increase in the number of iterations, particularly when the μ_ks are close together. As an example, the case of $\mu_1 = 20$ and $\mu_2 = 22$ requires 74 iterations with $\sigma^2 = 100$ and only 25 iterations when $\sigma^2 = 10$.

A major result from this example is that from many different initial ψ_0, the algorithm nicely converges to the same ψ_∞. This behaviour is attributable to the equal variance and priors. The general message is that while the choice of the initial group variances under the equal variance assumption may have only marginal effects with some μ_ks, i.e., by only increasing m, it could still make a pronounced impact under tight choices of μ_k or when the initial μ_ks flank one of the true parameters.

Figure 3.2 displays the converging parameter estimates of μ_1 and μ_2 for the data used in Table 3.1. Both Table 3.1 and Figure 3.2 exhibit that the impact of the initial parameters is on the number of iterations, rather than in the final estimates. For much tighter choices of μ_k, such as $\mu_1 = 44$ and $\mu_2 = 45$ the algorithm required over 200 iterations to converge. Further, the closer to the true means the μ_k choices are, the smaller the number of iteration needed.

3.3.2 Unequal group variance

In this section, we explore the EM algorithm when there are two variance parameters to estimate. We use the same simulated data as above. For ease of comparability, we use the same choices of μ_1 and μ_2 as in Table 3.1, but with a wide range of σ_k^2 sets. Again, the true parameter values are $\psi = (\mu_1 = 35,\ \mu_2 = 65$ and $\sigma^2 = 100)$. Table 3.2 provides a numerical summary of the performance of the algorithm for six different sets of ψ_0. The left and middle column sets provide the initial and final parameter estimates respectively, while the number of iterations necessary to attain convergence are given in the extreme right column.

A notable similarity between the ψ estimates in Table 3.2 and those in Table 3.1 is that they almost all converge to the same location regardless of the algorithm's starting point. However, as a consequence of setting $\sigma_1^2 \neq \sigma_2^2$, the estimates, ψ_∞ here

Initial parameters - ψ_0					**Final estimates - ψ_∞**					**Iterations**
μ_1	μ_2	σ_1^2	σ_2^2	π_1	μ_1	μ_2	σ_1^2	σ_2^2	π_1	m
2	120	10	30	0.5	36.28	66.57	119.71	86.60	0.53	97
18	25	5	6	0.5	36.28	66.57	119.71	86.60	0.53	162
20	22	10	30	0.5	36.28	66.57	119.71	86.60	0.53	110
20	40	199	200	0.5	36.28	66.57	119.71	86.60	0.53	151
32	62	10	30	0.5	36.28	66.57	119.71	86.60	0.53	100
44	45	0.1	100	0.5	36.28	66.57	119.71	86.59	0.53	209
90	115	0.1	100	0.5	89.51	50.46	0.03	330.84	0.0016	30

Table 3.2: *The table provides a numerical summary of the EM algorithm-generated parameter estimates for seven different initial parameters, based on the model with unequal group variance. The left and middle column sets provide the initial and final parameter estimates respectively, while the number of iterations are given in the extreme right column. With the exception of the last row entries, all ψ_∞ estimates are the same, regardless of the starting point, although they deviate more from ψ than those in Table 3.1. The number of iterations in each case has increased, typically, to three digits in contrast to the two-digit values under the equal variance assumption. With $\sigma_1^2 = 10$ and $\sigma_2^2 = 30$, the number of iterations in the last row was 139 and different estimates obtained. These results underline the critical role played by the assumptions made about the model in estimating its parameters.*

deviate more from ψ than those in the previous two examples under the $\sigma_1^2 = \sigma_2^2$ scenario. Note the cross-over effects, in the last row, for both the μ_ks and σ_k^2s brought about by that particular choice of ψ_0. Further, there are also exceptions relating to the choice of the ψ_0, as summarised below.

1. The algorithm is able to cope with initial μ_ks, around one of the true parameters, and tight σ_k^2s, but only if the latter are big enough. With $\mu_1 = 20$ and $\mu_2 = 40$, the algorithm is guaranteed to converge over a wide range of large σ_k^2s, but breaks down as the variances approach zero. The solution to this problem follows the same logic as in Section 3.3.1.

2. The initial values $\mu_1 = 90$, $\mu_2 = 115$, $\sigma_1^2 = 0.1$ and $\sigma_2^2 = 100$ leads the algorithm to "ignore" one of the modes, returning the following final estimates, $\mu_1 = 89.51$, $\mu_2 = 50.47$, $\sigma_1^2 \approx 0$, $\sigma_2^2 = 330.840$, $\pi_1 \approx 0$, and $\pi_2 \approx 1$. Clearly, the choice of the initial parameters, ψ_0, may lead to complete masking effects.

3. The scenario with $\sigma_1^2 \neq \sigma_2^2$ is generally associated with a much larger number of iterations than the when $\sigma_1^2 = \sigma_2^2$.

Figure 3.3 provides some graphical comparisons of the behaviour of the algorithm un-

der restricted and unrestricted variances for some values of ψ_0 in Table 3.2. Clearly notable features include the cross-over effect, masking and a large number of iterations resulting from initialising the algorithm with "odd" parameters. The results in both Section 3.3.1 and Section 3.3.2 underline the crucial role played by the assumption of group variance as we seek to detect clusters in data. The number of iterations is clearly not a serious issue, as the enhanced computing power can easily cope with that. However, losing a cluster to masking is, apparently, a more serious problem. In our case, masking was a direct consequence of initialising the algorithm with "odd" parameters, hence the need for a careful initialisation.

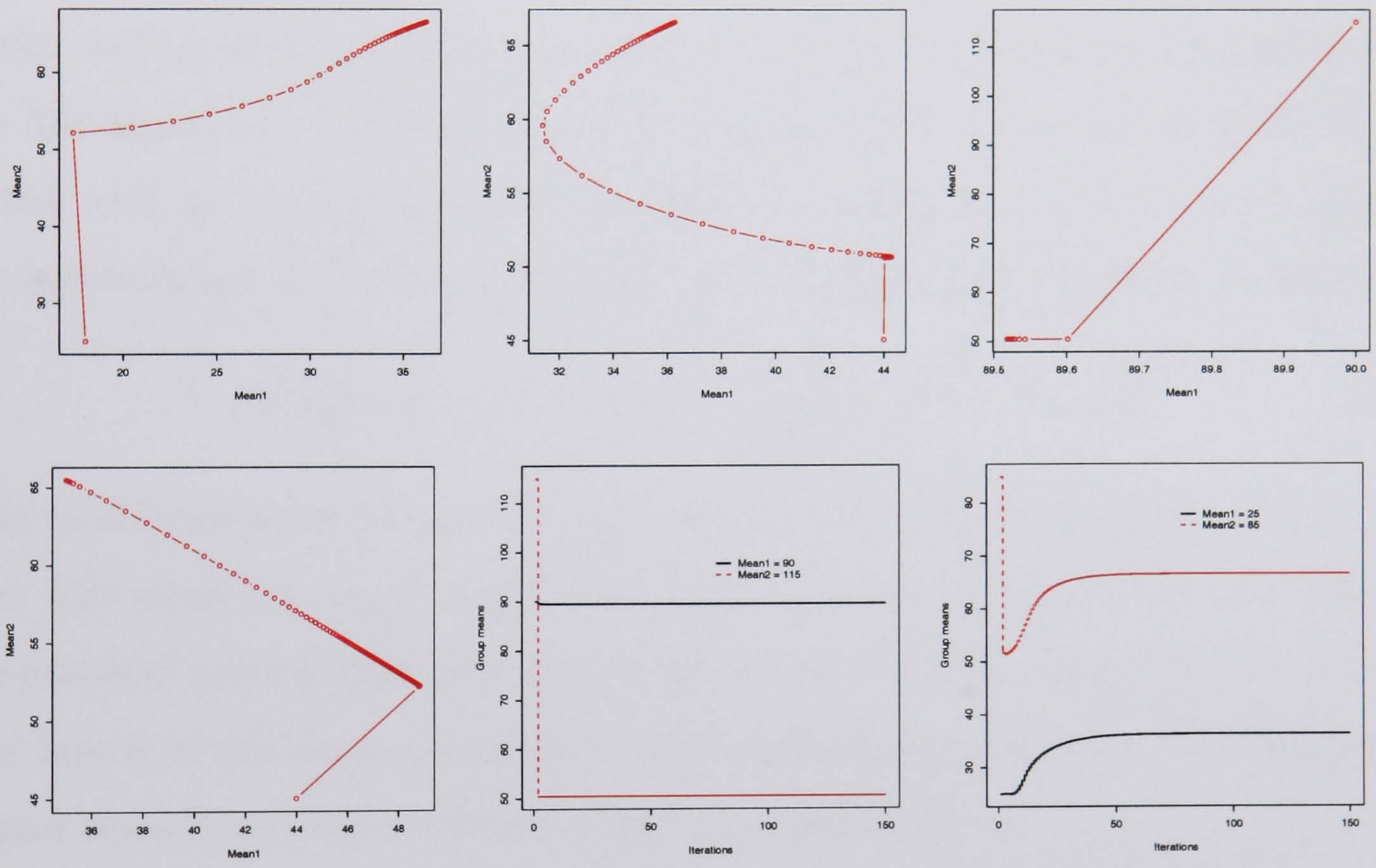


Figure 3.3: *The panels provide a graphical comparison for the restricted and unrestricted variance scenarios. The top row panels represents a selection of initial choices from Table 3.2. From left to right, the panels correspond to the initial means* $(\mu_1 = 18,\ \mu_2 = 25)$, $(\mu_1 = 44,\ \mu_2 = 45)$ *and* $(\mu_1 = 90,\ \mu_2 = 115)$ *respectively, with* $\sigma_k^2 s$ *as given in Table 3.2. The number of iterations for the RHS panel has dropped from 139, under* $\sigma_1^2 = 10$ *and* $\sigma_2^2 = 30$, *down to only 30. The LHS bottom panel corresponds to the tight choice* $\mu_1 = 44$ *and* $\mu_2 = 45$, *with* $\sigma_1^2 = \sigma_2^2 = 100$. *Convergence, in this case, required 214 iterations, just about what it took under unequal variance. These values are approximately ten times the average number of iterations in Table 3.1. The middle panel exhibits the cross-over effect resulting from the choice* $(\mu_1 = 90,\ \mu_2 = 115)$ *under* $\sigma_1^2 = \sigma_2^2 = 100$. *Compare the middle to the RHS panel, which is based on* $\mu_1 = 25,\ \mu_2 = 85$ *and the same choice of initial group variances.*

We picked a nice example, with two modes. Our choices of plausible initial param-

eters yielded convincing results for both restricted and unrestricted σ_k^2. The EM algorithm in both Figure 3.2 and Figure 3.3 converges to the same values, although the ψ estimates in the latter appear to be less accurate than those in the former. The choice of the initial parameters does mainly seem to affect the performance of the algorithm in both cases only as far as the number of iterations to convergence is concerned. However, some weird choices should be avoided in parameter initialisation. Depending on the magnitude of the initial values, the convergent point may be far from the corresponding MLEs.

3.4 Application under the regression model

In this section the regression parameters of a mixture model are estimated by using the EM algorithm. Data generation for the regression example is as in (2.14) and (2.15), with an equal number of observations on each arm and the data spanning the intersecting region of the two arms. The basic model to be fitted is defined as

$$\phi(y_i|x_i, C_i = k) \propto \exp\{-\frac{1}{2\sigma_k^2}(y_i - \hat{\beta}_{0k} - \hat{\beta}_{1k}x_i)^2\} \tag{3.11}$$

This model generates two equally likely arms in $\mathbb{R}^2$, representing two different clusters. The algorithm ought to pick both of the two arms of the data set and, typically, the posterior probabilities should converge to about 0.5 for each of the two clusters. The layout of the data is such that the regression slopes for the first and second cluster should converge to about -1 and +1 respectively.

3.4.1 Application setup and maximisation of the likelihood

The regression application here is similar to that of the $X-$shape problem of Chapter 2, with two main differences. Firstly, in the current application we simultaneously fit the model to both arms of the data and, secondly, we introduce an additional, *unobservable,* variable of indicators, $\{z_{ik}\}$ into the model. Then given x_i and z_i, the basic model to be considered is given as follows

$$[1^T z_i]y_i = \beta_0^T z_i + [\beta_1^T z_i]x_i + \epsilon_i, \tag{3.12}$$

where $z_i = (z_{i1}, ..., z_{iK})$, $\beta_0^T = (\beta_{01}, ..., \beta_{0K})$, $\beta_1^T = (\beta_{11}, ..., \beta_{1K})$, and $\sigma_v = (\sigma_1, ..., \sigma_K)$.

We can now compute the expected complete likelihood of y_i as follows

$$\mathrm{L}(\psi) = \prod_{i=1}^{n} \frac{1}{z_i^T \sigma_v \sqrt{2\pi}} \exp\Big\{ - \sum_{i=1}^{n} \Big(\frac{[1^T z_i] y_i - \beta_0^T z_i - [\beta_1^T z_i] x_i}{2 z_i^T \sigma_v} \Big)^2 \Big\}, \tag{3.13}$$

which after taking logarithms on both sides and differentiating the log-likelihood with respect to each of the parameters we can obtain the parameters $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\sigma}^2$. Indeed, if we let the mean of y_i in the k^{th} component be defined as

$$\bar{y}_k = \frac{\sum_{i=1}^{n} z_{ik} y_i}{\sum_{i=1}^{n} z_{ik}}, \tag{3.14}$$

then the estimate of β_1 for the particular k^{th} class can be computed as follows

$$\hat{\beta}_{1k} = \frac{\sum_{i=1}^{n} z_{ik}(x_i - \bar{x}_k)(y_i - \bar{y}_k)}{\sum_{i=1}^{n} z_{ik}(x_i - \bar{x}_k)^2}, \tag{3.15}$$

which provides an estimate of β_1 for the particular k^{th} group, hence we deduce

$$\hat{\beta}_{0k} = \bar{y}_k - \hat{\beta}_{1k} \bar{x}_k, \tag{3.16}$$

thus providing an estimate of β_0 for the particular k^{th} group. The variance for the k^{th} component is given as

$$\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{n} z_{ik}(y_i - \beta_{0k} - \beta_{1k} x_i)^2}{\sum_{i=1}^{n} z_{ik}} = \frac{\sum_{i=1}^{n} z_{ik}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} z_{ik}}. \tag{3.17}$$

If we make an assumption of a common variance, then (3.17) transforms to

$$\hat{\sigma}^2 = \frac{\sum_{k=1}^{K} \sum_{i=1}^{n} z_{ik}(y_i - \beta_{0k} - \beta_{1k} x_i)^2}{n}. \tag{3.18}$$

As in the previous section, the number of groups is $K = 2$ and $p = 1$. To run the EM algorithm, we need to initialise the parameters. Then using the current parameters we'll compute the weights in the E-Step, as shown in (3.7), and update the parameters in the M-Step. The case of observed data in (3.14) through (3.18) provides guidance on how to proceed with unobserved data. In fact, we need only replace the unobserved class labels generated in the E-Step by their expectations conditional on the data and subsequently maximise them in the M-Step. This process will be executed iteratively until convergence. We implement the EM algorithm for mixtures for both restricted and unrestricted variances.

3.4.2 Equal group variance

In this section, we restrict the group variances to be equal and run the algorithm for three different randomly generated z_{ik}s. The first z_{ik} is binomially generated with equal probabilities for the ones and zeroes, the second is based on the probabilities 0.1 and 0.9 and the third on 0.75 and 0.25. The sample size used was $n = 50$, $\sigma^2 = 100$ and the true least squares regression coefficient estimates, assuming known group membership, are $\beta_{01} = -0.01250$ and $\beta_{11} = 0.98479$, for the first group, and $\beta_{02} = 0.8987$ and $\beta_{12} = -0.9995$ for the second.

	Initial estimates				**Final estimates**			
	β_{01}	β_{11}	σ^2	π_k	β_{01}	β_{11}	σ^2	π_k
Group 1	1.1177	0.0317	38.3252	0.54	0.8796	-1.0004	0.2021	0.5065
Group 2	-0.0463	-0.0149	38.3252	0.46	-0.0332	0.9840	0.2021	0.4935
Group 1	1.6278	0.6042	36.4644	0.16	-0.0332	0.9840	0.2021	0.4935
Group 2	0.4395	-0.0639	36.4644	0.84	0.8796	-1.0004	0.2021	0.5065
Group 1	0.0618	-0.0789	35.8324	0.72	0.8796	-1.0004	0.2021	0.5065
Group 2	3.1221	0.4745	35.8324	0.28	-0.0332	0.9840	0.2021	0.4935

Table 3.3: *The table shows three different sets of initial model estimates on the LHS and the algorithm's final estimates, nicely converging to the MLEs. The parameters in the table, top-down, are based on the algorithm with three different z_{ik}s and probabilities of success 0.5, 0.1 and 0.75 respectively and the assumption $\sigma_1^2 \neq \sigma_2^2$. Regardless of the initial points, the parameters appear to converge to the same MLEs.*

Table 3.3 gives a numerical summary of the performance of the algorithm based on the three choices of z_{ik}. As to be expected, the parameters converge nicely to their MLEs, after just over 20 iterations. Note, in particular, the change in slope signs for the first set of entries. The slope for the first group starts with a positive slope and converges to a negative slope, while that of the second group starts with a negative and converges to a positive slope. Figure 3.4 graphically summarises Table 3.3. Left to right, the panels correspond to the three sets in the table, from top to bottom. All panels show that the algorithm was successful in detecting the two arms of the data. The number of iterations necessary to attain convergence inversely decreased with the probabilities of success used in the construction of z_{ik}.

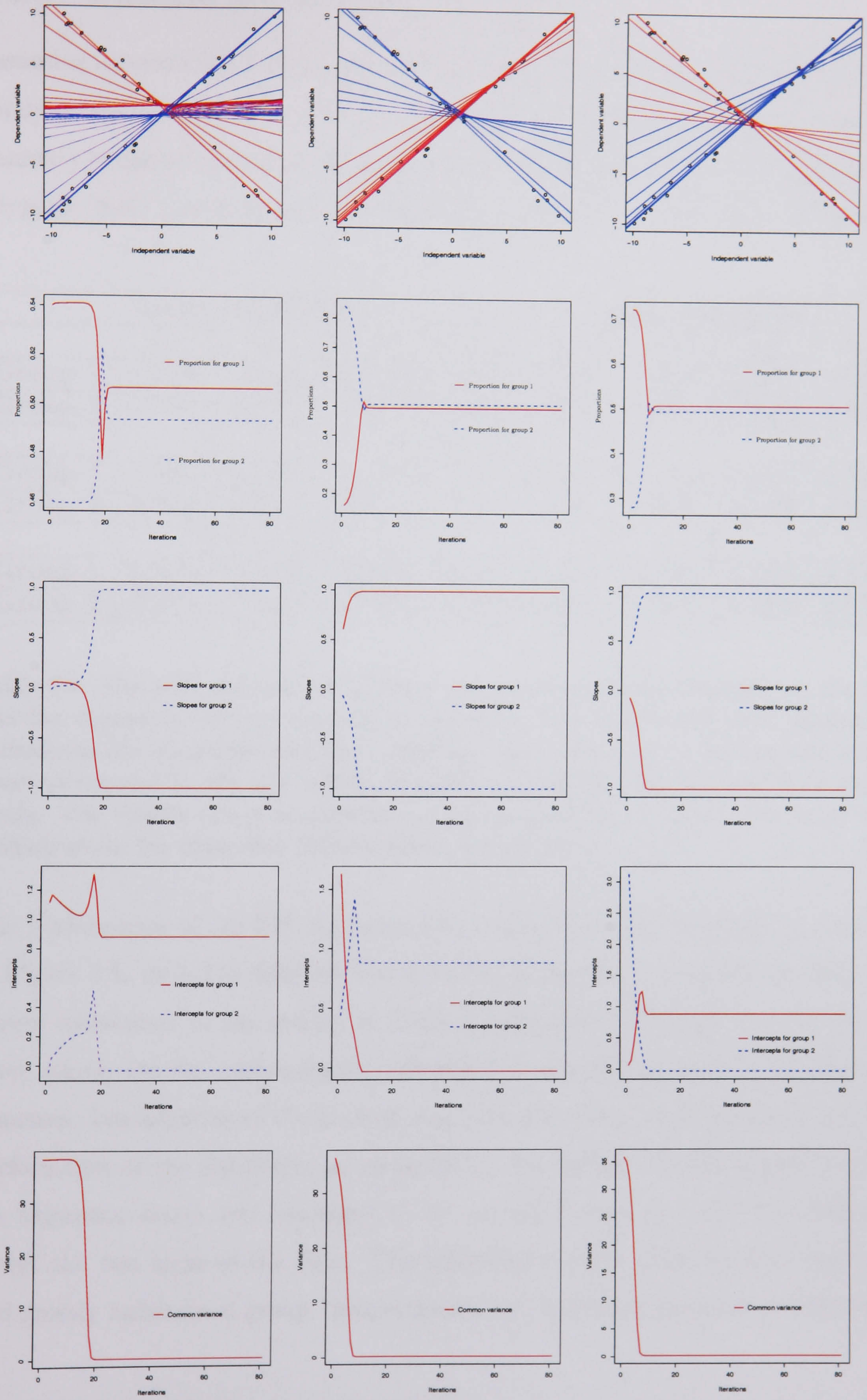


Figure 3.4: *The panels, left to right, graphically summarise Table 3.3 top to bottom. For the LHS panel, z_{ik} was binomially generated with equal probabilities for ones and zeroes. For the middle and RHS panels the corresponding probabilities were 0.1, 0.9 and 0.75, 0.25 respectively. Typically, a z_{ik} based on equal proportions of the binary digits is associated with a larger number of iterations than a rather "biased" z_{ik}. The "proportions" in the second row are, de facto, estimated posterior probabilities.*

3.4.3 Unequal group variance

Assuming unrestricted variance, the EM algorithm was initialised with the same randomly generated *boolean* vectors as in Section 3.4.2. Table 3.4 provides a numerical summary of the performance of the algorithm. The middle rows in the table exhibit a typical "bad" initialisation and convergence. Here too, $n = 50$ and $\sigma^2 = 100$.

Initial estimates					**Final estimates**			
	β_{01}	β_{11}	σ_k^2	π_k	β_{01}	β_{11}	σ_k^2	π_k
Group 1	0.3933	-0.0030	33.7616	0.44	0.8794	-1.0004	0.1813	0.5083
Group 2	0.8555	0.0636	42.2922	0.56	-0.0332	0.9840	0.2238	0.4917
Group 1	5.7816	-0.0139	8.3971	0.1	4.8124	-0.1295	9.2284	0.6204
Group 2	0.0640	0.0204	38.7199	0.9	-6.3311	0.0400	6.5959	0.3796
Group 1	0.0472	0.0567	36.3090	0.74	-0.0332	0.9840	0.2238	0.4917
Group 2	2.9721	0.1937	40.4010	0.26	0.8794	-1.0004	0.1813	0.5083

Table 3.4: *The table shows three different sets of initial model estimates on the LHS and the algorithm's final estimates on the RHS. The parameters were obtained by initialising the algorithm with three different z_{ik}s. From top to bottom, the sets of rows correspond to the z_{ik}s with probabilities of success 0.5, 0.1 and 0.75 respectively. The middle two rows represent a typical situation in which, due to a "bad" initialisation, the algorithm fails to detect the two arms.*

The performance of the EM algorithm for mixture models is graphically presented in Figure 3.5, for three different sets of initial parameters. From left to right, the panels correspond to the entries in Table 3.4 top down. Beneath each of the top three panels are the corresponding convergence patterns for each of the four parameters. Initialisation of the unobservable variable plays a crucial role in the final performance of the algorithm, as exhibited by the middle column-panels, in which the algorithm starts and converges to two almost horizontal lines, thus failing to detect the two arms of the data. The algorithm returns large variance estimates and grossly imbalanced group "proportions", i.e., estimated posterior probabilities.

3.5 Concluding remarks

In this chapter, we used the EM algorithm for mixtures to perform clustering under both univariate and regression settings. In both cases the data sets had two

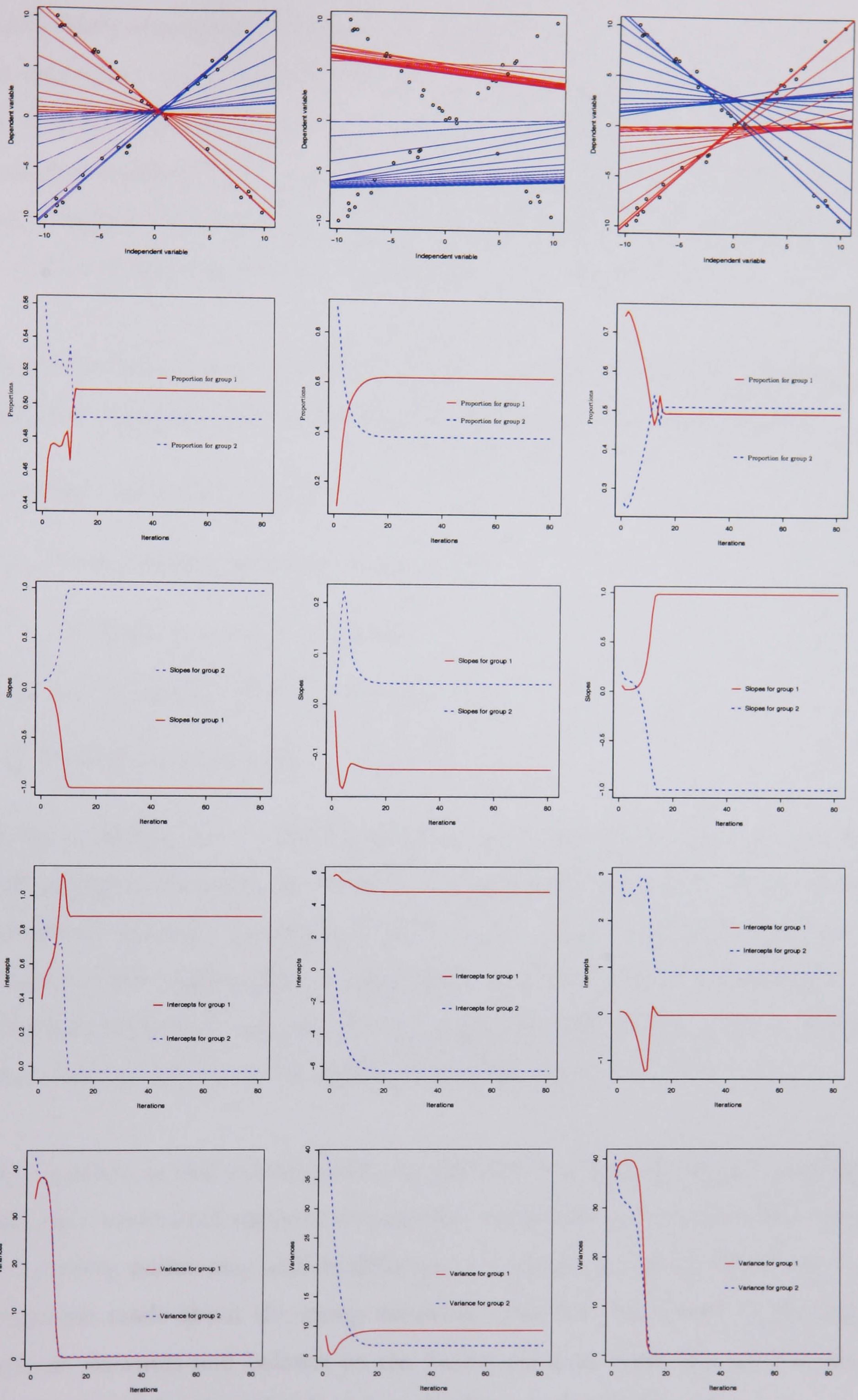


Figure 3.5: *Left to right, the top three panels show convergence patterns for the EM algorithm for mixtures invoked with three different z_{ik}s as shown in Table 3.4 top down. Below each of the "butterfly" plots are the corresponding converging patterns for the parameters. Note that, due to the impact of initialisation in the middle column-panels, the algorithm not successful in detecting the two arms, resulting in large variance estimates and grossly imbalanced group proportions. Again here, the "proportions" in the second row are, de facto, estimated posterior probabilities.*

equally-likely conspicuous clusters. The algorithm sought to allocate each case from the data set to one of the two clusters. As far as the simultaneous detection of the two clusters is concerned, an obvious measure of performance turns out to be the convergence points for the model parameters and the regression coefficients respectively. Computationally, the number of iterations needed to attain convergence may be of little concern, but the factors affecting it are interesting.

The general behaviour of the EM algorithm for mixtures under both univariate and regression is largely dictated by the combination of the following factors.

1. Initialisation of the algorithm.

2. The assumption on group variances, i.e.

   2.1 Equal or restricted variance.

   2.2 Unequal or unrestricted variances.

3. Data structure and size.

For the regression case, we chose to initialise the algorithm through a random choice of the group membership, z_{ik}. With such a choice, the algorithm would sometimes converge to the right places and sometimes not. Under both restricted and unrestricted variance assumptions, the algorithm will behave fairly well if initialised with moderately horizontal and intersecting lines from a rather balanced z_{ik}. A grossly imbalanced z_{ik} may cause the algorithm to break down.

The algorithm is very robust under the restricted variance assumption and less so under the unrestricted variance assumption. Figure 3.6 demonstrates how "equally bad" starting points may lead to different convergence locations, depending on the assumption made about the group variances. The first two panels in the top row represent skewness and balance on the horizontal axis, while the third shows the performance of the algorithm based on the first panel and $\sigma_1^2 = \sigma_2^2$.

Note that although the initial fits in the top RHS and bottom LHS panels in Figure 3.6 are almost horizontal and farther apart, the algorithm is able to intersect them

and detect the two arms. On the other hand, the bottom second and third panels are based on the top left panel and $\sigma_1^2 \neq \sigma_2^2$. The corresponding probabilities of success in the initial z_{ik} are 0.1 and 0.75 respectively. In both cases, the algorithm never recovers from a bad start and ends up converging to roughly horizontal lines.

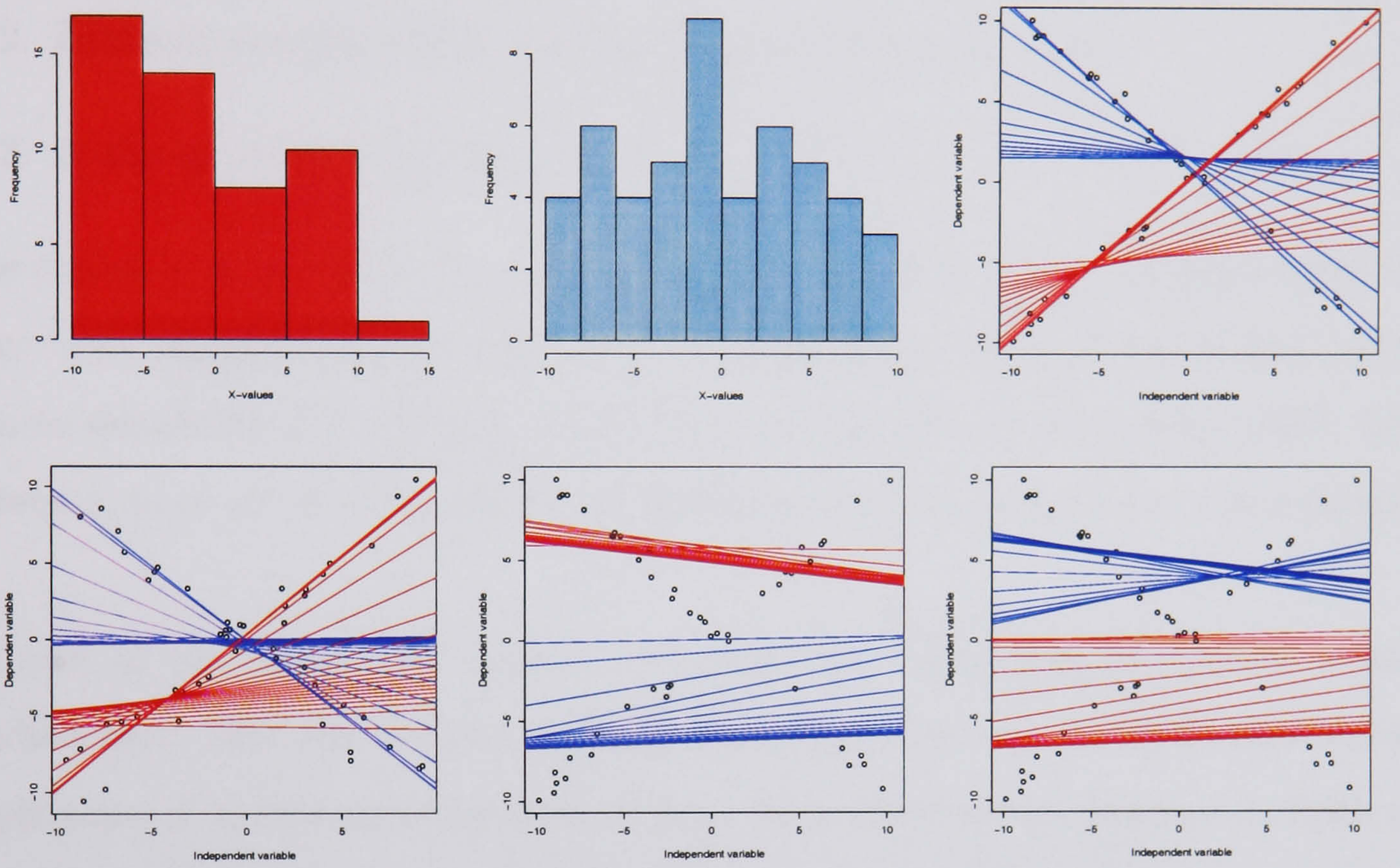


Figure 3.6: *Typical examples of the performance of the algorithm based on imbalanced and balanced data, with the x co-ordinates shown in the top left and middle panels. The corresponding performances of the algorithm, based on $\sigma_1^2 = \sigma_2^2$, are given in the top right and bottom left panels respectively. Note how the patterns contrast with those of the "equally bad" starting points in the bottom middle and right panels, based on the assumption $\sigma_1^2 \neq \sigma_2^2$. The main aim of these plots is to establish whether "skewness" affects the general performance of the algorithm.*

The allocation of points to clusters was based on the estimated posteriors, given estimated means and variances — which depended much on the z_{ik}s. Typically, if the z_{ik}s are based on random equal proportions of the binary digits, the algorithm will never go wrong. This is because "bad" convergence is a consequence of the algorithm starting and ending with observations from each half of the two arms for the first group and from the remaining two halves for the other. Indeed, the nice and unambiguous separation of the two structures in both our univariate and regression examples plays an important role in the behaviour of the algorithm.

Based on the results in this chapter, it is probably best to assume equal group variances, even when they are not, as that adds stability to the algorithm. The following factors are also likely to affect the performance of the EM algorithm for mixtures. We present them as issues requiring further investigation.

1. Unequal class priors.

2. Different spacing within groups along the horizontal axis.

3. A change in the variance.

The choice of K is one of the major issues in clustering. Indeed, various approaches have been suggested in the literature — most notably, using kernel density estimation to determine the modality of the data as proposed by Silverman (1981, 1986). However, such approaches will be successful only if the clusters are well-separated.

As part of the search for "good" estimates, the underlying population must be studied well. This may include multiple sampling, whenever possible, and repeated application of clustering techniques such as those discussed in Chapter 2. With even a crude knowledge of the form of the data, the choice of the initial parameters, ψ_0, may be "guided" to yield convincing ψ_∞. We confined our findings to the two-group data structure for easy comparability with the results in the previous Chapter. To handle a single or multiple arms scenarios, the algorithm can be adapted accordingly. Further generalisation of the performance of the method is envisioned as a potential future work. In particular, with applications based on different forms of clusters such as the lunar-star and embedded circles type of clusters. In general we do not expect the method to work well on such clusters, hence some modification will be necessary.

Finally, more recently, Figueiredo and Jain (2002) provide an algorithm that seeks to avoid initialisation problems for the EM algorithm for mixtures. They propose an unsupervised algorithm for learning a finite mixture model from multivariate data. They consider the twin problems of estimating the number of components and the parameters defining the mixture model. They try to "mitigate" initialisation dependence by starting with a large number of clusters and evolving down.

Chapter 4

Classical classification methods

4.1 Introduction

Classification or, equivalently, discriminant analysis, involves K classes where observations from each class follow some distribution, typically in $\mathbb{R}^p$. Given an observation x from an unknown class, the objective is to classify it into one of the K classes. The probabilistic framework consists of a distribution for a pair of random variables (y, x), where $y \in \{1, ..., K\}$ labels the class C_k and $x \in \mathbb{R}^p$ represents an observation. Suppose that the classes C_k have prior probabilities $\pi_k = P(y = k)$ and the observations have conditional densities $f_k(x) = f(x|y = k)$. Then, the classification problem is to assign each new case to one of the K classes. The conditional probability of group membership is given by the posterior probability

$$P(y = k|x) = \frac{\pi_k f_k(x)}{\sum_{k=1}^{K} \pi_k f_k(x)} \tag{4.1}$$

The expression in (4.1) is a typical example of a **classifier** function — a fundamental concept in classification. It describes a rule by which new observations are allocated to known classes. One way to allocate a class for x is to derive the value of k which maximises (4.1), which is called the Bayes' rule (or the Maximum Likelihood rule if the π_ks are equal). In an ideal setting the π_ks and $f_k(x)$s are known and it is straightforward to apply the rule. We can think of the Bayes' theorem in terms of updating our knowledge about y as new information x arrives. Theoretically, we can assume that the π_ks and the $f_k()$s are known. The Bayesian error is a benchmark for any classifier, and, in a 2-class setting, it is defined as follows

$$\xi_{B,\text{pop}} = \pi_1 P(x \in R_2|y \in C_1) + \pi_2 P(x \in R_1|y \in C_2), \tag{4.2}$$

where R_k and C_k correspond to the predicted and true classes respectively. The conditional probabilities are given for the event that an observation x is allocated to class k while it actually belongs to class $\overline{k}$. For an overlapping scenario, at least one of the conditionals will be greater than zero, thus making $\xi_{B,\text{pop}} > 0$.

In practice, however, we usually do not know the distribution within each class and so we need to devise a discriminant rule based on some "training data", a set of data observed in the past with known true labels. An allocation rule is devised by showing examples from each class to a "classifying algorithm" with the hope that it would be able to learn and "remember" similar cases in the future. The "classifying algorithm" can be described as follows. Suppose the measurement space Ω contains all possible variables to be used for prediction. The classifier is a function of the training data and describes the rule by which membership in C_k is assigned to every observation in Ω. It is the function, $\phi(x)$, defined on Ω such that $\phi(x) : \Omega \rightarrow \{1, ..., K\}$, i.e. the rule maps vectors onto one of the known classes.

Whether a new observation will be assigned to its correct class or not will depend on the "accuracy" of the devised classification rule. Accuracy is a crucial concept in classification, which we can "crudely" describe as the rate at which predicted labels correspond to the true labels. How well or badly the rule performs is usually revealed by testing it on an independent data set called the "test data set."

Classification problems have been addressed by different authors and different methods. Mardia *et al.* (1979) and McQueen (1967) are standard references that provide a detailed account of classical classification approaches, such as discriminant analysis; Breiman *et al.* (1984) thoroughly discusses classification from the classification and regression trees (CART) perspective. Artificial Neural Networks (ANN) and Support Vector Machines (SVM) — another quite effective method for tackling classification problems with a binary nature — are two popular classification methods that are widely applied — a condensed and comprehensive account of which is given by Hastie *et al.* (2001). Other useful references for applied Artificial Neural Networks are Looney (1997) and Bishop (1995). Support Vector Machines are detailed in Vapnik (1995) and Cortes and Vapnik (1995), while Goldberg (1989)

provides a comprehensive account of Genetic Algorithms.

This chapter has the following layout. We discuss parametric discriminant analysis in Section 4.2. Section 4.3, is devoted to non-parametric discriminant analysis, in particular the kernel density estimation and the nearest neighbour classification rule. Section 4.4 provides a detailed discussion of the concept of learning and the methods of assessing the performance of classifiers. Classification trees are a subject of Sections 4.5 and 4.6 — providing the fundamentals and illustrations respectively.

4.2 Parametric discriminant analysis

Although the Bayes' rule is a powerful tool for estimating the posterior probabilities when both the priors and the densities are known, it is not always possible to apply it as the priors and densities are often unavailable and have to be estimated from data. A typical classification technique would then seek to estimate the posteriors in (4.1) as accurately as possible. In the following exposition we illustrate the case of using training data in the context of parametric discriminant analysis. Typically, the training data set, T, is defined as in (4.3), where x_i represents n training observations and $y_i = k$ if $x_i \in C_k$, where k denotes the class label $k = 1, ..., K$.

$$T = \{(x_1, y_1), ..., (x_n, y_n)\} \tag{4.3}$$

4.2.1 A two-class normal distribution example

Suppose the densities in (4.1) are known but include unknown parameters, θ. The unknown parameters can be estimated as $\hat{\theta}_k$, which is the MLE from the training data for class k. Since the priors may also be unknown, they can be estimated as the sample proportions, $\hat{\pi}_k = p_k$ (or from prior knowledge), where π_k and p_k are class priors and proportions respectively. Consider a two-class problem, where C_1 and C_2 are two normal populations, with known parameters so

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\Big[- \frac{(x - \mu_k)^2}{2\sigma_k^2} \Big] \text{ where } k = 1,\ 2. \tag{4.4}$$

If we consider class priors, π_ks, the maximum likelihood rule will allocate an observation to C_1 if $\pi_1 f_1(x) > \pi_2 f_2(x)$ and to C_2 otherwise. Note that the foregoing

condition will arise only if $f_1(x)/f_2(x) > \pi_2/\pi_1$, that is,

$$\frac{\sigma_2}{\sigma_1}\exp\Big\{-\frac{1}{2}\Big[\frac{(x-\mu_1)^2}{\sigma_1^2}-\frac{(x-\mu_2)^2}{\sigma_2^2}\Big]\Big\} > \frac{\pi_2}{\pi_1} \tag{4.5}$$

Taking logarithms on both sides of (4.5), multiplying by 2 and reorganising yields

$$2\ln\frac{\pi_1\sigma_2}{\pi_2\sigma_1} > x^2\Big[\frac{1}{\sigma_1^2}-\frac{1}{\sigma_2^2}\Big]-2x\Big[\frac{\mu_1}{\sigma_1^2}-\frac{\mu_2}{\sigma_2^2}\Big]+\Big[\frac{\mu_1^2}{\sigma_1^2}-\frac{\mu_2^2}{\sigma_2^2}\Big] \tag{4.6}$$

as the discrimination rule, given imbalanced priors. The maximum likelihood rule is a special case of the Bayes rule when the populations are equally likely. Using this simple case, we can consider different scenarios involving the two variances. If $\sigma_1 > \sigma_2$, then the coefficient of x^2 is negative and the set of xs for which (4.6) holds will fall into two different regions of low and high x values. If $\sigma_1 = \sigma_2$ the quadratic component in (4.6) disappears and we have a discriminant rule by which $f_1(x) > f_2(x)$ if $|x-\mu_2| > |x-\mu_1|$. In other words, assuming $\mu_2 > \mu_1$, the maximum likelihood rule will allocate x to C_2 if $x > \frac{1}{2}(\mu_1+\mu_2)$ and to C_1 otherwise.

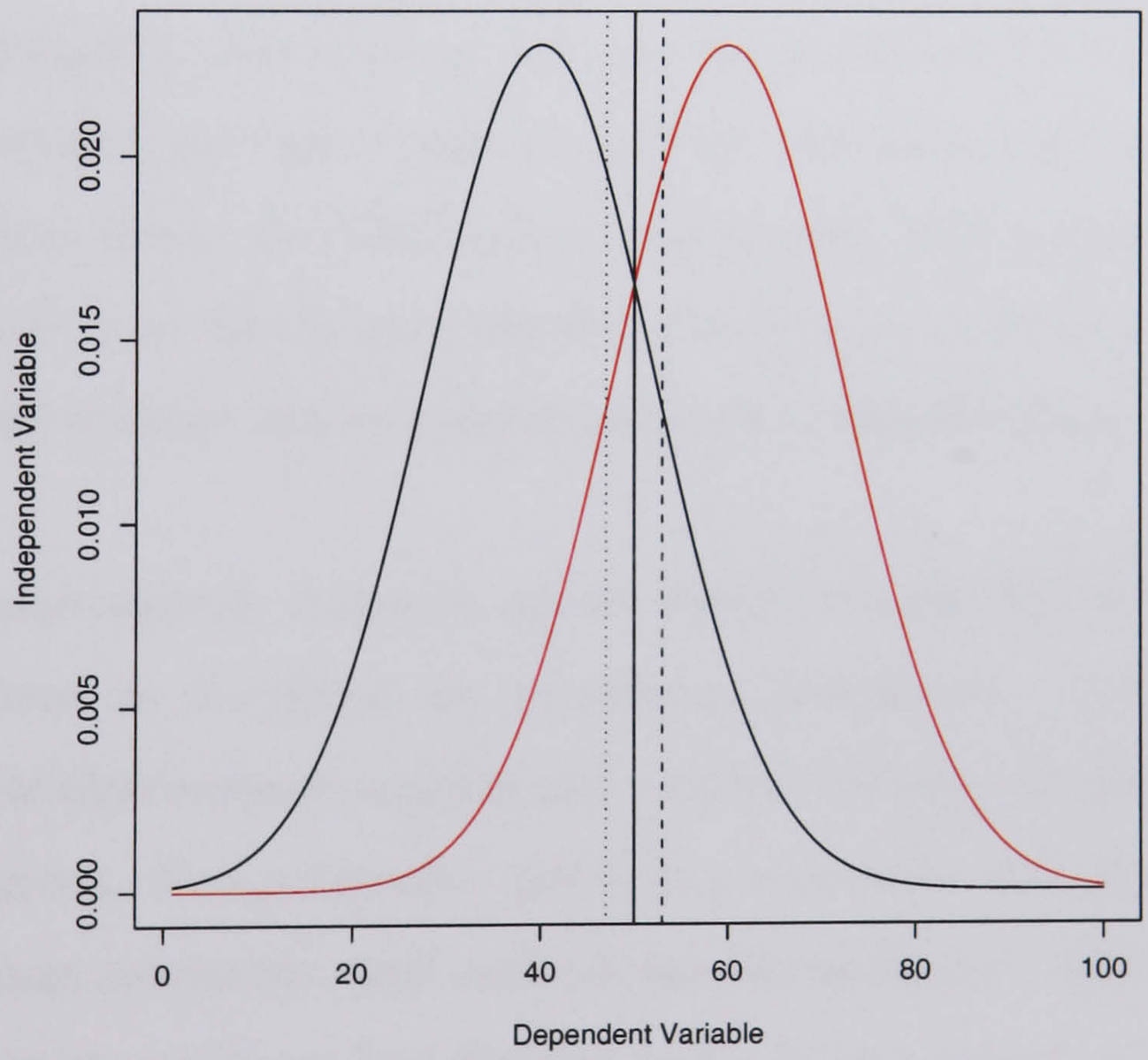


Figure 4.1: *Bayes' rule discrimination between two normals with different means and the same standard deviation is shown with possible deviations. The LHS and RHS normals correspond to the class labels C_1 and C_2 respectively. The solid vertical line represents the critical point $X_c = (\mu_1+\mu_2)/2$ and the dotted and dashed lines reveal how the estimated critical point may deviate from the theoretical.*

To get a clear insight, consider Figure 4.1 of two normal populations with $\mu_1 < \mu_2$ and $\sigma_1 = \sigma_2 = \sigma$. Further, let us denote the LHS and RHS normals by regions

R_k, $k = 1, 2$, corresponding to classes C_k respectively. The linear discriminant critical value is given as $X_c = (\mu_1 + \mu_2)/2$. The classifying rule allocates x to C_2 if $x > X_c$ and to C_1 otherwise. But this applies only to the case when the exact distributions are known and the population parameters can be computed explicitly.

When the parameters are to be estimated from samples, the μ_ks and σ_ks for the k^{th} class, are replaced by their sample estimates, $\bar{x}_k$ and s_k, respectively. Then the above population-based critical point transforms into the sample-based version, $\bar{X}_c = (\bar{x}_1 + \bar{x}_2)/2$, leaving accuracy dependent on the sample representativeness. The resulting critical point may deviate about the Bayesian, as shown by the non-solid lines in Figure 4.1 — making the quantity $|X_c - \bar{X}_c|$ an interesting one. Further aspects of discriminant analysis are detailed in Mardia *et al.* (1979).

4.3 Non-parametric discriminant analysis

The basic assumption of normality, such as made in Section 4.2.1, is fairly dominant in statistical studies, but can be fairly restrictive. Although many observed variables may be known to follow the distribution, some do not. Non-parametric discriminant analysis enables us to discriminate between two or more naturally occurring groups, without having to make any statement about their distribution.

The term non-parametric is generic and it simply implies that the technique is not parameter-driven in describing the population distribution, and because of that, non-parametric discriminant analysis may also be described as distribution-free discriminant analysis. Non-parametric methods are in quite wide practical use today. Some of the most commonly used methods are the histogram, kernel density estimation, the κ-nearest neighbour and decision trees. In this section we consider the first two methods, with a detailed account of classification trees appearing in Section 4.5.

4.3.1 Kernel density estimation

Kernel density estimation describes a probability density in terms of a functional form containing a smoothing parameter. Given n real-valued observations, we define

the kernel density function as

$$\hat{f}_k(x) = \frac{1}{n_k b_k} \sum_{i=1}^{n} K(\frac{x - x_i}{b}) I(y_i = k) \tag{4.7}$$

where n_k is the number of observations in class k and x_i denotes the data.

The function $K(.)$ is the kernel density function, of which there are several variants. One of the most popular choice of K is the Gaussian, but other kernels such as the Epanechnikov and the uniform kernels are also used. A thorough discussion of the Kernel Density Estimation (KDE) is provided by Silverman (1986). The quantity b_k is a smoothing parameter, also known as the bandwidth; increasing it increases the level of smoothing and the bias but decreases variability.

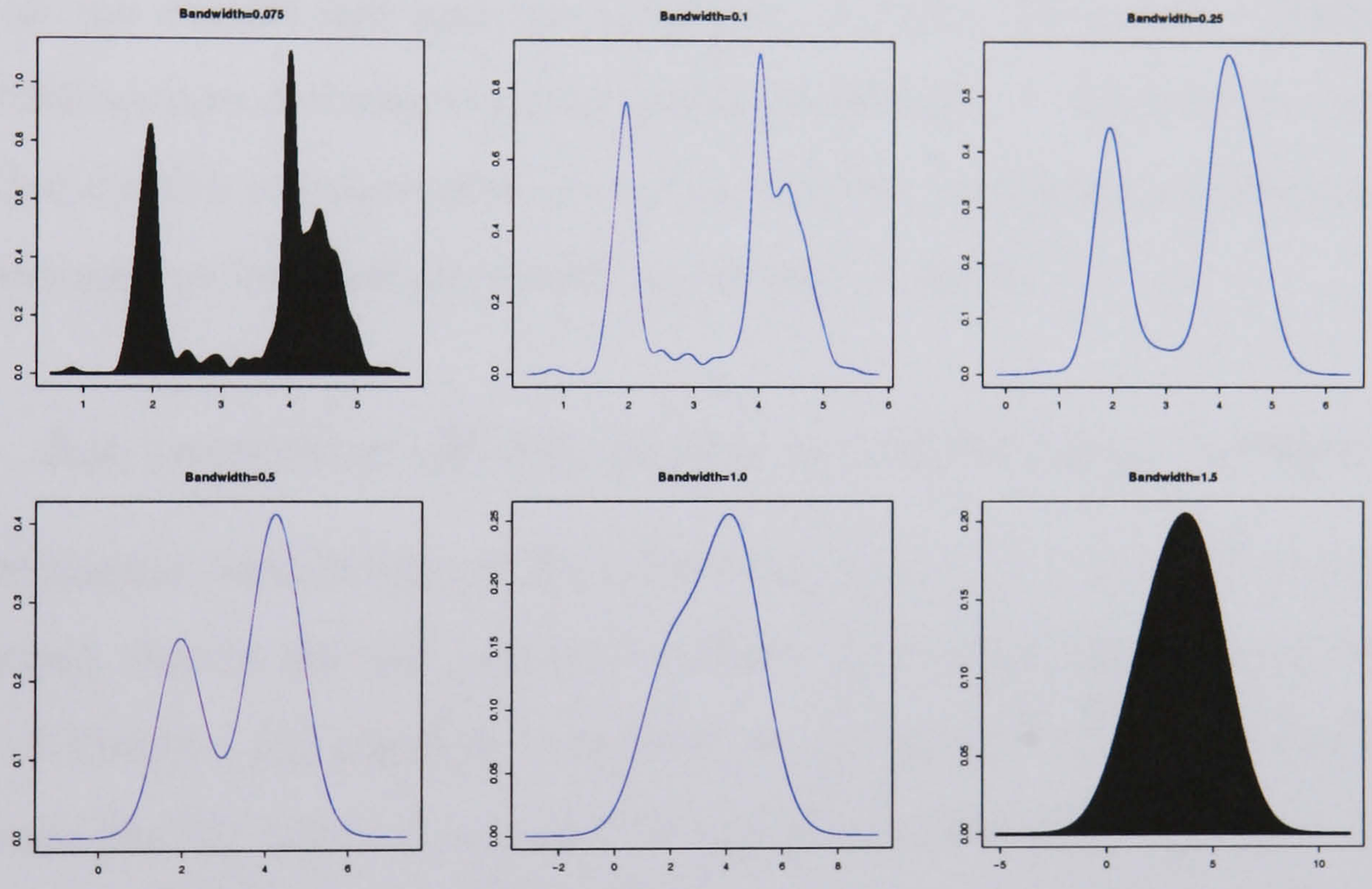


Figure 4.2: *Different kernel estimates of the variable DURATION in the Old Faithful Geyser data presented as a direct function of the bandwidth — increasing from top left to right to bottom left to right — spanning over the bandwidth range of 0.075 through 1.5 of the Gaussian kernel. Swamping and masking of the clusters is noticeable in the two shaded panels, corresponding to the two extreme bandwidths.*

Illustration of kernel density estimation

We illustrate kernel density estimation by using the Old Faithful Geyser data. The OFG throws a tower of thousands of gallons of hot water approximately 60 feet into the air. It remains up for about three or four minutes, before receding. Repeat performances occur on an average of about 30 minutes. Barometric pressure, the

moon, the tides and the earth's tectonic stresses determine the height the geyser shoots and time between eruptions. The geyser data represent recorded duration and waiting time for eruption.

The two plot-rows in Figure 4.2 exhibit the impact of the bandwidth on the kernel estimate for the variable DURATION. Notice how the multi-modal pattern of the variable, as observed under small bandwidth, becomes concealed as the bandwidth increases. The variations in the plots spell a trade-off between an estimate bias and its variability. Generally, large bandwidths tend to cause masking of local density features, while small bandwidths may introduce spurious bumps. The choice of b is of paramount importance in density estimation and, typically, a good choice will depend on the sample size and the variability of data. Silverman (1986) provides some guidelines into choosing the appropriate bandwidth. A data-driven approach to optimal bandwidth selection, also known as variable (sample-point adaptive) kernel density estimation has been proposed by (Hazelton, 2003).

4.3.2 An overview of the κ-nearest neighbour technique

The conventional terminology is "K-nearest neighbour", but we use κ-nearest neighbour instead, due to the fact that the notation K is used elsewhere to denote the number of classes. An important question in the process of density estimation is whether one density function is larger or smaller than the other at each point. Such a question may arise in the case when we are more interested in the classification of the density functions rather than their estimation. Suppose we estimate the density in (4.7) over a given neighbourhood $\mathcal{N}$, where $K(.)$ is uniform over the neighbourhood, containing κ observations. The allocation rule based on the two densities would be equivalent to that based on the proportions κ_k/κ.

It is reasonable to assume that data objects clustered together in the sense of some distance measure would tend to belong to the same class. Thus, if we were to classify a new case it would make sense to weight the evidence of already classified nearby cases most heavily. Effectively, this is what the κ-nearest neighbour classification technique does. The method relies on Euclidean distances and it is an excellent

alternative to the linear discriminant techniques as it works particularly well in classification problems involving irregular decision boundaries.

κ-nearest neighbour density estimation

Given n real-valued observations labelled $i = 1, ..., n$ and κ observations in some neighbourhood, $\mathcal{N}$, we define the κ-nearest neighbour density estimate as

$$\hat{f}_k(x) = \frac{\kappa}{n\Psi(\kappa, n, x)} \tag{4.8}$$

where $\Psi(\kappa, n, x)$ is the sample-dependent volume of $\mathcal{N}$ centred at x and containing exactly κ observations. Using a single 1-nearest neighbour, a unit is classified into the class corresponding to the class of the closest single unit. If the number of observations in each C_k class is large, it makes sense to consider the majority class among the nearest neighbours.

A simple relationship exists between kernel density estimation and the $\kappa - NN$ rule. Under the former, Ψ is fixed and κ is allowed to vary while the latter fixes κ and lets the volume be determined from the data. Since the kernel density estimation uses a fixed bandwidth for all data points, fixing κ and allowing Ψ to vary is particularly important as the optimum choice may then depend on location. To get around this problem, data-driven methods, such as Hazelton (2003), have been proposed.

The κ-nearest neighbour classification rule

The main idea of κ-nearest neighbour classification is that given a focal point x, find κ training points x_r, where $r = 1, ..., \kappa$, closest to x and classify in accordance with the majority votes among the κ-neighbours, with ties broken randomly. Of the κ neighbours, let κ_k represent the number of observations that belong to C_k.

To minimise the probability of misclassification error, the κ-nearest neighbour classification rule should assign each new case to the class C_k for which the proportion κ_k/κ is largest. In a two-class problem, classification of new cases based on the estimated density proceeds as follows. To classify a new case x, its κ-nearest neighbours are found among n samples consisting of $n_1 \in C_1$ and $n_2 \in C_2$, and are stored. If

κ_k are the number of observations in class C_k among the κ-nearest neighbours, then the density estimate in (4.8) transforms into

$$\hat{f}_k(x) = \frac{\kappa_k}{n_k \Psi}, \ k = 1, 2 \tag{4.9}$$

Since the source of both κ_1 and κ_2 is the same, the volume, Ψ is the same for both C_1 and C_2. Thus, in the Bayes' rule (4.1), we replace the $f_k(x)$s by their estimates $\hat{f}_k(x)$s. Intuitively, the rule can also be written as

$$(\frac{n_1}{n})\hat{f}_1(x) \gtrless (\frac{n_2}{n})\hat{f}_2(x) \Rightarrow x \in \begin{cases} C_1 \\ C_2 \end{cases} \tag{4.10}$$

which can be simplified by substituting (4.9) into it, to yield the simpler rule

$$\kappa_1 \gtrless \kappa_2 \Rightarrow x \in \begin{cases} C_1 \\ C_2 \end{cases} \tag{4.11}$$

which allocates new cases to classes on the basis of a majority vote, by simply comparing κ_1 and κ_2 after the initial choice of the κ-nearest neighbours has been made.

More explicitly, the posteriors can be obtained by first recognising that the class conditional densities can be estimated as $p(x|C_k) = \kappa_k/n_k\Psi$ and the class priors as $p(C_k) = n_k/n$, yielding

$$\hat{p}_k(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}. \tag{4.12}$$

The number of neighbours may be chosen by cross-validation, as in the choice of p, b, κ and the tree size. For the κNN rule, one course of action is to choose different values for κ and compare the results. In principle, once a smoothing parameter has been chosen (with a test set or cross-validation), a second test set should be used to get an unbiased assessment of the misclassification probability, although this is seldom done in practice. A discussion of cross-validation is given in Section 4.4.2.

Note that the class conditional densities are very crude for small κ. If $\kappa = 1$ the density is either 0 or 1! A large κ yields reliable estimates and helps maintain the estimated density function relatively constant at the expense of accuracy, unless n is very large. On the other hand, it is desired that the κ-nearest neighbour be very close to the focal point, hence the need for a small κ. The choice of κ is one of

the major issues surrounding the κ-nearest neighbour method. The rule's major drawback is that it requires storage of all samples and comparison of each with unknown sample. Further, as is the case with most rules, missing data and noisy feature variables can adversely affect its performance.

4.4 Assessing the performance of classifiers

Assessment of the performance of classifiers is strongly related to the twin concepts of "learning" and "training", and indeed, derives from them. This section provides a basic understanding of the "learning" process, discusses methods of assessing classifiers and proposes an assessment method based on a notionally infinite test set.

4.4.1 The process of learning for classification

Learning is a fundamental concept in classification and it can be defined along the same lines as we do human learning processes, i.e. in terms of the experience gathered through seeing, doing or reading. The process of classification can be viewed as the partitioning of the measurement space Ω. The "learner" is an algorithm that takes a training set as input and uses it to transform Ω into a partition. A learner differs from a classifier in that the former describes the path from the training data to a partition, while the latter describes the partition itself.

The partitioning of Ω based on the training data yields a misclassification error — i.e., the proportions of misclassified cases over the training set. This error is likely to differ from its counterpart observed over an independently drawn test sample. In both cases the misclassification error is data-dependent. The difference in the two errors may be interpreted to mean that the algorithm failed to properly learn about the given classes. It is therefore natural to seek high accuracy during the learning process, and expect consistent replicability of the results across different samples, which may lead to the problem of "over-fitting".

What is over-fitting?

Although the overall objective of any classification procedure is to work towards error minimisation, care should be taken not to "over-do" it. Over-fitting can be described as a situation under which the learning algorithm adapts so well to a training set that the random disturbances in the set are included in the model as being meaningful. If the algorithm is applied to a different (test) data set, with a different set of noise, it may yield poor results. The problem of over-fitting is most severe in the following settings:

(a) A large p in relation to n in a parametric modelling, e.g. in LDA.

(b) A small b in kernel density estimation. The parameter b determines the level of trade-off between a good fit to the data and a reasonably smooth function.

(c) A small κ in a κ-nearest neighbour model. As does the bandwidth, the parameter κ measures the trade-off between accuracy and the stability.

(d) A large number of nodes under classification trees. As we shall see, in subsequent sections, growing excessively large trees will over-fit the data.

In a more specific definition relating to boosting (discussed in Chapter 5), data over-fitting occurs when the classification algorithm indicates a decreasing error rate in the training set at the time when the error rate in the test set is increasing. In all the above cases of over-fitting, the use of independent data set, for assessment of the performance of the rule, provides protection against over-fitting.

4.4.2 Methods of classifier assessment

The overall purpose of classification is to get as accurate results as possible and in the most consistent way possible. With known densities, the Bayes' rule yields the best results, and, theoretically, we can use it on the population — as that is the best way we would proceed if we could. In practice, we use the following four assessment methods, although, in most applications, we would be limited to the first three, because of the "over-optimistic" nature of the fourth as explained below.

1. Test on the population, i.e. using a notionally infinite test set.

2. Cross-validation.

3. Test data set.

4. Plug-in or re-substitution approach.

The plug-in approach is a typical source of over-fitting, i.e. if predictions are based on the same training data used to produce the rule, there would be a high rate of accurate predictions. The major drawback of this approach is that the model is literally "tested" on the same data used to construct it, which means it will have seen the same examples, which makes the plug-in error estimate over-optimistic.

Cross-validation is a method for estimating generalisation error based on re-sampling results. The resulting error estimates are often used for choosing among various models, such as different bandwidth or different tree size. In a $q-$ fold cross-validation, the data set is divided into q approximately equal subsets. The learning algorithm is then trained q times, each time leaving out one of the subsets but using it to test the algorithm. If q equals the sample size, we have what is called leave-one-out cross-validation. Leave-one-out cross-validation often works well for estimating generalisation error for continuous error functions such as the mean squared error, but it may perform poorly for discontinuous error functions such as the number of misclassified cases. Leave$-v-$out is a more elaborate and expensive version of cross-validation that involves leaving out all possible subsets of size v.

Theoretical and empirical rules

We can distinguish two types of allocation rules, namely, **theoretical** and **empirical** rules. The theoretical, or Bayesian, rule assumes that the priors and densities are known and it is therefore tested on a notionally infinite test data set. We denote by $\xi_{D,\text{pop}}$, the error associated with this rule. The error $\xi_{D,\text{pop}}$ differs from $\xi_{B,\text{pop}}$ in that the former estimates the latter. In practice, however, empirical rules are produced from training data and tested on test data, and, as a rule, they possess some kind of "randomness". Empirical errors associated with the above assessment rules are summarised in Table 4.1. The randomness of the empirical error can arise from two sources, namely, randomness due to the allocation region and randomness

due to the assessment of the rule with random training, test and cross-validation data. We denote the two types of randomness as r_1 and r_2 respectively.

Random Error	**Population**	**Cross-validation**	**Test**	**Training**
Randomness (Learner)	(r_1) $\xi_{D,\text{pop}}$	$(r_1 \& r_2)$ $\xi_{D,\text{cv}}$	$(r_1 \& r_2)$ $\xi_{D,\text{test}}$	$(r_1 \& r_2)$ $\xi_{D,\text{train}}$

Table 4.1: *Data-dependent errors shown alongside their associated randomness. Note that a notionally infinite test data set is assumed for the population error.*

Accuracy and reliability

How much knowledge about the classes the algorithm acquires in the process of learning is an important property as it determines its learning "accuracy" and "reliability". We define accuracy as the rate at which the rule is able to reproduce the true partitions and reliability as the rule's consistency across samples.

The quantity $\xi_{D,\text{pop}}$ is a random estimate of $\xi_{B,\text{pop}}$, which depends on data and assessed on the population. Typically, this error tends to $\xi_{B,\text{pop}}$ as the size of the learning set becomes increasingly large. If the π_ks are known, $\xi_{D,\text{pop}}$ will always be greater or equal to $\xi_{B,\text{pop}}$, i.e. $P\big(\xi_{D,\text{pop}} \geq \xi_{B,\text{pop}}\big) = 1$. This probability is not guaranteed to hold, if both the π_ks and the partitions are data-dependent, as would be the case for $\xi_{D,\text{cv}}$ and $\xi_{D,\text{test}}$. For a given population, the performance of any given algorithm may be measured by the quantity

$$E[\Delta] = E[\xi_{D,\text{pop}}] - \xi_{B,\text{pop}} \tag{4.13}$$

where Δ denotes the discrepancy between the empirical and theoretical errors. A good rule would yield $E[\Delta] \approx 0$, i.e. yields a minimal discrepancy. We can, therefore, measure the reliability of a given algorithm by the quantity $\text{Var}[\Delta]$.

In most real-world applications, only a finite number of learning samples would be available to the learner. To ensure consistency over a number of different samples from the population, we need the generalisation error $\xi_{D,\text{pop}}$, as a population error estimate. This estimate tends to the Bayesian as the training set tends to infinity.

Most of our work addresses issues relating to the accuracy and reliability of classification techniques that use rules based on training data to classify new cases to known classes. We start with a discussion of the classification tree algorithm.

4.5 An overview of classification trees

Classification trees (Breiman *et al.*, 1984) are used to predict the class of a categorical variable in response to one or more predictors. The method recursively partitions a dataset by thresholding a single variable at each step. Partitions are chosen to decrease impurity of the sub-divisions. More precisely, the technique is used to predict membership of cases or objects in the classes of a categorical dependent variable from their measurements on one or more predictor variables. Its main objective is to predict or explain responses on a categorical dependent variable, which puts it in a similar domain with the more traditional methods of Discriminant Analysis, differing mainly in the distributional assumptions made by the latter.

4.5.1 Growing the tree

Growing the tree amounts to sequentially splitting the data into two parts, say, A and B based on a single predictor at each stage. Be it at the beginning of the tree growing process or at a later stage, the data splitting point is referred to as a "node", with the root node denoting the beginning. Let T^* denote the observations in a node at a current level in a tree. We need to split T^* into two parts, A and B. Let the number of observations in T^* belonging to a particular class be denoted by n_k, where $\sum_{k=1}^{K} n_k = n$, denotes the total number of cases at the current node. Detailed examples of tree growing are given in Section 4.6.

The tree growing procedure can be described as follows: Given the choice of a variable, say x_p and the threshold on it, say m, consider splitting the data into two parts A and B, with n_{ik} denoting the number of cases in part $i = A, B$, belonging to one of the classes $k = 1, ...K$ and v some observations at a node, such that

$$\begin{cases} A = & \{v \in T^* : \ x_p \leq m\} \\ B = & \{v \in T^* : \ x_p > m\} \end{cases} \tag{4.14}$$

where A and B are sets of observations on each side of the hyper-plane, $x_p = m$, chosen in such a way that a given measure of impurity, such as the deviance, is minimised. The procedure recursively splits A and B, based on a single predictor at a time. The splits are chosen in an optimal way, based on maximum reduction in the adopted measure of impurity, without attempting to optimise the whole tree. This process goes on until some pre-specified condition is reached. A detailed discussion of the deviance is given by Dobson (1989) and Fahmeir and Tutz (1994). In the following exposition we give a brief description of the method.

		True Classes			
		n_1	n_2	...	n_K
Partitions	n_A	n_{A1}	n_{A2}	...	n_{AK}
	n_B	n_{B1}	n_{B2}	...	n_{BK}

Table 4.2: *The number of observations at the current node belonging to a particular class is denoted by* $n_k,\ k = 1, ..., K$, *while* $\sum_{k=1}^{K} n_{Ak}$ *and* $\sum_{k=1}^{K} n_{Bk}$ *denote the total number of cases, belonging to the appropriate classes, going to* A *and* B *respectively.*

The deviance as a branching criterion

The effectiveness of the classification tree technique derives from the decision made at any particular node. The most popular branching criteria with classification trees are the *deviance*, the *entropy* and the *gini*. As most of our work involving classification trees uses the *deviance*, we address the criterion further. We chose the deviance simply because it happens to be the default choice in the package **R**, a free software from the R Foundation for Statistical Computing — *http://www.r-project.org*.

Under classification trees, the deviance is related to the ordinary sum of squares but it is constructed in a different way. Before the split, the given model involves probabilities p_k in each of the K cells and after the split, it involves the probabilities $p_{ik},\ i = \{A, B\}$ in $2K$ cells. Table 4.2 provides a graphical illustration based on $2K$ cells. Here is how to calculate the deviance. Before splitting, fit $\hat{p}_k = n_k/n$ for the null (no-split) model and define the deviance as

$$D_{\text{Before}} = -2 \sum_{k} n_k \log \hat{p}_k \tag{4.15}$$

Fit two multinomial distributions to A and B respectively, where $\hat{p}_{Ak} = n_{Ak}/n_A$ and $\hat{p}_{Bk} = n_{Bk}/n_B$ for the after-splitting moment. Here, the formulation of the *deviance* uses weights for each example to weight the log-likelihood as follows

$$D_{\text{After}} = \sum_i D_i \text{ where } D_i = -2 \sum_k n_{ik} \log \hat{p}_{ik} \tag{4.16}$$

where $i \in \{A, B\}$, the sum is over the terminal nodes, p_{ik} is the probability distribution and n_{ik} the number of individual cases at the particular node, of class k. Some of the n_{ik}s will be equal to zero. Mathematically, $0 \times \log 0$ is defined as 0. Typically, $(D_{\text{Before}} > D_{\text{After}})$ just because the model after the split is bigger than the one before it and the threshold and the variable are chosen so as to minimise D_{After}. The tree-growing object in **R** contains a display option, by which the vertical lengths of the tree branches are proportional to the decrease in impurity $(D_{\text{Before}} - D_{\text{After}})$.

4.5.2 Pruning the tree and stopping rules

The first step before pruning is to grow a large tree. Initially, all data are at one node, called the root node. For continuous data, the tree can be grown until all the nodes are pure, i.e. yielding a saturated model in which all individuals are correctly classified. In practice, the splitting process continues until the terminal nodes are too pure or too small to be split further. Node purity and data size at a node represent two important stopping rules in tree-growing, although, in principle, one can keep on growing the tree until each leaf (terminal node) has only one observation.

Typically, the threshold of the stopping rule is such that over-optimistic trees are grown. That is, the proportion of correctly classified cases is usually higher than would be experienced with new data from the same mixed population, a problem also known as data over-fitting. Pruning reduces the tree size, and, in so doing, yields a less optimistic, but more robust predictor. The rationale behind tree growing and pruning is the following. If no limit is placed on the number of splits we can end up with a *pure* split with each terminal node containing only one class. On the other hand, growing a very small tree may lead to masking of some of the information in the attributes, thus leading to a higher misclassification error.

It is often tempting to think of stopping growing the tree when there is no substantial

decrease in the adopted measure of impurity. But it may happen that there is only a momentary stabilisation and further splits may yield significant decreases in the measure — hence the need to grow an over-fitted tree and trim it down to size.

Some notation, concepts and basic definitions

We use the following notations and definitions in describing the pruning process. The largest tree is denoted by $f^{(0)} = F$, an arbitrary tree is denoted by f and j denotes an arbitrary pre-terminal node in f, as graphically illustrated by the LHS panel of Figure 4.3. The tree branch $f_j \in f$ has a root node $j \in f$ and consists of the node j and all sub-nodes below it, as shown in the middle panel of Figure 4.3. We denote the measure of impurity for an arbitrary tree by $R(f)$ and $R(j)$ denotes the measure of impurity for an arbitrary pre-terminal node. The measure $R(.)$ is defined as the sum of all measures of impurity at the terminal nodes. Here the terminal nodes are relative — they may refer to a tree or a sub-tree of an arbitrary tree.

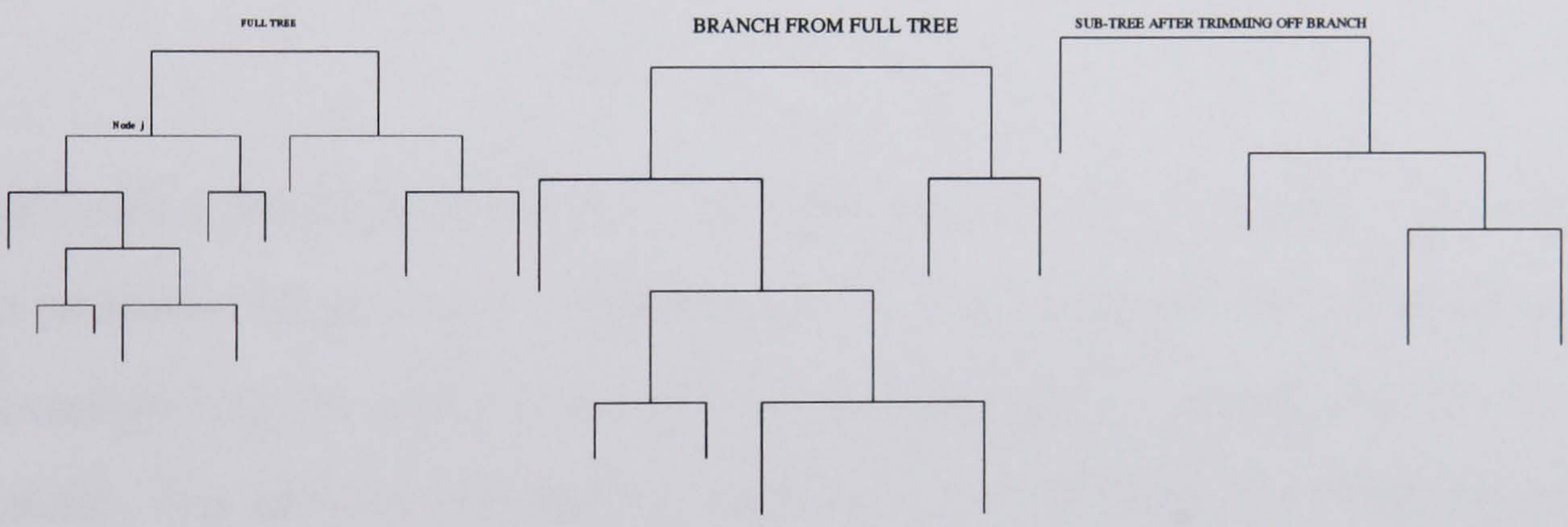


Figure 4.3: *The three panels, left to right, represent the full tree, a branch of the tree starting at node j and a sub-tree, i.e. what is left after the branch is trimmed off.*

For a very large tree there will be a large number of sub-trees. Searching for the best tree through the complete list of sub-trees would be computationally expensive. Exhaustive search can be avoided by selecting only the "best" sub-trees. The main idea here is to try and compare different sub-trees of the same size.

The pruning process and Breiman's cost-complexity measure

Pruning a tree branch f_j from f deletes from f all the sub-nodes of f_j except j itself. This process, denoted by $f - f_j = f^*$, yields the sub-tree on the RHS panel of Figure 4.3. Any sub-tree $f^* \in f$ obtained by successively trimming off tree branches in this way is called a pruned sub-tree of F. Note that f^* is the best sub-tree of a particular

size. Thus we can define the nested sub-trees $f^{(1)}, f^{(2)}, ..., \{t\}$, where $\{t\}$ is the tree stump, obtained by pruning upwards from F, as the best sub-trees of decreasing sizes.

The pruning process starts with the largest tree, F, computes $R(j)$ for each $j \in F$ and progressively trims off F branches in the bottom-up direction, such that at each stage $R(f - f_j)$ is as small as possible. In other words, if f is the current sub-tree, the next iteration chooses $f - f_j$, with the pre-terminal node j chosen from amongst the nodes of f such that $R(f - f_j)$ is minimised. The remaining question is which sub-tree is optimal. One popular pruning method is Breiman's cost-complexity criterion.

The main idea behind Breiman's automated cost-complexity measure is the following. Let the complexity of the sub-tree $f \in F$ be defined by its total number of terminal nodes, L_f. Further, let $0 \leq \alpha < \infty$ denote a real number, called the cost-complexity parameter. The cost-complexity measure can then be defined as

$$R_\alpha(f) = R(f) + \alpha L_f \tag{4.17}$$

Breiman's cost-complexity measure considers not only the minimisation of impurity, but also penalises large trees. Note that (4.17) can be viewed as a linear function of the tree complexity, in which α is the rate at which $R_\alpha(f)$ changes as the tree grows by one node. For each α, we want to find the sub-tree $f^\alpha \in F$, which minimises

$$R_\alpha(f^\alpha) = \min_{f \in F} R_\alpha(f) \tag{4.18}$$

The mechanics of pruning involving the cost-complexity criterion can be illustrated as follows. Let f_j be any branch of the sub-tree $f^{(1)}$, and define $R(f_j)$ as

$$R(f_j) = \sum_{j^* \in L_{s.f_j}} R(j^*) \tag{4.19}$$

where $L_{s.f_j}$ represents the set of all terminal nodes in f_j. Breiman *et al.* (1984) show that for j any non-terminal node in $f^{(1)}$, $R(j) > R(f_j)$ holds. For any non-terminal node $j \in f_1$, denote by $\{t\}$ the sub-branch of f_j consisting of the single node $\{t\}$. Set $R_\alpha(\{t\}) = R(j) + \alpha$, where, typically, $R_\alpha(\{t\}) \gg R(j)$. Now, for any f_j, define the measure of impurity as a function of α by

$$R_\alpha(f_j) = R(f_j) + \alpha L_{f_j}. \tag{4.20}$$

As long as $R_\alpha(f_j) < R(\{t\})$, the branch f_j has a smaller cost-complexity than the single node, $\{t\}$. But at some value of α the two cost-complexities become equal, then $\{t\}$ is smaller than f_j and because it has the same cost-complexity, it is preferable. The process to get $f^{(2)}$ from $f^{(1)}$, $f^{(3)}$ from $f^{(2)}$ and so on, recursively trims off the most "not so important" nodes in $f^{(1)}$, $f^{(2)}$ and so on respectively, yielding a decreasing sequence of sub-trees. Examples of tree pruning are given in Section 4.6.

The choice of alpha and best tree

Although α is varied between 0 and ∞, there are only a finite number of αs for which the sub-trees are distinguishable. Let those values be $0 = \alpha_1 < \alpha_2 < ... < \alpha_i$, where i is less or equal to the number of tree leaves. For $\alpha_i \leq \alpha < \alpha_{i+1}$, the sub-tree f^α is the smallest subset of the largest tree. Increasing α penalises large trees, i.e. when $\alpha = 0$, $f^\alpha = F$ and as $\alpha \to \infty$, f^α tends to the root node. The best tree, with respect to the criterion, can be chosen from the foregoing sequence of sub-trees.

The aim is to estimate the expectation of the mean of the impurity measure over the sub-trees and choose the estimate giving the smallest measure of impurity. This can be done through cross-validation. Note that the choice of the best tree involves choosing the complexity parameter α. The cost-complexity algorithm depends on the parameter α. An appropriate choice of α is therefore necessary. One popular choice of α is based on Akaike's information criterion (Sakamoto *et al.*, 1986), which is defined as $\alpha = 2(K - 1)$, where K is the number of classes.

Another method for choosing α is by cross-validation, whereby prediction is made on a test set, then the deviance is computed versus α for the pruned trees. We can then choose the smallest tree with an approximately minimum deviance. There are various ways of generating the cross-validation data. In our examples we adopt a simple and fast approach by randomly extracting it from the training set in a pre-specified proportion. For instance, from 150 observations we randomly set aside 50 to be used as a test set and use the rest as training dataset.

Summary of the phases

The classification tree methodology can be viewed as a three-phase process. In the first phase, a tree over-fitting the data is grown, then branches are trimmed off the grown tree and finally, the best sub-tree that estimates the true model as well as possible is selected from the resulting sequence of sub-trees. In between the first and last phases lie the criterion of predictive accuracy, the decision to split, the decision to stop splitting and the choice of the right size tree.

4.6 Illustrations of classification trees

In this section, we demonstrate the tree classification methodology based on two main examples. The first is based on Fisher's iris data of 150 observations equally divided into three species — setosa, versicolor and virginica. There are originally four variables — Sepal.Length, Sepal.Width, Petal.Length and Petal.Width. In Section 4.6.1 we introduce an example based on a slightly modified iris data — a simplified example with well separated classes. Section 4.6.2 presents a more complex example based on the Painters data — a small data set with multiple classes. Both datasets are pretty standard and embedded in the statistical application package **R**. In both cases, we consider growing, pruning and choosing the optimal tree.

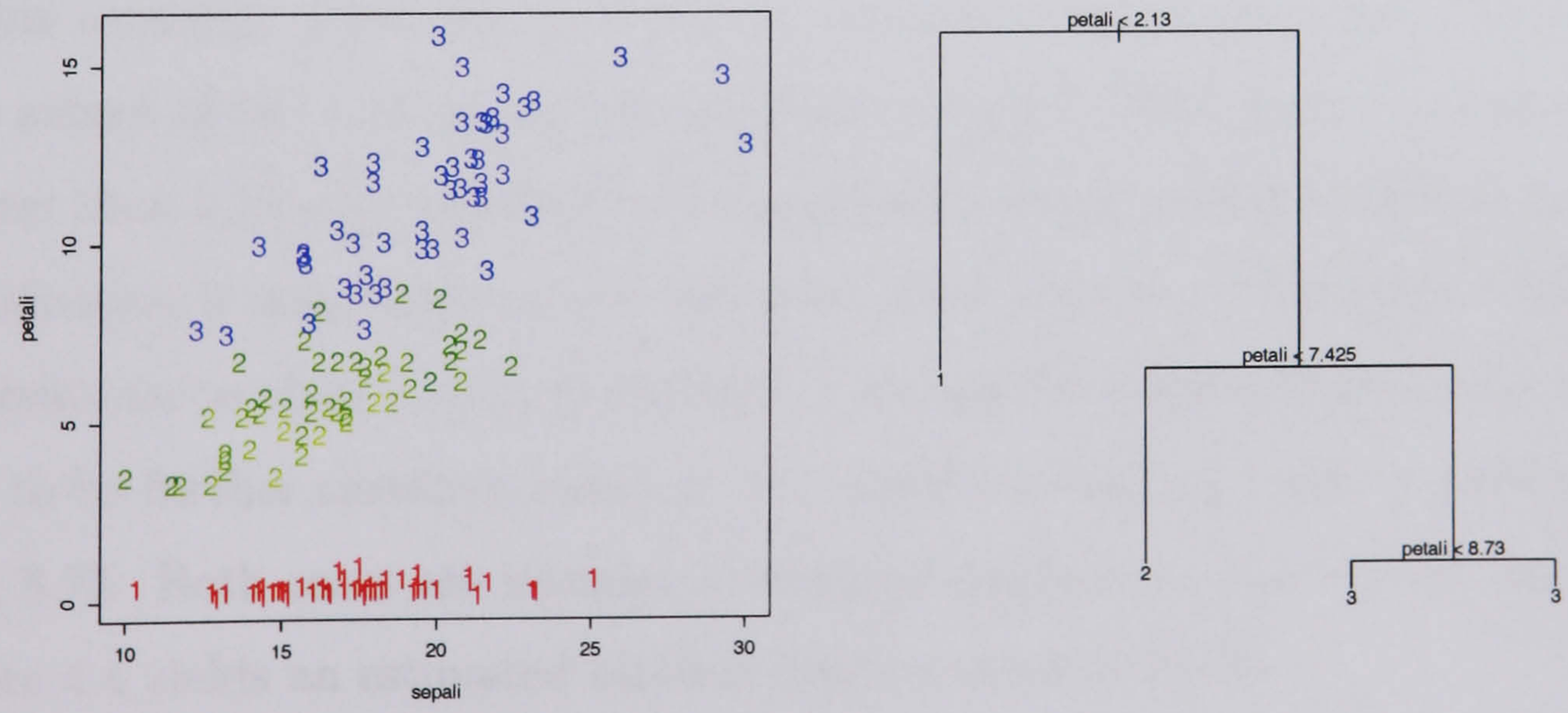


Figure 4.4: *The three-class problem in a pictorial form, on the LHS, and the resulting tree with an estimated misclassification error of 2.67%, on the RHS. The vertical lines of the tree plot are proportional to the reduction in deviance. There is a minimal reduction of deviance in the final split, even though all cases go to the same class.*

There are two important features to note from the RHS side plot in Figure 4.4. The

first is that the lowest two leaves are both of the same class, implying that deviance reduction can still occur under such circumstances. The second feature is that the tree is grown on only one of the two variables. Generally, this phenomenon may be a consequence of either the unused variable having only a minimal association with class membership or its effect being masked by other variables. The latter is particularly crucial as small variations in either the training set, class priors or both may juggle the split from one x_p to another.

4.6.1 Examples based on the modified iris data

In this section we introduce two examples based on a slightly modified Fisher's iris data set of size $n = 150$ observations. In one example, we generate two new variables, *sepali* and *petali* by multiplying the length and width of the sepals and petals respectively. In the other example, we simply discard one of the classes. The data and the resulting full-grown tree, for the first example, are presented in Figure 4.4, on the LHS and RHS respectively. The tree is grown on 150 observations from three different species of 50 observations each, thus giving an equal probability of 1/3 for each of the species, which in this case holds by design and forms the *a priori* class probabilities with which the tree growing process starts. These initial values reflect the process of fitting one multinomial class to all data.

In this example, when the independent variable *petali* is less than 2.13 we get a pure subset of 50 observations all classified to class 1. This leaves the case of *petali* greater than 2.13 with 100 impure observations needing further classification. Their classification is then based on the condition *petali* less than 7.425 which classifies 46 observations to class 2 with probability 1 leaving 54 observations for the opposite case to be further classified based on the condition whether *petali* is greater or less than 8.73. Both cases are ultimately classified to class 3. The tree on the RHS of Figure 4.4 yields an estimated misclassification error of 2.67%.

Notice that the tree in Figure 4.4 is grown based on information provided by the two independent variables although the actual growing of the tree is based exclusively on only one of the variables, *petali*, implying that the algorithm does not find the variable *sepali* important in performing the splits. It is important to emphasise the

behaviour of the algorithm in this case, as it creates the illusion that there is only one useful independent variable. Typically, a tree may be grown only on $v - w$ variables even when v variables are available. This would be fine in as far as growing the tree is concerned and can simply be interpreted to mean that the other w variables are not important in growing the tree.

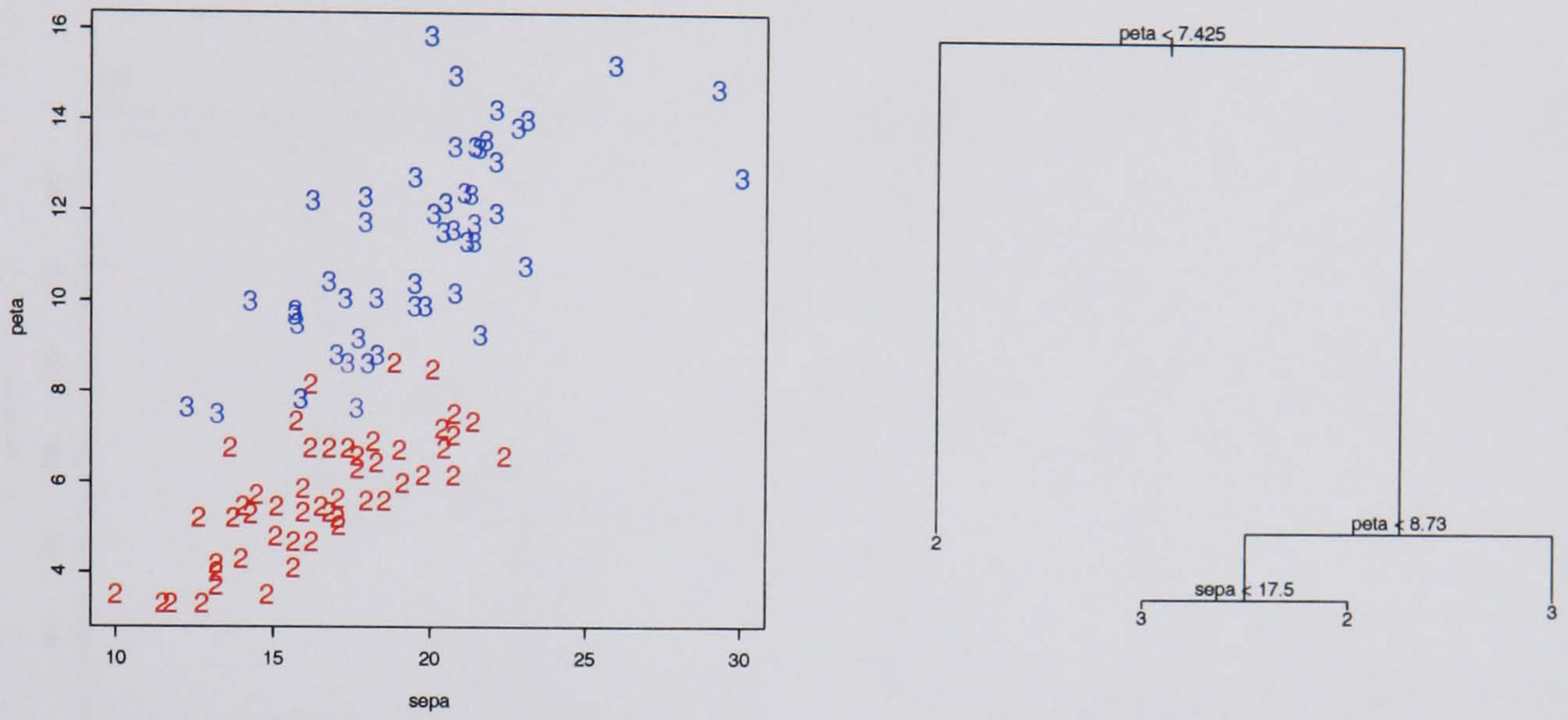


Figure 4.5: *The LHS panel gives the pictorial form of the data after discarding one of the classes and the RHS displays the resulting tree. Note that the first two splits are based on the same predictor and threshold, as in Figure 4.4, but the second predictor is used for the last split. The estimated misclassification error is 3%.*

In this example, we throw away one of the classes and rename the predictors as *sepa* and *peta*, with the sample size cut down to $n = 100$. Figure 4.5 shows the modified data on the LHS and the resulting tree on the RHS. Note that the first two splits are based on the same variable and thresholds as in Figure 4.4, but the last split is based on the other predictor. The estimated misclassification error is 3%.

Tree pruning and optimal size choice by the cost-complexity method

For pruning, we adopt Breiman's cost-complexity criterion as implemented in **S-Plus/R**. The pruning command takes in several arguments including the parameter k for the α. The cost-complexity parameter k defines either a specific subtree of, if given as a scalar or denotes a sequence of subtrees minimising the cost-complexity measure. If not specified, the pruning algorithm determines it algorithmically.

Logically, the final split for the tree in Figure 4.4 is redundant — the resulting sub-tree is therefore a "good candidate" for pruning. The LHS and RHS panels in

Figure 4.6 provide the pruning summary for the two trees in Figure 4.4 and Figure 4.5 respectively. In both cases, pruning uses the deviance as the measure of impurity with α determined algorithmically. Both panels in Figure 4.6 show that the lowest deviance is obtained for tree size 4. For the same tree sizes, stabilisation of the deviance on the RHS occurs at a much lower deviance than on the LHS. In both cases a tree of size 3 appears to be good enough, with about the same misclassification error.

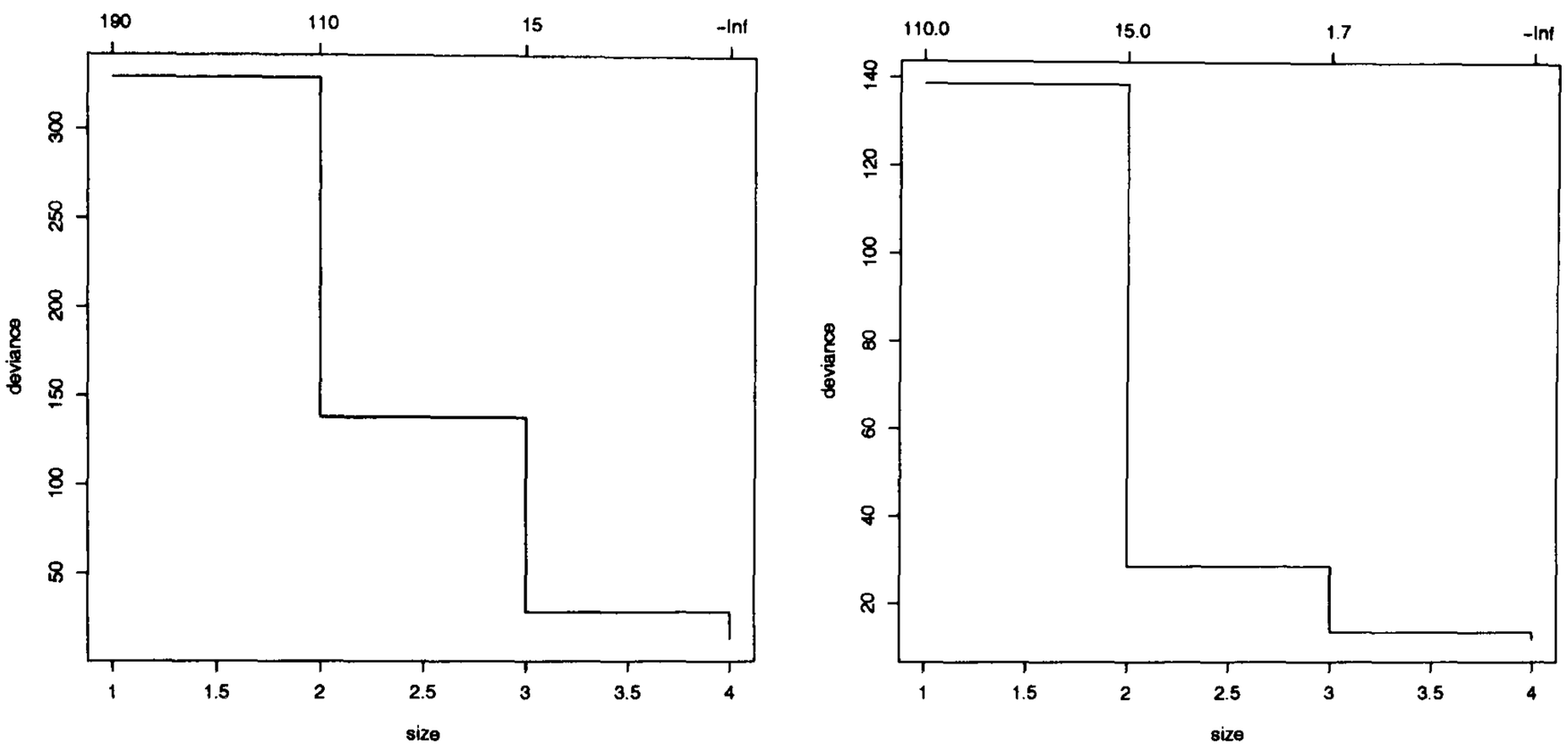


Figure 4.6: *The LHS and RHS panels correspond to the trees in Figure 4.4 and Figure 4.5 respectively. They provide pruning sequences based on Breiman's cost-complexity measure. Plausible trees are attained more efficiently on the RHS.*

Choosing the best tree by cross-validating

The optimal tree size can be determined by cross-validation. The method is also driven by the trade-off between the adopted measure of impurity and tree size. In **R**, the cross-validation command takes in a number of arguments, including the default $K = 10$, which specifies the number of folds of the cross validation and is based on the assumption that the training set is randomly split into 10 equal parts of which 9 are used to grow the tree and 1 to test the validity of the model. A 10-fold cross-validation simply means that 10 trees have to be grown and averaged. The averaging is done over 10 trees for fixed α, as the tree size is allowed to vary.

For both data sets in Figures 4.4 and 4.5, we cross-validate on the basis of $K = 3, 10, 15$ and 30 so that we use 2, 9, 14 and 29 observations to grow the tree and cross-validate by the remaining. Cross-validation outputs are given in Figure 4.7.

The top and bottom row panels correspond to Figure 4.4 and Figure 4.5 respectively. There appears to be a general consensus on tree size 3 as being optimal in both cases.

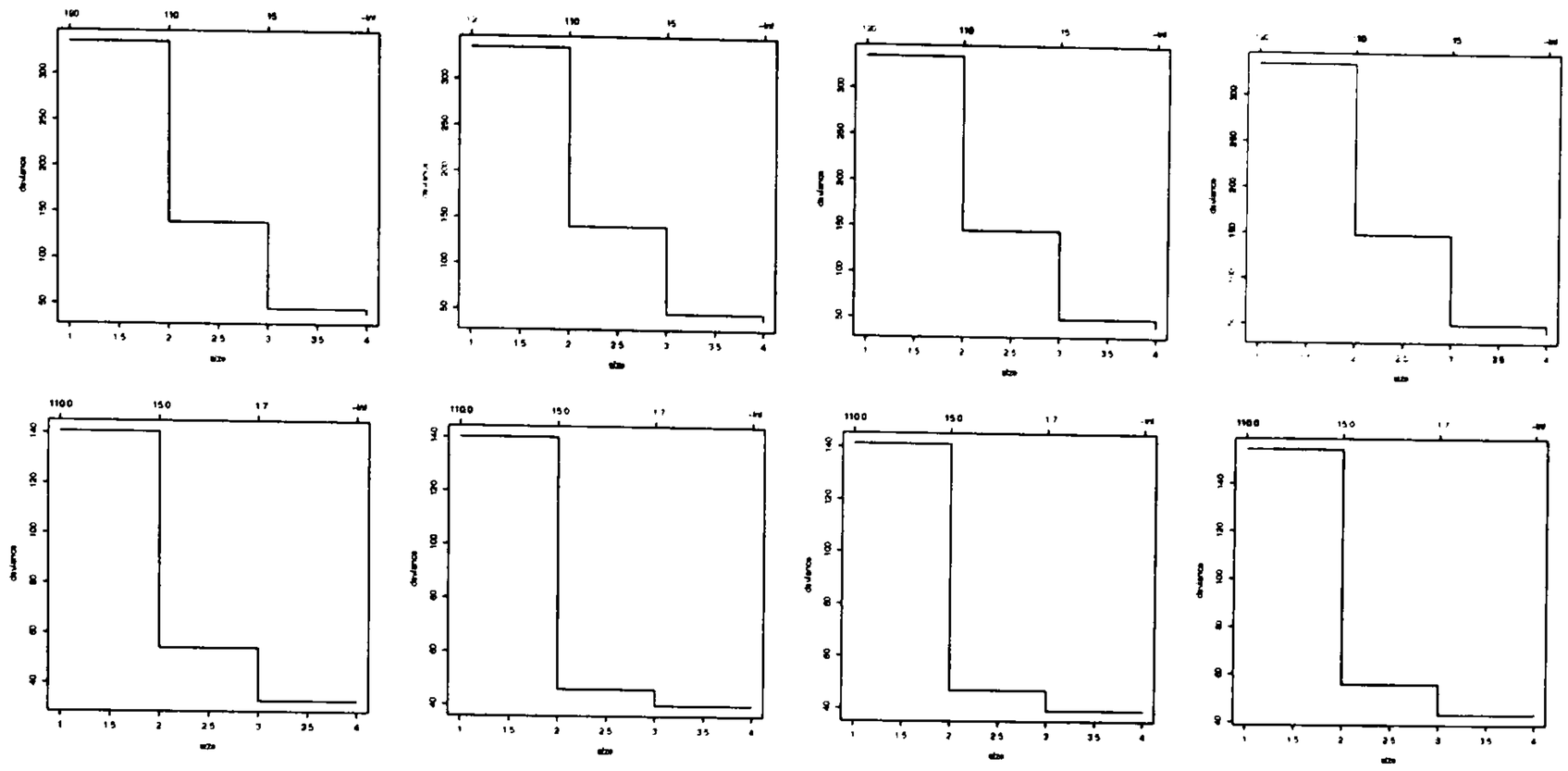


Figure 4.7: *The top and bottom row panels correspond to Figure 4.4 and Figure 4.5 respectively. From left to to right the panels represent 3-fold, 10-fold, 15-fold and 30-fold cross-validation patterns respectively. Size 3 appears unambiguously optimal.*

4.6.2 Examples based on more complex data

The two examples in Section 4.6.1 represent a simplified pattern with clearly demarcated classes. Applying the tree methodology to the data yields unambiguous results. We soon get all the cases classified into one of the three species. In this section we use the **R**-embedded Painters data set of 54 observations with 8 classes, as a typical example of complex data. The data set represents de Piles' scores of artists based on four variables, Composition, Drawing, Colour and Expression. A painter is classified into one of the eight classes, A through H.

Figure 4.8 represents the tree built on all four predictors of the Painters data. Looking at the tree it is possible to tell that there is a large overlap of classes. Note that class F has disappeared from the scene and also how the variable Composition appears at different levels of the tree and the appearance of the same classes, e.g. A and D on either side of the tree. Note also that the total number of observations is quite small in comparison with the total number of classes involved.

The misclassification error in Figure 4.8 is 40.74%. Clearly, growing the tree on all four predictors produces a rather balanced tree with nine terminal nodes. However, none of the terminals is pure and partitioning has not been evident here because of

the data structure. Tree growth has to stop here as the procedure cannot make it any better according to the adopted criterion.

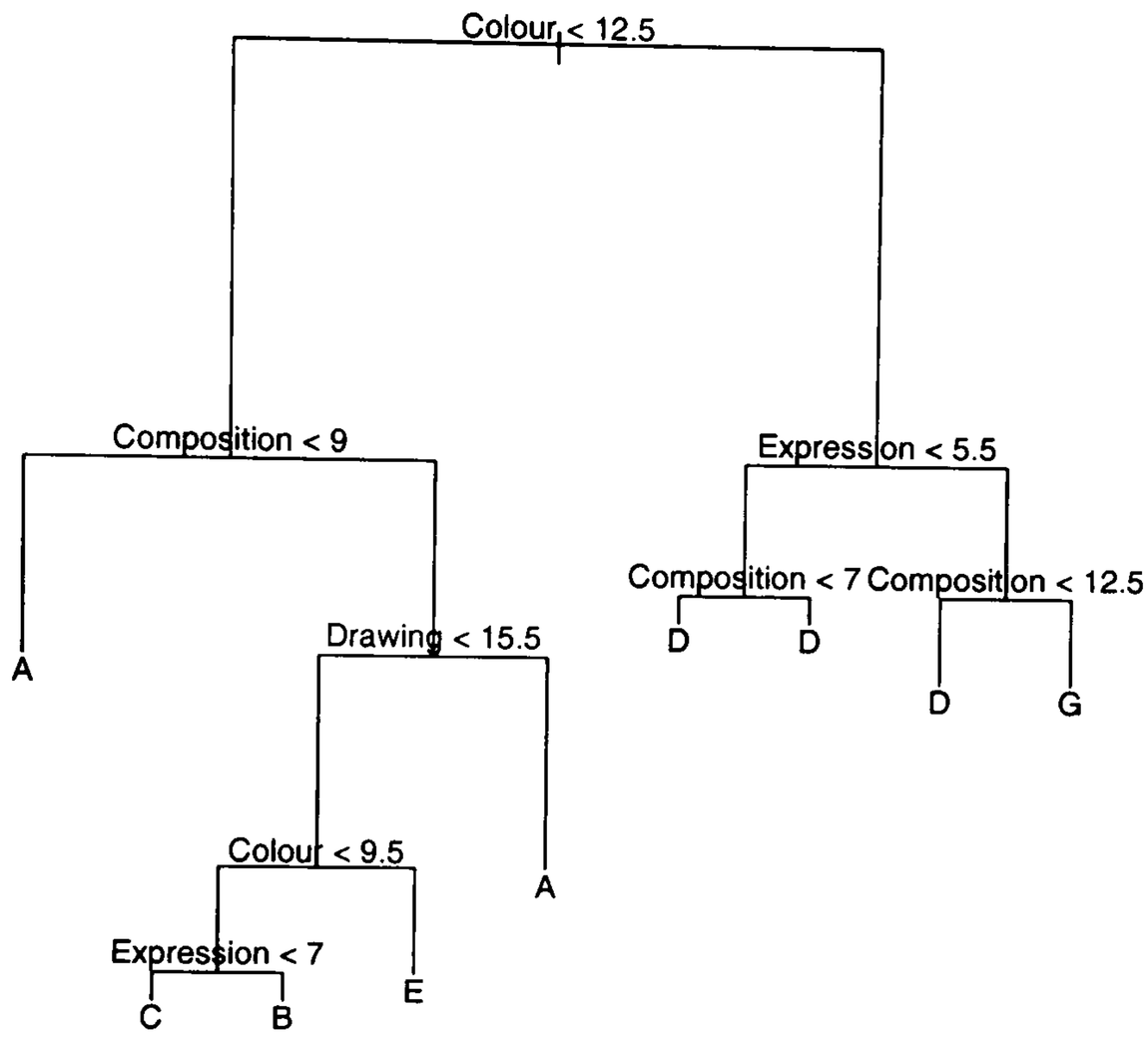


Figure 4.8: *The tree was grown on the Painters data using all four predictors. Although the tree looks rather balanced, none of the terminal nodes is pure and the estimated misclassification error stands high, at 40.74% — apparently a direct consequence of the sample size being too small and the number of classes too large.*

In two arbitrary different combinations of the four predictors we grew the tree again and performed new partitions as exhibited in Figure 4.9. Column-wise, the panels correspond to the partitions and grown trees based on the predictors as labelled. The estimated misclassification error in both cases is over 50%! To see how atrocious the classification results for the Painters data are, we can just compare the summaries with those obtained under our iris data example.

The misclassification error under the iris case was 0.02667 in which we had only 4 out of 150 observations misclassified. Compare this with the Painters data case in which we have 22 out of 54 observations misclassified when all predictors are used and more than 50% observations going the other way when 2 predictors are used! This undesirable result is a consequence of small n and too many classes.

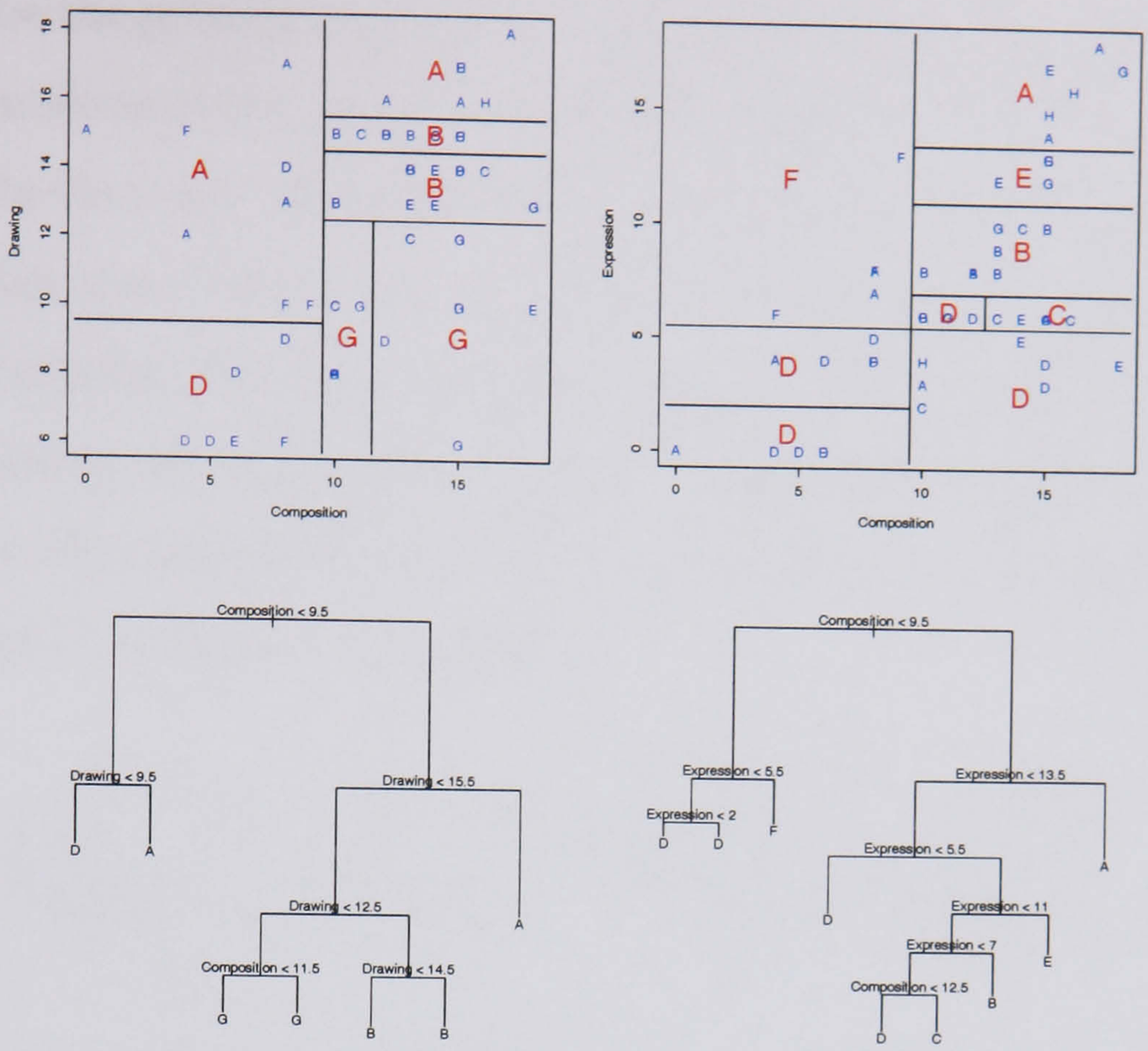


Figure 4.9: *The LHS tree was grown on the two predictors, Composition and Drawing and yielded an estimated error slightly over 50%! The RHS column corresponds to the predictors Composition and Expression and yielded about the same estimated error. Clear partitioning is unattainable as the data follow a rather irregular pattern.*

Choosing the optimal size for the tree

Pruning based on error rate can prove to be effective, especially in cases of massive overlaps as in the case of the painters data. This method is rather appealing because of its simplicity and amenability to overlapping classes. But although the purpose of pruning is to select the "best tree", consequences of pruning may at times be unconvincing, mainly because of the inherent randomisation in splitting the data into K roughly equal parts. The process of randomisation is likely to generate different subsets and possibly different cross-validation patterns. Ideally, the patterns should not be very different otherwise the plausibility of cross-validation would be questionable. To clear the mystery we produce a series of cross-validation figures at different values of K and look at the average pattern.

The first and the second rows of Figure 4.10 are deviance-based on 9 and 27-fold cross-validation respectively. The last two rows are based on the misclassification method. Still there does not seem to be a consensus between the two methods as to

what should be the pruning level for the tree. An 18-fold cross-validation revealed a pronounced masking under deviance, suggesting, almost categorically, 2 to 3 leaves as optimal. On the other side, the misclassification method seemed to perform quite well, suggesting more than 7 leaves. The lack of consensus is attributable to the massive class-overlap, the large number of classes and the small n. This draws our attention to issues relating to how the performance of a classifying algorithm can be affected by data aspects like size and class structure. We address similar issues in the subsequent chapters of the thesis.

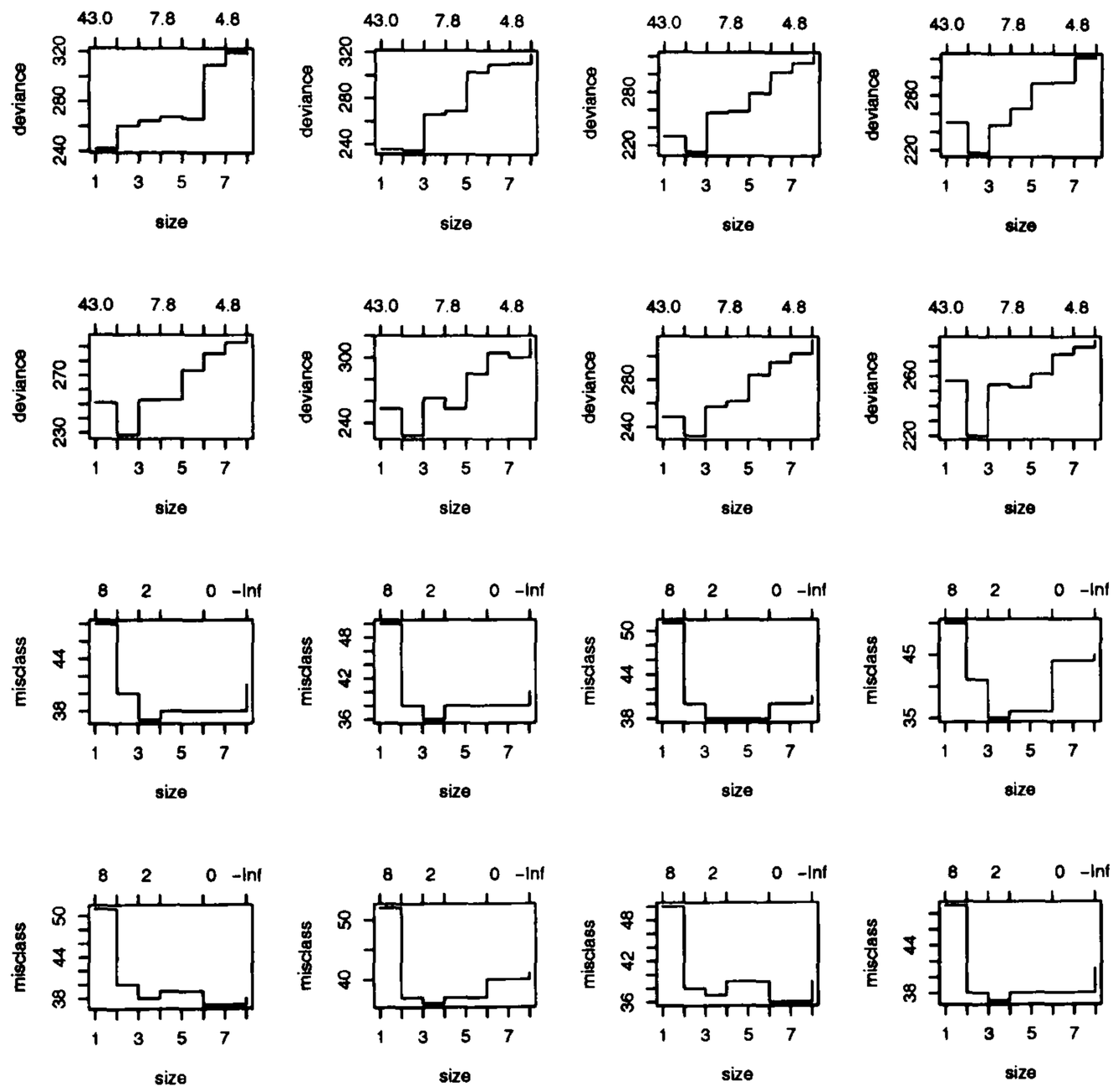


Figure 4.10: *9 and 27-fold cross-validation results based on the painters data. The first two rows are deviance-based and the last two are misclassification-based. The two are alternative methods in* **R***'s cross-validation object. Note that each of the columns corresponds to a different cross-validation. We selected the four plots for each of the two methods simply to ensure clear visibility of the plots.*

4.7 Concluding remarks

In this chapter we introduced the classical classification techniques and provide a detailed discussion of the concepts of classifier and data-based learning and training. We consider the issue of the different types of randomness to be particularly crucial because it implicitly raises alarm against the approach of using a single test data

set to test a rule, mainly because of the inherent randomness in choosing the dataset.

Another issue in this chapter is software-related. The tree-growing object in **R**, **tree(.)**, is designed to accommodate case weights between 0 and 1. However, we uncovered that the embedded conditional construct, makes it impossible to force a split at a node if the weights sum to less than 1. To overcome this problem, we can re-scale the weights to ensure that they sum to n. This can be implemented either internally by modifying the **tree(.)** object or externally by making the appropriate weight adjustments each time the weights are defined.

We saw that data structure can be crucial in attaining accurate predictions and with convenient data structure, such as the modified iris data, the algorithm performed exceptionally well. However, a high misclassification error was observed with a small n and a large number of classes, as was the case with the Painters data. The challenge is to minimise the error. In practice, choosing an optimal tree may often prove to be a challenging task. Further, questions may be raised as to how the decision tree methodology, that assumes the data have an inherent rectangular layout, would cope with an $X-$ shape data, ring-shape data or data with a sine-wave boundary, which present a direct challenge to linear discrimination. Such data types may be challenging not only to the methodology's vertical-horizontal rule, but also to the trade-off between over-fitting and high misclassification.

A well performing method should be as robust as possible. In particular, its performance should be assessed on the basis of handling complex, sometimes rare and unexpected, data structures. It is therefore appealing to envision a refined version of the method that would avoid large error margins as well as excessively large trees. In Chapter 5 we introduce an enhancement tool for classifiers called *boosting* and use it to "boost" the performance of tree stumps on data with a curved boundary.

Chapter 5

Classification by boosting

5.1 Introduction

Our general concern in Chapter 4 was that minimising misclassification error on the training data may lead to over-fitting, as the learning algorithm adapts too well to the data. It also emerged that most classification rules use all data points equally no matter how close to a decision boundary — which may lead to masking or swamping effects. There is therefore a strong need to formulate a set of rules to cope with ambiguous data. For instance, a tree treats all observations equally — a plausible approach could be to down-weight in accordance with a criterion based on each observation's location. This chapter addresses some of these issues.

In general, having trained the algorithm on a data set and tested it on an independent set we want a generalised performance on future data sets to be consistently accurate across different data sets. The first idea that comes to mind is to work out a procedure that averages all sample-based rules over the population. In part this is what the boosting technique we introduce in this chapter attempts to emulate, except that its averaging is by re-weighting the observations of the same training set, thus creating notional multiple data sets. The monograph of Hastie *et al.* (2001) provides an introduction to the boosting algorithm.

Boosting (Schapire, 1990) is an iterative classification enhancement tool which yields refined predictions by combining rough and moderately accurate classifiers, code-named "weak learners", such as tree stumps. The method seeks to transform "weak" into "strong" learners and has been praised for being both accurate and over-fitting

resistant (Freund and Schapire, 1997; Friedman *et al.*, 2000). If the "weak learner" is a tree stump, the boosted classifier is a result of many stumps which vote by weighted majority. Each stump is learned from a weighted set of data, with the weights being derived from the results of the previous iteration. Boosting has been shown to work better with tree stumps than with most classifiers (Hastie *et al.*, 2001). Using trees with boosting may also provide an insight into resolving some of the issues relating to the optimal choice of tree size as discussed in Section 4.6. In this Chapter we apply the method on data with a complex decision boundary.

The basic ideas of boosting originally appeared in the Computational Learning Theory literature through works of Schapire (1990), Freund (1995) and Freund and Schapire (1997). However, the method has its roots in the "Probably Approximately Correct" (PAC) model in Machine Learning, pioneered by Valiant (1984). He was the first to pose the question of whether a rough and moderately accurate ("weak") learning algorithm could be "boosted" into a strongly accurate algorithm, marking the beginning of what came to be known as "weak/strong" learnability.

Our examples are based on simulated data with a sine-wave boundary — chosen because it provides a natural challenge to linear methods of discrimination. Similar to Schapire (1990), we will use 2 classes, labelled ± 1. Extensions to multi-class cases are straightforward as detailed by Schapire and Singer (1999). We mainly boost tree stumps, but we also consider larger trees of various sizes.

This chapter is organised as follows. Section 5.2 provides an overview of the transformation of "weak" into "strong" learners and presents our interpretation of the concepts of "weak" and "strong" learnability. Section 5.3 introduces the mechanics of the **Discrete AdaBoost** algorithm. Section 5.4 provides boosting illustrations based on a noise-free simulated data set. Section 5.5 thoroughly investigates the role of observational weights in boosting and Section 5.6 provides concluding remarks.

5.2 Transforming "weak" into "strong" learners

A weak learner is such that no matter how much data is available, the Bayes' error rate may never be achieved, although the learner will perform better than a random guess. Conversely, a "strong learner" should obtain error rates close to $\xi_{B,\text{pop}}$ with very high probability. The requirement that we attain such high accuracy most of the time is rather stringent. This section considers the transformation of "weak" into "strong" learners and provides a statistical interpretation of "weak learnability".

5.2.1 The conventional weak learnability

The "weak" learning model of Schapire (1990) differed from the basic PAC model (Valiant, 1984) in that it needed only achieve a moderate error rate and confidence. This learner could then be "boosted" into a relatively high accuracy learner, by allowing the algorithm to make multiple accesses to different versions of the same training set, thus emulating the notion of multiple samples.

The major property of "strong" learning is the ability to recover the true partitions in the population. For a sufficiently large n, a strong learner comes arbitrarily close to the Bayes' error. In other words, for any $0 < \epsilon$ and $\delta < 1$, we can get a large enough sample, n, for which

$$P(\xi_{B,\text{pop}} < \xi_{D,\text{pop}} < \xi_{B,\text{pop}} + \epsilon) > 1 - \delta, \tag{5.1}$$

where $\xi_{B,\text{pop}}$ is the Bayes error and $\xi_{D,\text{pop}}$ is the population-based error estimate. On the other hand, a "weak" learner is required to perform just better than a random guess — it may be defined as $P(\xi_{D,\text{pop}} < \frac{1}{2}) > \frac{1}{2}$, provided n is sufficiently large. The probability holds in most cases, with exceptions such as linear discriminant rules or cases of complex data structures such as the "chess-board" type. In this sense, the LDA does not ever satisfy "weak" let alone "strong" learnability

In the boosting context, transformation of a "weak" into a "strong" learner can be described as follows. A different distribution is recursively passed to the weak learner, with an objective of enhancing the chances of correct classification for the ambiguous cases. Enhancement is due to the fact that the weak learner generates

new classifiers with fewer mistakes on these cases. One of the first versions of boosting was referred to as "boosting by majority" (Freund, 1995).

Under boosting by majority, an initial classifier ϕ_1 is devised based on the first sample of n_1 training points. A new sample of n_2 points, half of which have been misclassified by ϕ_1, is used to train the next classifier, ϕ_2. The third classifier ϕ_3 is trained on a new set of points n_3 for which the first and second classifiers disagree, i.e. $\phi_1 \neq \phi_2$. The boosted classifier is defined as $\phi_b = \text{majority vote}\{\phi_1, \phi_2, \phi_3\}$, thus accounting for its name. The technique requires some knowledge of the performance of the classifier in order to allocate the training set appropriate to the n_i. The problem is that, in practice, the required information is often unavailable and on top of that it is likely to vary with the distributions. This problem led to the formulation of a modified version of boosting, namely **AdaBoost**.

5.2.2 A simple insight into weak learnability

Consider the two sets of partitions, $\mathcal{C}$ and Φ, such that $\mathcal{C} \subseteq \Phi$, where $\mathcal{C}$ is the set of true partitions and Φ is the set of all partitions of Ω. We can define a new restricted set of partitions $\Phi' = \overline{\mathcal{C}}$, such that $\Phi = \Phi' \cup \mathcal{C}$. The sets Φ and Φ' may be referred to as strong and weak classes of partitions respectively. We illustrate, by example, that under certain learning conditions, the two sets yield approximately the same results — i.e, an algorithm trained only on Φ' may recover the true partitions.

Emphasising population behaviour of various classification rules is analogous to working with a large training set — that is, letting $n \to \infty$. Suppose a variable assumes three discrete values in $\Omega = \{A, B, C\}$ comprising 3 elements and there are 2 classes with prior probabilities π_1 and π_2 and class probabilities $\{p_{1a}, p_{1b}, p_{1c}\}$ and $\{p_{2a}, p_{2b}, p_{2c}\}$ where $\sum_{k=1}^{2} \pi_k = 1$ and $p_{ka} + p_{kb} + p_{kc} = 1$, for $k = 1, 2$ and $p_{k[a,b,c]}$ is the probability of observing the appropriate element, given that we are in class k. Consequently, the 2^3 – element set of all possible partitions is

$$\Phi = \{A|BC, BC|A, C|AB, AB|C, ABC|\emptyset, \emptyset|ABC, AC|B, B|AC\}, \tag{5.2}$$

where | denotes a partition into classes 1 and 2 on the LHS and RHS respectively. For instance, the partition $AC|B$ means that we classify an observation to class 1

if $x = A$ or C and to class 2 otherwise. An example of a restricted set of partitions excluding the true partitions $AC|B$ and $B|AC$ can be defined as follows

$$\Phi' = \Phi \cap \overline{\mathcal{C}} = \{A|BC, BC|A, C|AB, AB|C, ABC|\emptyset, \emptyset|ABC\}. \tag{5.3}$$

With $\mathcal{C}$ removed from the scene, the restricted set, Φ', is made available to the learner and using it the learner attempts to "accurately learn" the true rule.

Claim: *Weak learnability holds for the restricted class of partitions — that is, given any set of non-overlapping probability distributions (such that* $p_{1a}p_{2a} + p_{1b}p_{2b} + p_{1c}p_{2c} = 0$*) we can find a partition* $\phi \in \Phi'$*, such that the expected misclassification error is less or equal to some constant,* $\lambda < \frac{1}{2}$.

Proof: Let two of the three possible values be from one class and the third from the other class. Now, suppose class 1 has support AC and class 2 has support B or vice versa (otherwise there will be a partition in Φ' yielding $\xi_{D,\text{pop}} = 0$ misclassification error). Consequently, $p_{1a} + p_{1c} = 1$ and $p_{2b} = 1$, $p_{1b} = p_{2a} = p_{2c} = 0$. For each partition $\phi \in \Phi'$ the corresponding Bayes' rates are given as in Table 5.1.

No	Partition	Bayes error rate
1	$A\|BC$	$\pi_1 p_{1c}$
2	$BC\|A$	$\pi_1 p_{1a} + \pi_2$
3	$AB\|C$	$\pi_1 p_{1c} + \pi_2$
4	$C\|AB$	$\pi_1 p_{1a}$
5	$ABC\|\emptyset$	π_2
6	$\emptyset\|ABC$	$\pi_1(p_{1a} + p_{1c}) = \pi_1$

Table 5.1: *The restricted partitioning of* Ω *in column 2 shows classes 1 and 2, with priors* π_1 *and* π_2 *and supports* AC *and* B*, lying on the LHS and RHS respectively. The Bayes error rates for each possible partition* $\phi \in \Phi'$ *are in column 3.*

To illustrate the calculation consider the first line of Table 5.1. The partition classifies x to class 1 if $x = A$ and to class 2 if $x = B \quad \text{or} \quad x = C$. As noted above, class 1 has support AC and class 2 has support C. However, the ideal rule is: given an observation x on $\Omega = \{A, B, C\}$, we classify x to class 1 if $x = A$ or $x = C$ and to class 2 if $x = B$, based on the Bayesian allocation rule, by which we allocate an individual to a class maximising (4.1). Given that x belongs to class 1

with probability π_1 we make a mistake if $x = C$ with probability p_{1c}. Given that x belongs to class 2 with probability π_2, we never make a mistake with this partition.

For any values of the class priors, π_1, π_2, and the probabilities, p_{1a}, p_{1c} the best rule minimises the Bayes error over $c \in \mathcal{C}$. Since $p_{1a} + p_{1c} = 1$, at least one of p_{1a} and p_{1c} must be $\leq \frac{1}{2}$. If $\pi_1 \leq \frac{2}{3}$, the minimum of the Bayes's error rates 1 and 4 in Table 5.1 is $\leq \frac{2}{3} * \frac{1}{2} = \frac{1}{3}$. If $\pi_1 \geq \frac{2}{3}$, then $\pi_2 \leq \frac{1}{3}$ and the Bayes' error rate 5 is $\leq \frac{1}{3}$. Hence for any set of parameter values, the best rule has an error rate less or equal to $\frac{1}{3}$. That is, if we set a threshold $\xi_T = \frac{1}{3}$, the best rule has an error $\xi_{D,\text{pop}} \leq \xi_T = \frac{1}{3}$, based on the "vote" for the smallest error — thus implying weak learning. ■

The plots in Figure 5.1 illustrate the above example with vertical and horizontal stump-based partitions of Φ'. Effectively, we envision a two-dimensional set-up in which $x = A$ corresponds to observing $0, 0$, $x = B$ to $0, 1$ and $x = C$ to $1, 1$. Since class 1 has support AC and class 2 has support B, the restriction imposed on Φ' implies that the set is confined to "stumps-only" partitions, yielding an inevitable misclassification error in each case. It requires a modified algorithm to devise a rule that would reduce this error.

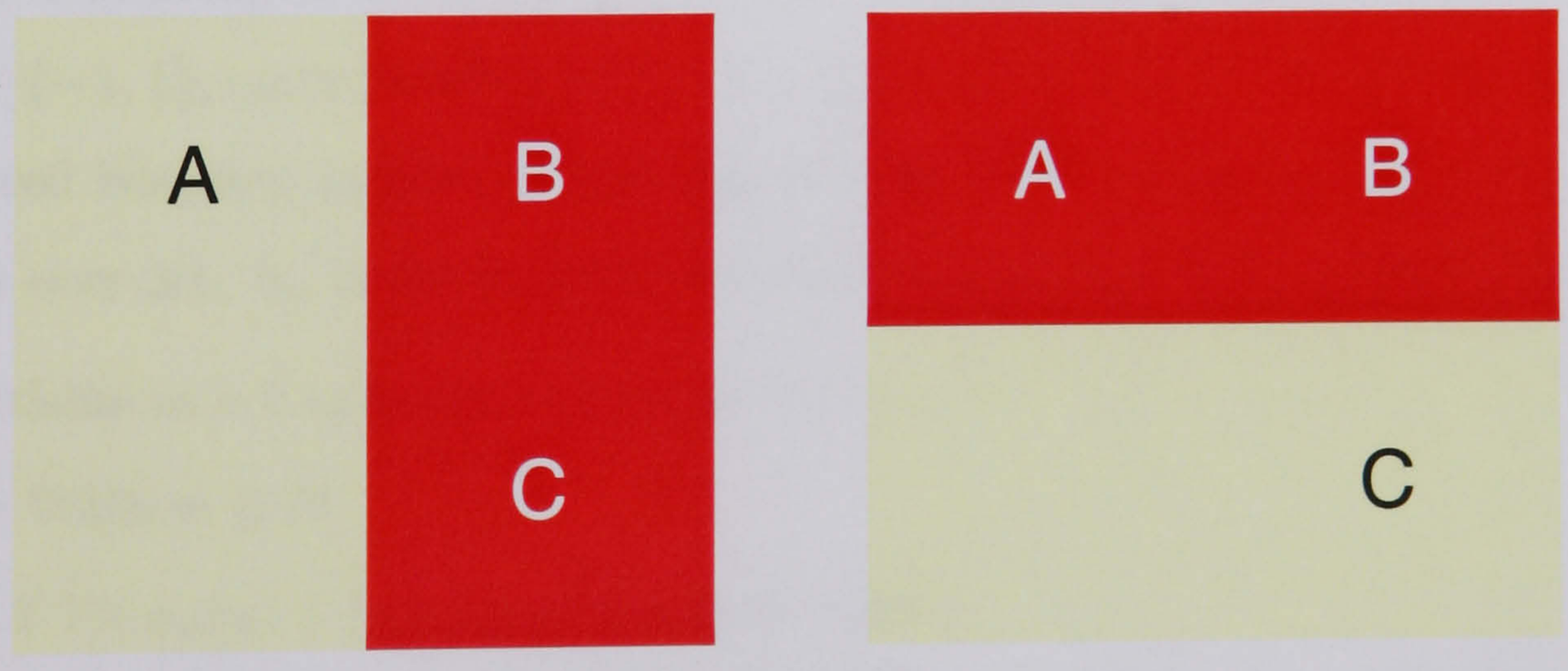


Figure 5.1: *The two panels illustrate the performance of a stump on a two-class data with supports AC and B for classes 1 and 2 respectively. Note that stumps-only splits cannot avoid misclassification according to the supports, as exhibited by the two possible stump-based splits in this figure. The vertical split on the LHS misclassifies A while the horizontal split on the RHS misclassifies both A and C. The colours red and cream represent the two partitions in the setup described above.*

5.3 The Discrete AdaBoost algorithm

The **Discrete AdaBoost** algorithm differs from its predecessors in that it **ada**pts to the errors returned by the weak classifiers. The method has been shown to work well with a domain-partitioning classification method such as a decision tree. The algorithm is initially invoked with equal weights to all observations but over successive iterations, it assigns more weight to observations that are currently misclassified.

The learner is trained using the current weights, appropriately updated and set to sum to one and the algorithm requires that this condition be maintained over the training set. The method's ultimate objective is to get as many observations as possible correctly classified. Its implementation requires that some conditions and ingredients be in place. Typically, boosting needs the following ingredients:

1. A weak learner, $\phi(x)$, and the way to update case weights.

2. Training set and the way to combine successive classifiers.

3. The number of iterations M, often a (large) tuning parameter.

5.3.1 The mechanics of boosting

Consider a training data set $(x_i, y_i),\ i = 1 : n$, where x_i is a vector valued feature and $y_i \in \{-1, 1\}$, corresponding to $C_k,\ k = 1, 2$ respectively. Then we can define the unobserved boosting prediction function in step 3 below, where $\phi_m(x)$ is a binary classifier over ± 1, b_m is the boosting constant and m labels the current iteration.

1. Initialise $m = 1$ and $w_i(1) = 1/n,$ for $i = 1, ..., n.$

2. Do While $m \leq M$

   2.1 Fit $\phi_m(x) \in \{-1, 1\}$, with weights $w_i(m)$.

   2.2 Compute $\xi_m = E_w\big[I[y \neq \phi_m(x)]\big] = \sum_{i=1}^{n} w_i(m)\big[I[y_i \neq \phi_m(x_i)]\big] / \sum_{i=1}^{n} w_i(m)$.

   2.3 Compute $b_m = \log[(1 - \xi_m)/\xi_m]$.

   2.4 Set $w_i(m+1) \leftarrow w_i(m)\exp\big[b_m * I[y_i \neq \phi_m(x_i)]\big]$.

   2.5 Scale $w_i(m+1) \leftarrow w_i(m+1)/\sum_i^n w_i(m+1)$.

   2.6 Set $\Phi_m(x) \leftarrow \text{sign}\big[\sum_{j=1}^{m} b_j\phi_j(x)\big]$.

   2.7 Update $m \leftarrow m + 1$.

3. The final classifier is: $\Phi_M(x) = \text{sign}\big[\sum_{m=1}^{M} b_m\phi_m(x)\big]$.

The algorithm is said to be discrete because $\phi_m(x)$ returns discrete ± 1 class labels. The name **AdaBoost** derives from the fact that at each iteration the algorithm "adapts" to its previous performance and seeks to "do better" by re-weighting the observations. Most applications have shown that for noise-free data, the algorithm will drive down the misclassification error without over-fitting. Convergence for noisy data is not guaranteed, although it may be attained at low levels of noise.

The misclassification error, case weights and the boosting constant jointly play a very crucial role in powering the **Discrete AdaBoost** engine. We give an algebraic account of the role played by the three parameters in the algorithm. Consider the interaction of the training misclassification error and the boosting constant. Note that as $\xi_m \to 0,\ b_m \to +\infty$ and as $\xi_m \to 1,\ b_m \to -\infty$. Clearly, if the misclassification error, $\xi_m = 50.00\%,\ b_m = 0$ and all the current weights are maintained. This is a "chess-board"-like problem, in which case the algorithm gets stuck, which implies that there will then be no point using boosting.

5.3.2 Interpretation of b_m and error generalisation

The parameter b_m can be interpreted as the log(**odds**), which is used to measure the importance that the algorithm assigns to the classifier $\phi_m(x)$. An intuitive understanding of this constant is important in understanding the mechanics of **AdaBoost**. The quantity $\frac{\xi_m}{1-\xi_m}$ can be interpreted as the **odds-ratio** for an observation being misclassified and $\log\frac{\xi_m}{1-\xi_m}$ is called its **log-odds** or simply **logit**(ξ_m). A detailed interpretation of boosting is given by Friedman *et al.* (2000) and Hastie *et al.* (2001).

For a clear insight into the general performance of the algorithm, we need to focus our attention on the final training error, ξ_M, returned by the final classifier $\Phi(x)$. In a general binary problem setting, a prediction that randomly guesses each case's class would have an error rate of $\frac{1}{2}$. A better than random performance can be obtained by expressing the error as $\xi_m = \frac{1}{2} - \alpha_m$, where $\alpha_m > 0$ denotes a measure of how much better than random the predictions $\phi_m(x)$ are. Freund and Schapire (1996) show that the training misclassification error for the final classifier is bounded, i.e.

$$\xi_M \leq \prod_m \left[2\sqrt{\xi_m(1-\xi_m)}\right] = \prod_m \sqrt{1-4\alpha_m^2} \leq \exp\Big(-2\sum_m \alpha_m^2\Big) \qquad (5.4)$$

The formulation implies that if each classifier performs slightly better than a random guess then the training error would be driven down exponentially fast.

5.4 Boosting examples

The conventional **Discrete AdaBoost** algorithm is defined in a two-class setting, which we adopt in our examples. This section presents three main examples of boosting. In Section 5.4.1 we introduce an example based on our interpretation of weak learnability as presented in Section 5.2.2. The second example is given in Section 5.4.2 based on simulated data with a sine-wave boundary, $k = 2$ classes and $p = 2$ variables and Section 5.4.3 explores boosting different tree levels. The sine-wave data is considered to be an excellent challenge to the tree-type classification methods — in particular, the tree stumps. To test the performance of the algorithm, we consider both overlapping and non-overlapping classes.

5.4.1 An example to illustrate weak learnability

This example derives from the ideas in Section 5.2.2 and seeks to produce a rule that out-performs the stumps in Figure 5.1. Suppose we set the class priors to be $\pi_1 = 0.6$ and $\pi_2 = 0.4$ and the conditional probabilities for observing the variable $x = A,\ B$ and C given that we are in a particular class, be denoted by $(p_{1a}, p_{1b}, p_{1c}) = (0.8, 0, 0.2)$ and $(p_{2a}, p_{2b}, p_{2c}) = (0, 1, 0)$. The probabilities of observing the variable $x = A,\ B$ and C are $(0.48, 0.4,$ and $0.12)$ respectively. We use these numerical values to illustrate how boosting works.

Partition	Error	b_m	Cum. Values	Boost. Error
$A\|BC$	0.12000	1.99243	(1.99243, -1.99243, -1.99243) + - -	0.12
$CAB\|\emptyset$	0.22727	1.22377	(3.21620, -0.76865, -0.76865) + - -	0.12
$C\|AB$	0.17647	1.54044	(1.67576, -2.30910, 0.77179) + - +	0.00

Table 5.2: *Optimal partitions with boosting-based error reduction parameters. The boosting partitions, top-down, show how boosting finally manages to avoid the inevitable error, by combining the classifiers in the first two steps to form the third.*

We rename the class labels 1 and 2 as 1 and -1 to fit the boosting convention. The

first split is $A|BC$, with $\xi_m = 0.12$, resulting from $x = C \in C_2 = -1$. The boosting constant computed on the basis of this error rate is $b_m = \log \frac{(1-0.12)}{0.12} = 1.99243$. We update the weights and re-normalise to determine new splits. The second and third splits, their corresponding boosting constants as well as the cumulative values are given in Table 5.2. Note that the order of the cumulative values in column 3 is that of A, B, C, and also that perfect partition is attained after only three iterations.

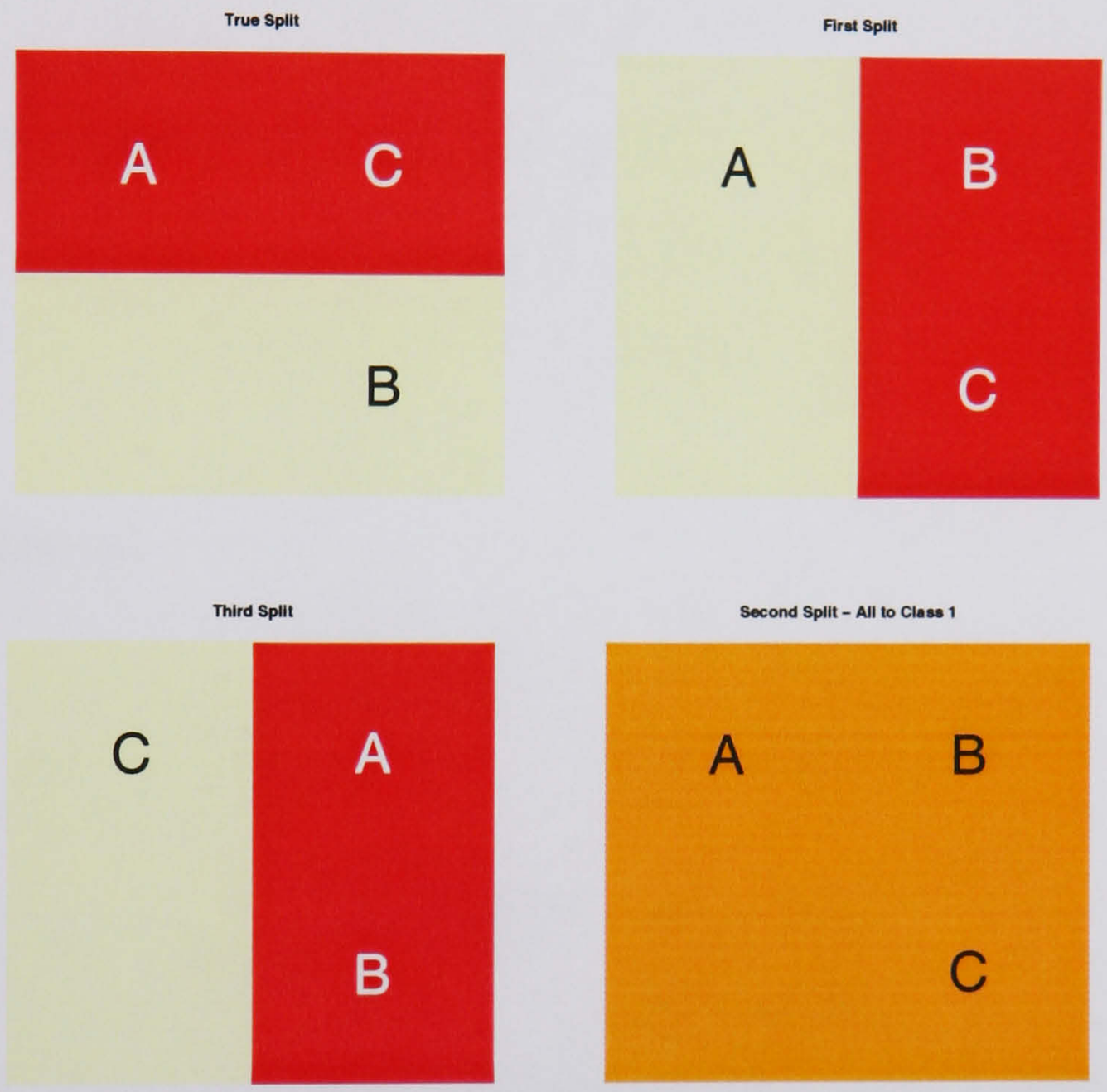


Figure 5.2: *The four panels provide a graphical summary of the stump boosting in Table 5.2. From the north-west, clock-wise, the panels correspond to the true, first, second and third partitions respectively. In only three steps, the boosting algorithm is capable of recovering the true partition. Again here, the two colours on each plot correspond to the two partitions as defined earlier, while the single colour on the south-east plot implies that all three cases are allocated to the same class.*

Figure 5.2 provides a graphical summary of Table 5.2. Starting from the true partitions panel in the north-west, the remaining panels in the clock-wise direction correspond to the first partition and the subsequent combined partitions obtained as the algorithm adapts to its previous performance.

5.4.2 Noise-free data with a sine-wave boundary

The RHS panel in Figure 5.3 gives a 2-D plot of 300 data points of simulated training dataset used in this example. The simulation of both the training dataset and an equally-sized test dataset proceeded as follows

$$\begin{cases} x_{1i} \sim \mathrm{U}(0, 3\pi) \\ x_{2i} \sim \mathrm{U}(-1, 1) \\ i = 1, ..., n \end{cases} \tag{5.5}$$

and then making allocations by following the rule

$$y_i = \begin{cases} 1 & \text{If } \sin(x_{1i}) > x_{2i} \\ -1 & \text{Otherwise,} \end{cases} \tag{5.6}$$

which is equivalent to

$$f_1(x) = \begin{cases} (\frac{3\pi-2}{6\pi})^{-1} & \text{for} \quad x \in R_1 \\ 0 & \text{for} \quad x \notin R_1 \end{cases} \tag{5.7}$$

$$f_2(x) = \begin{cases} (\frac{3\pi+2}{6\pi})^{-1} & \text{for} \quad x \in R_2 \\ 0 & \text{for} \quad x \notin R_2 \end{cases} \tag{5.8}$$

where R_1 and R_2 are regions above and below the sine-wave boundary defined as

$$R_1 = x_1, x_2 | x_1 \in (0, 3\pi),\ 1 \geq x_2 > \sin x_1 \tag{5.9}$$

$$R_2 = x_1, x_2 | x_1 \in (0, 3\pi),\ -1 < x_2 < \sin x_1 \tag{5.10}$$

Class priors corresponding to the R_k regions are defined as $\pi_1 = \frac{3\pi-2}{6\pi}$ and $\pi_2 = \frac{3\pi+2}{6\pi}$ respectively. If we define the areas of the two regions as $A_k,\ k = 1, 2$, the class priors can then be seen to be proportional to the areas, i.e. $\pi_k \propto A_k$.

Boosting tree-stumps in the non-overlapping case

The RHS panel in Figure 5.3 shows our simulated data with non-overlapping classes separated by a sine-wave boundary. The classes ± 1 have been labelled as 1 and 0 for ease of visibility. The LHS displays a tree stump — a tree with a single splitting node at which the classifier, $\phi(x)$, splits the data into two classes based on a

particular variable and threshold. The training and test misclassification errors for the stump were 24% and 28% respectively, while those from a full tree were 3% and 6.63% respectively. Given a sine-wave type of boundary, any linear method would grossly misclassify the data, hence the need for a more sophisticated technique.

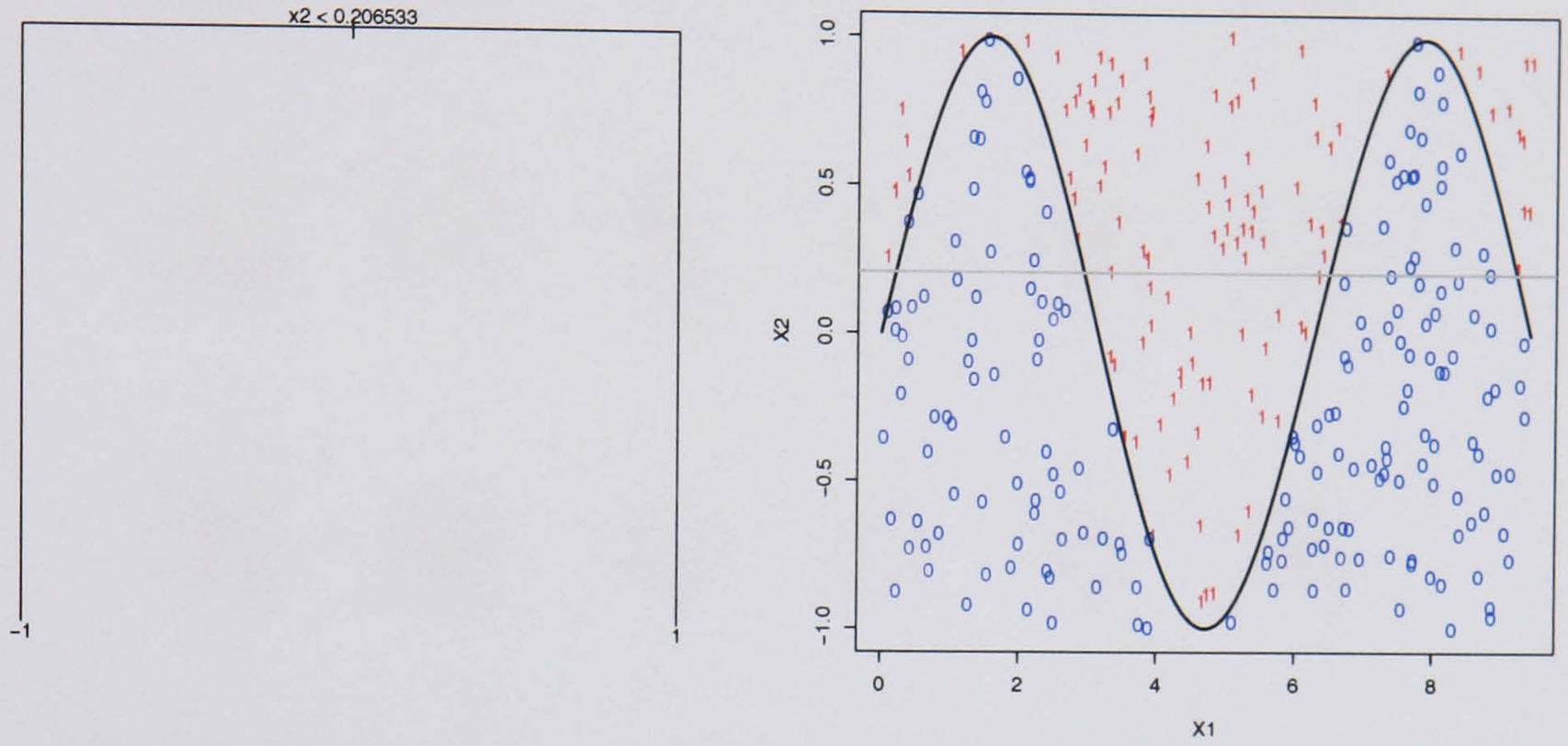


Figure 5.3: *The LHS panel shows a tree stump and the RHS exhibits a typical stump-based partitioning of the region. Note how grossly the cases are misclassified by the tree stump. The training and test errors are 24% and 28% respectively.*

Although boosting can be implemented at any tree level, the algorithm would not clearly manifest its strength if the underlying classifier is too strong. A simple starting point would, therefore, be to boost a weak classifier such as a decision stump. Using the stump as a weak learner the algorithm was trained and tested on the two independent data sets, i.e. by using the recursively boosted stump to make predictions based on the training and test data sets.

The impact of added classifiers

Figure 5.4 shows the sequential effects of boosting. The first and third rows show the stumps contributing at iterations 1 through 6 and 495 through 500 while the second and fourth rows show the cumulative boosting positions. The training error converged to 0.00% from 24.00% after 134 iterations and the test error dropped from 28.00% to oscillate at about 07.00%. In isolation, the stump predictions grossly misclassify the data but collectively they create an improving pattern with an increasing number of iterations as exhibited in the second and fourth rows of Figure 5.4.

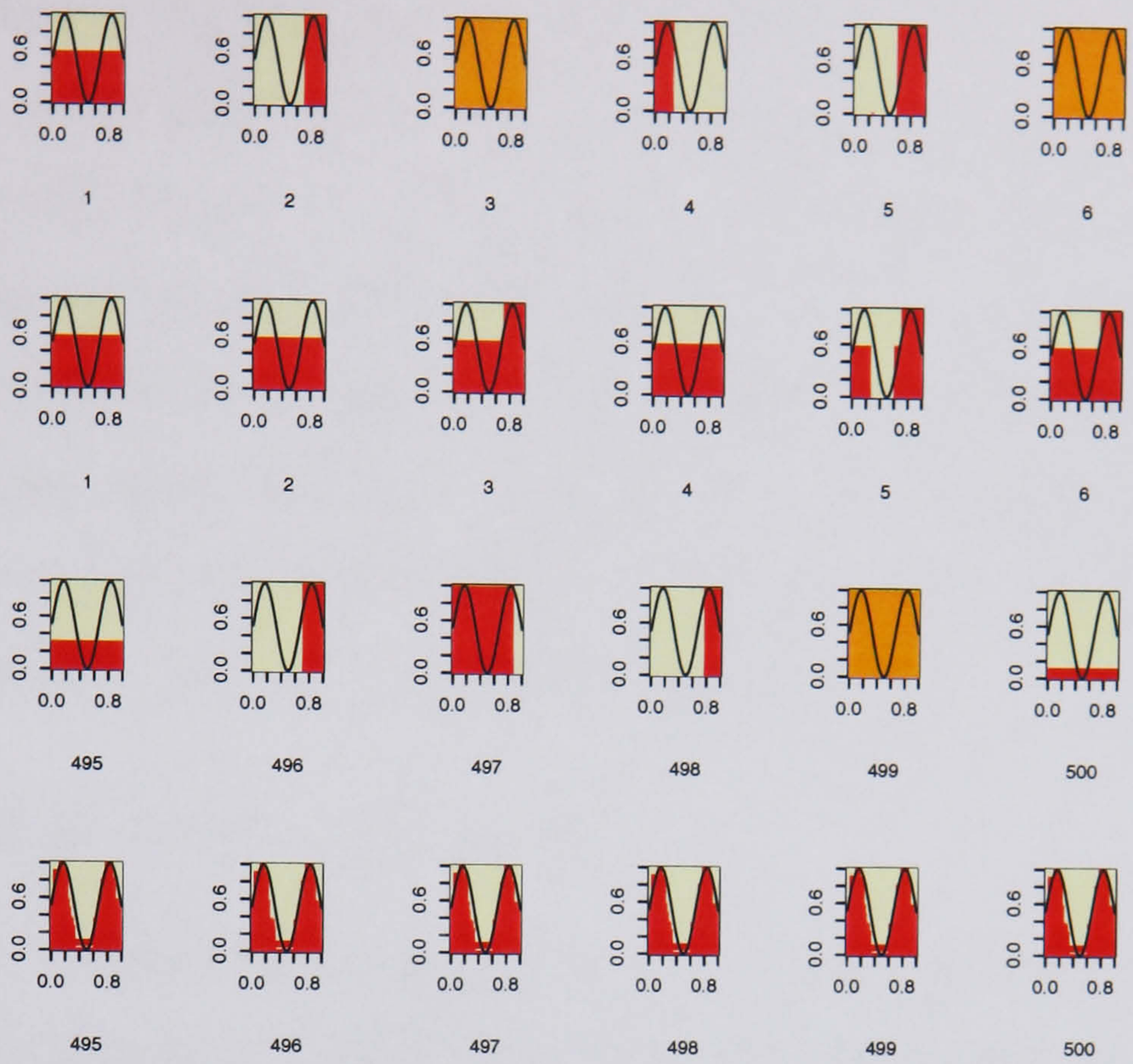


Figure 5.4: *Resulting patterns from boosting the stumps on the test data (other than on the data the algorithm was trained on). Again, the two colours denote each of the two classes and a single colour indicates that all cases are allocated to the same class. The first and third rows are stump-only partitions and the second and fourth are obtained by cumulatively combining the stumps. Notice the big jump after the* 4^{th} *iteration onto the* 497^{th} *iteration. The boosting algorithm recovers the true boundary after approximately 134 iterations with a misclassification error of 7%.*

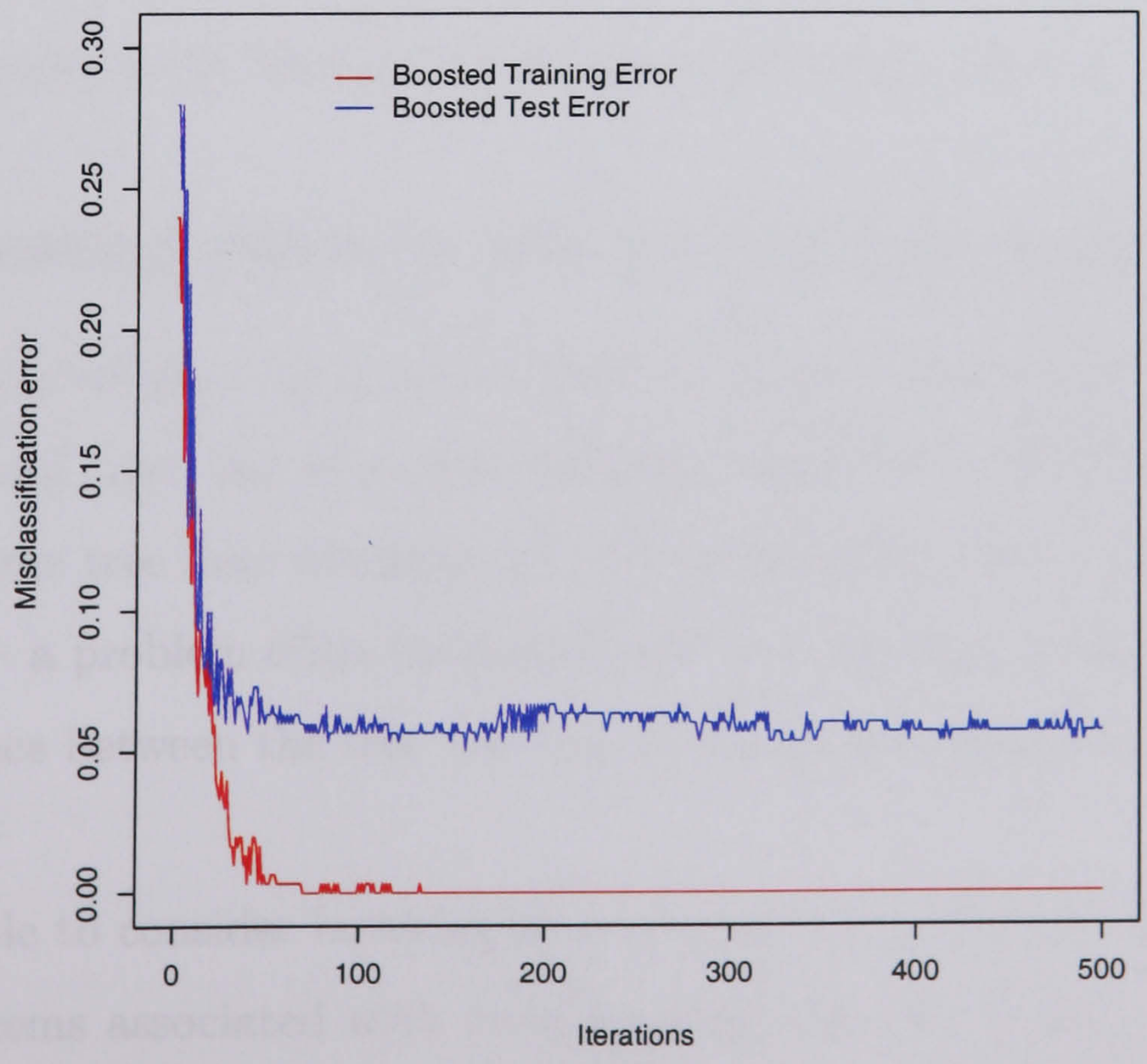


Figure 5.5: *The pattern of prediction errors based on a recursively "boosted" stump, for the noise-free dataset in Figure 5.3. The training error was driven to zero from 24% after 134 iterations while the test error dropped from 28% to about 07.00%. In a large tree scenario, the training and test error rates*

The impact of added classifiers as the boosting algorithm moves forward can clearly be seen in this figure. An initial pattern of the boosted classifier is given by the first prediction and, in this case, the second prediction has had no impact on the boosted classifier. It is only when the third stump is grown and added on, that a pattern starts to emerge and at this point the training and test errors drop from 24.00% and 29.00% to 21.00% and 27.00% respectively. But both errors subsequently go up again, before going down, as the next few predictions are added on, showing that boosting does not guarantee monotonically decreasing errors. The training and test misclassification error rates are summarised in Figure 5.5.

The impact of M and the sample size

Other than the structural components of the boosting procedure, its results are influenced by the number of iterations as well as the sample size. In one experiment we doubled the sample size to 600 data points and boosted to $M = 500$. The initial training error was 29.17% and converged to zero after 325 iterations, while the test error started at 26.33% and went on to oscillate at about 4.00% after about 200 iterations. As a rule, as M becomes increasingly large, i.e. as more and more classifiers are added, the misclassification training error ceases to change or simply oscillates within negligible margins. Doubling the sample sizes had a clear positive impact on the test error although the learning process was slightly lengthened. We return to these issues in Chapter 6, in which we provide a detailed study of boosting.

5.4.3 Boosting different tree levels for the noise-free case

Typically, a tree stump — as a weak learner — would yield a higher misclassification error than a full tree. An immediate solution would be to grow a larger tree, but although a large tree may minimise the misclassification error, it remains prone to over-fitting — a problem often tackled via the tree pruning technique that seeks to strike a balance between the tree size and the misclassification error.

It is reasonable to consider boosting as a technique that may be applied to resolve the two problems associated with weak learning and over-fitting. The results and comparisons of this section may give an insight into the plausibility of the algorithm.

The method has been praised not only for minimising the misclassification error, but also for its resistance to over-fitting (Freund and Schapire, 1997; Friedman *et al.*, 2000). In consideration of these properties, we implement the algorithm on multiple simulations of different tree sizes. Intuitively, this means boosting a tree with a decreasing misclassification error rate.

Tree Size	Training Error			Test Error			
	Initial	Final	Iterations	Initial	Minimum	Iterations	Final
2	27.7%	0%	181	31.78%	6.23%	180	6.51%
3	21.12%	0%	35	25.48%	6.43%	45	6.96%
4	9.12%	0%	15	13.63%	5.95%	40	6.5%
5	7.1%	0%	8	12.95%	6.36%	40	6.75%
6	7%	0%	7	12.05%	6.51%	30	6.91%
7	7%	0%	6	11.45%	5.78%	20	6.18%
8	7%	0%	6	12.06%	6%	19	6.41%
9	7%	0%	6	11.65%	5.41%	40	5.81%
10	3.2%	0%	5	10.48%	6.45%	27	6.9%
11	2.62%	0%	4	7.86%	5.31%	25	5.65%
12	2.62%	0%	3	9%	6.25%	25	6.78%

Table 5.3: *Results of sequentially boosting trees of different sizes for non-overlapping sine-wave data. The number of iterations for the test error refer to the points at which the minimal errors were reached. As to be expected, there is a dramatic decrease in the number of iterations as the tree grows in size.*

Table 5.3 provides a detailed summary of 50 simulations on different tree sizes — it can be seen that at some point, both the initial and final test errors take an upward trend as the tree size increases. Similarly, the number of iterations dramatically decreases with an increasing tree size. Generally, the pattern seems to confirm the assertion that the algorithm is meant for weak classifiers. It is evident from the plots and the table that as the tree size increases, at best the algorithm is left with little to do and at worst it appears to over-fit the data.

Figure 5.6 reveals the pattern of both initial and final training and test errors, as well as the minimum test error achieved by each tree size. Theoretically, we should be able to achieve a zero initial training error with a large enough tree, although this was not achieved even for tree size 12.

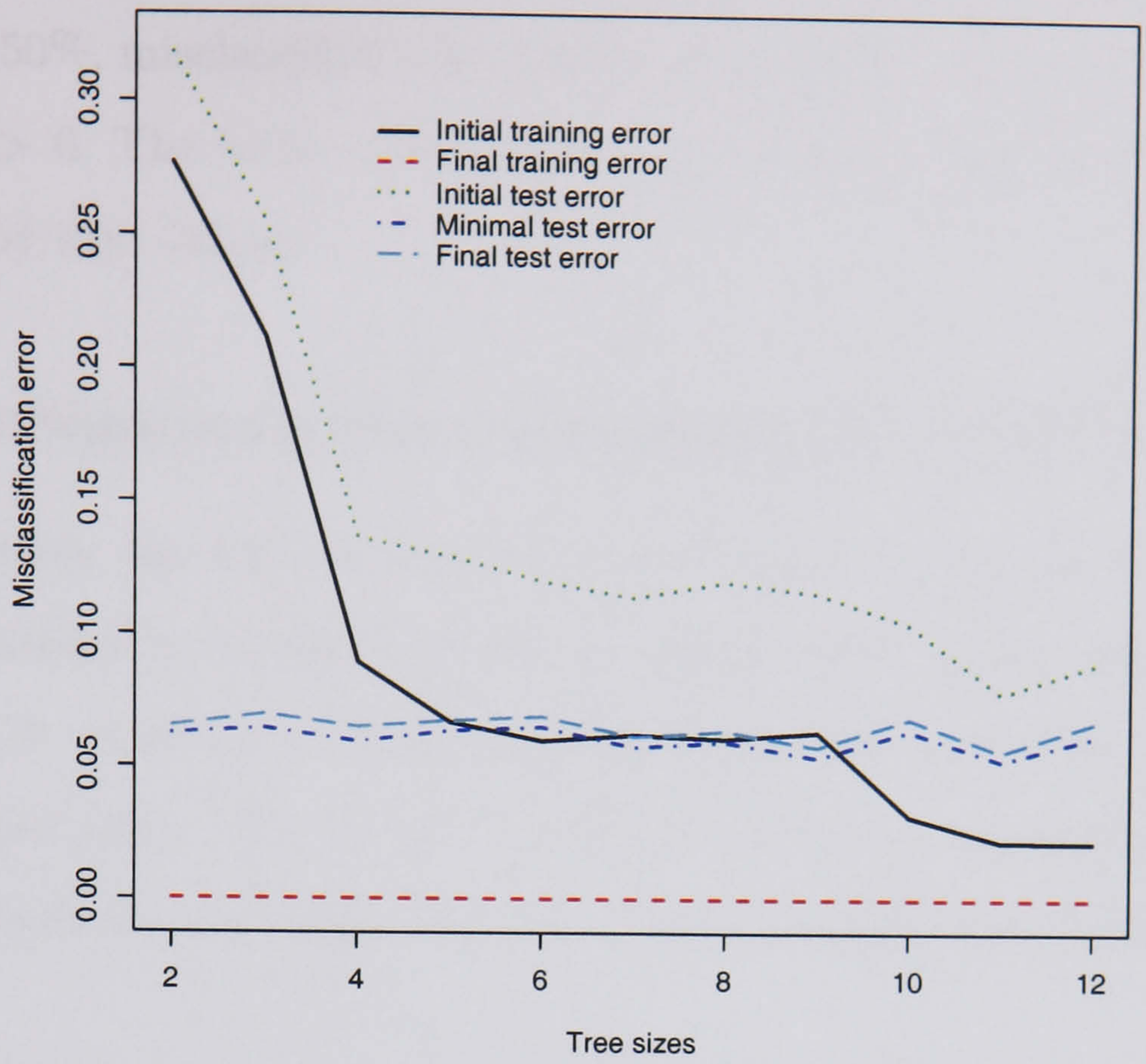


Figure 5.6: *Initial and final errors for training and test sets based on 50 simulations of boosting different tree sizes, for a noise-free sine-wave data of size 300. Data over-fitting is evident as the tree size increases. The minimum error reached is also plotted for comparison with the final error.*

The foregoing example raises a crucial question as to whether it is possible to determine the optimal complexity of the classifier to boost. Considering the resulting error patterns between a tree stump and a size 8 tree, as shown in Table 5.3, tree size 7 may be considered optimal. Beyond that point, tree sizes 9 and 11 may also be considered optimal. The underlying philosophy is that boosting a "not-very-weak" classifier is plausible and will usually yield better results than boosting a "very-weak" or "very strong" classifier. Peter Bühlmann and Bin Yu raise this issue in their discussion of Friedman *et al.* (2000), as does Greg Ridgeway.

5.5 An exploratory study of observational weights

In this section we provide a brief account of the role played by case weights in the boosting algorithm. The weight updating function multiplies the previous weights by the exponential of the boosting constant. If an observation is currently misclassified it is assigned a new weight by having its previous weight multiplied by $e^{b_m} > 0$, which boosts up its weight and therefore hopefully increases its chances of being correctly

classified in the next step. The foregoing situation applies only in the case of $\xi_m \leq 50\%$. If $\xi_m > 50\%$, misclassified observations have their weights reduced by dividing them by $e^{b_m} > 0$. The latter case implies that correctly classified observations are up-weighted by that factor.

5.5.1 Ambiguously and unambiguously located observations

We wish to study the factors that drive boosting case weights — we start off by considering ambiguity of observational locations. We consider three observations, 84, 91 and 129 which are a long way, moderate way and close to the sine-wave boundary respectively. The top plot in Figure 5.7 shows the location of the observations and the bottom plot shows how the boosting weights vary during the iterations.

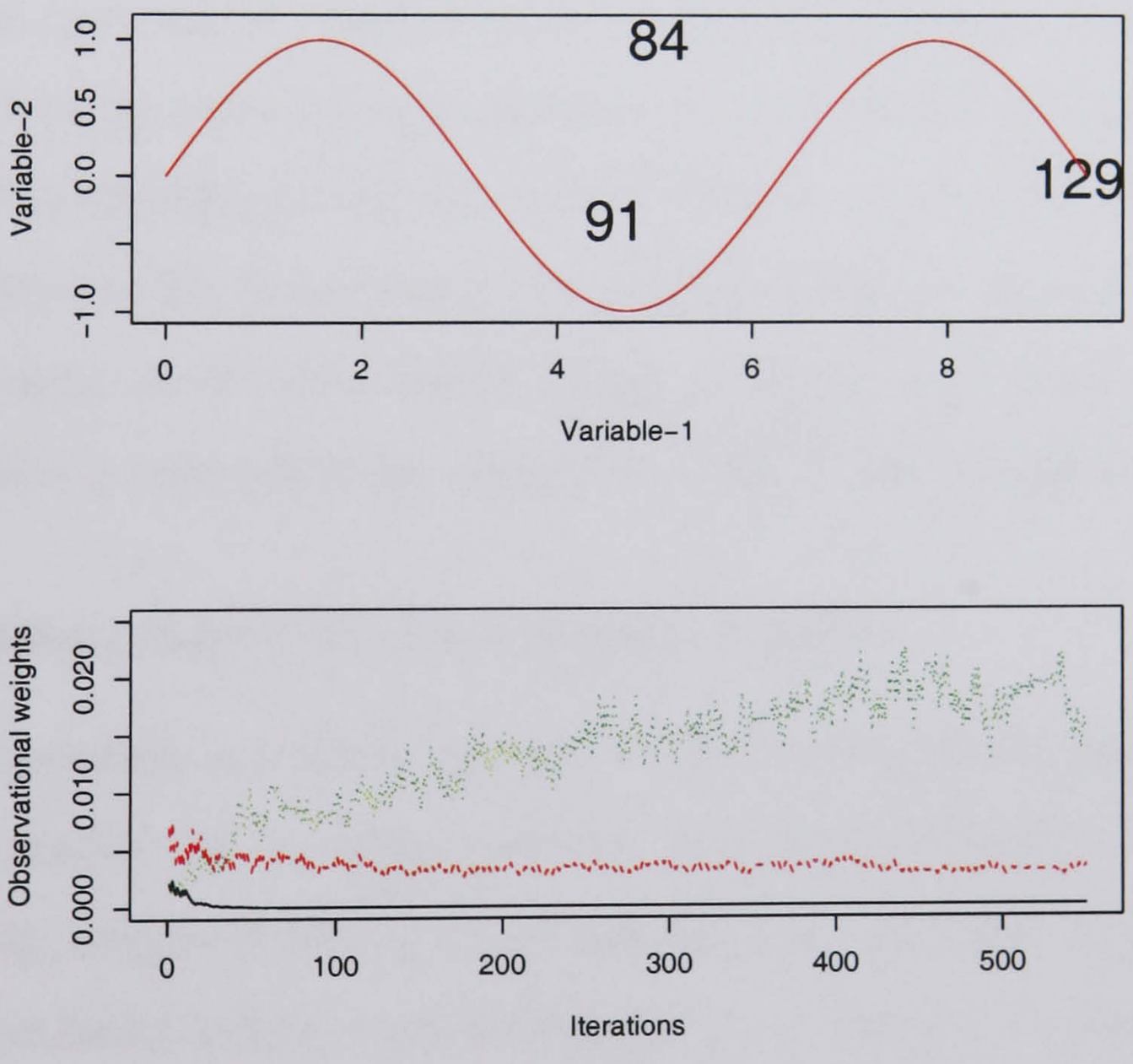


Figure 5.7: *The upper plot gives geometrical locations for the three observations. The bottom plot provides graphical patterns for the three observational weights returned by the boosting algorithm — the three lines, bottom-up, correspond to the case weights for observations 84, 91 and 129 respectively. Notice how the weight for the ambiguously located observation 129 tend to increase with the number of iterations.*

Note that the weights are higher for observations closest to the boundary. Observation 129, closest to the boundary, was the hardest to classify, hence the algorithm recursively assigned it very high weight. Observations 84 and 91 both lie in a clearly

unambiguous region, although the former is farther from the decision boundary than the latter and as such the algorithm finds it easier to classify it. The weight of observation 84 stabilises close to zero after about 100 iterations while that of observation 91 is constantly updated alongside those of the other observations. The emerging patterns seem to suggest that as the number of iterations increases, re-weighting stabilises and we can expect to see a converging pattern of weights in the final plots.

One likely source of error in data handling is class "mis-labelling". Typically, "mis-labelling" may happen for any observation although an observation farther away from the boundary may be considered to be less prone to it than an observation close to the boundary. We use observations 84 and 91 in Figure 5.7 as candidates for "labelling errors" in a non-overlapping training data set. The true class labels for the two observations were manually switched to belong to an opposite class. The result was particularly interesting for observation 84 as its weight dramatically increased. Switching only the label of observation 84 led to no pronounced impact on the error but switching both 84 and 91 yielded a training error of 0.7%. The farther from the decision boundary is the switched label the harder boosting finds to classify it. Clearly, at all times, higher weight is allocated to observation 84 than to 91. The converse is true when the observations lie in the corresponding true classes.

The converging pattern of observational weights

Because the algorithm iteratively produces different sets of weights, it would be interesting to consider the emerging patterns as it moves forward. A clearer pattern of the changing weights in the course of running the algorithm can be seen by plotting the accumulated weights versus observations. What one would expect to see in the final plot is a pattern with the weights of the "hard-to-classify" observations still standing out as outliers, as exhibited in the RHS panel of Figure 5.8. After 300 iterations, some observations, e.g. 188 and 175 still stand out with very high weights. This way, the algorithm may be said to possess a diagnostic power.

Despite what the plots may suggest, the weights never actually reach zero, but asymptotically approach it, particularly for the unambiguous observations. Since the weights sum to 1, even highly weighted observations would have weights be-

tween zero and one, and getting increasingly smaller as the sample size increases.

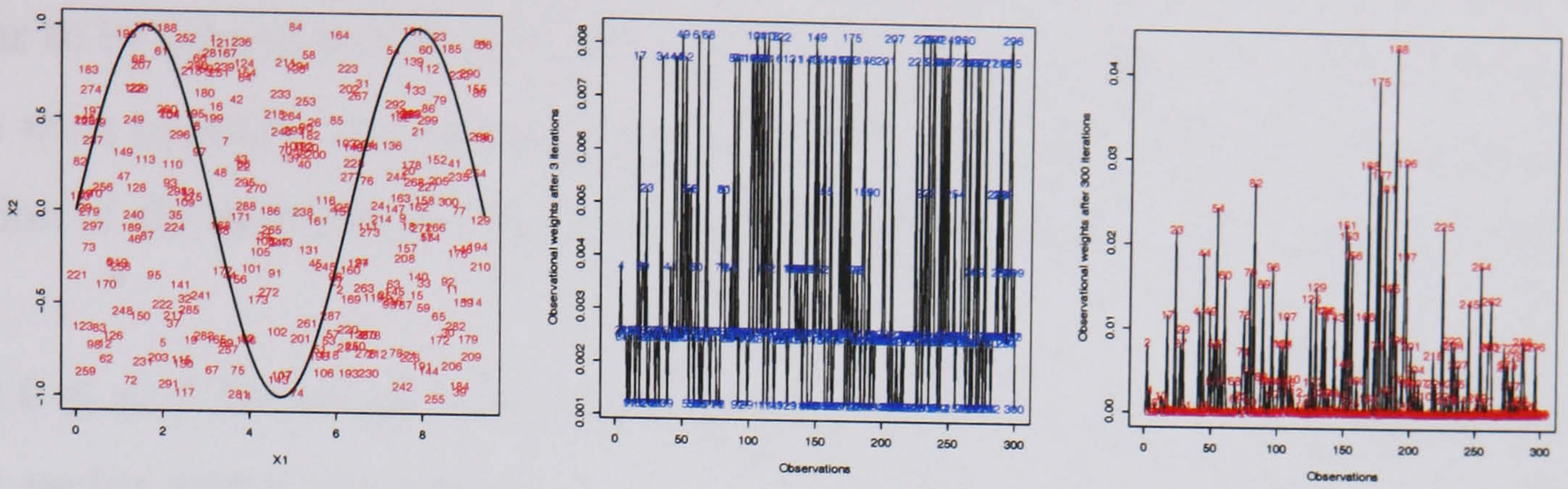


Figure 5.8: *The LHS panel shows observational locations and the middle and the RHS panels present the status of observational weights after* $M = 3$ *and* $M = 300$ *iterations respectively. Note, in particular, the "hard-to-classify" cases, in the RHS panel, that still stand out as "outlying", after* $M = 300$ *boosting iterations.*

To get an insight into what happens to the case weights, consider the initial weights as a cut-off point and look at the observations that end up with higher weights than this threshold. The algorithm initially starts with an equal weight of $1/n$ for each observation. While the middle panel in Figure 5.9 shows a large number of observations weighted above the threshold, it finally emerges that only a handful of observations end up with weights higher than the threshold. This pattern is exhibited in the RHS panel of the plot — implying a diagnostic feature in boosting.

Although there seems to be a converging pattern in the RHS panel, the algorithm may not guarantee converging weights for some of the badly-located observations, the weights of which may remain oscillating. Oscillation of the weights can possibly be explained by an alternating cycle such as the following: Once misclassified an observation is highly-weighted in the next round, down-weighted in the following round, it gets misclassified thus entailing a high weight again.

The discussion thus far suggests that there is a clear relationship between an observation's distance from the boundary and the weight allocated to it at every iteration. In a non-overlapping setting, the smaller the distance, the higher the weight and vice versa. Although observational locations remain geometrically static, the changing weights may eventually place an observation into a different region.

5.5.2 Observational distances from the boundary

We have seen that the maximum observational weights reached at each iteration appear to be related to the minimal observational distance from the decision boundary. We wish to study this relationship and as a starting point, we consider the computation of the shortest distance of each observation from the boundary $y = \sin(x)$.

Let $0 \leq x_i \leq 3\pi$ and $y_i = \sin(x_i)$, where $i = 1, ..., v$ and v is the length of the boundary vector and let α_j and β_j, $j = 1, ..., n$ represent the first and second predictors in our sinusoidal data respectively, then we can plot a 2-D plot of α_j versus β_j as shown in Figure 5.9. The minimum distance of each observation from the boundary is computed as

$$\Delta_j^2 = \min\{(x_i - \alpha_j)^2 + (y_i - \beta_j)^2\}. \tag{5.11}$$

It is logical to think of an observation having a large such distance as lying in an unambiguous region, therefore likely to be lowly weighted and vice versa. The middle and RHS panels in Figure 5.9 present two 2-D plots of the distances against the final weights after 5 and 500 boosting iterations respectively. Boosting considers observational weights as a decreasing function of distance, an inverse relationship that becomes more pronounced as the number of iterations increases.

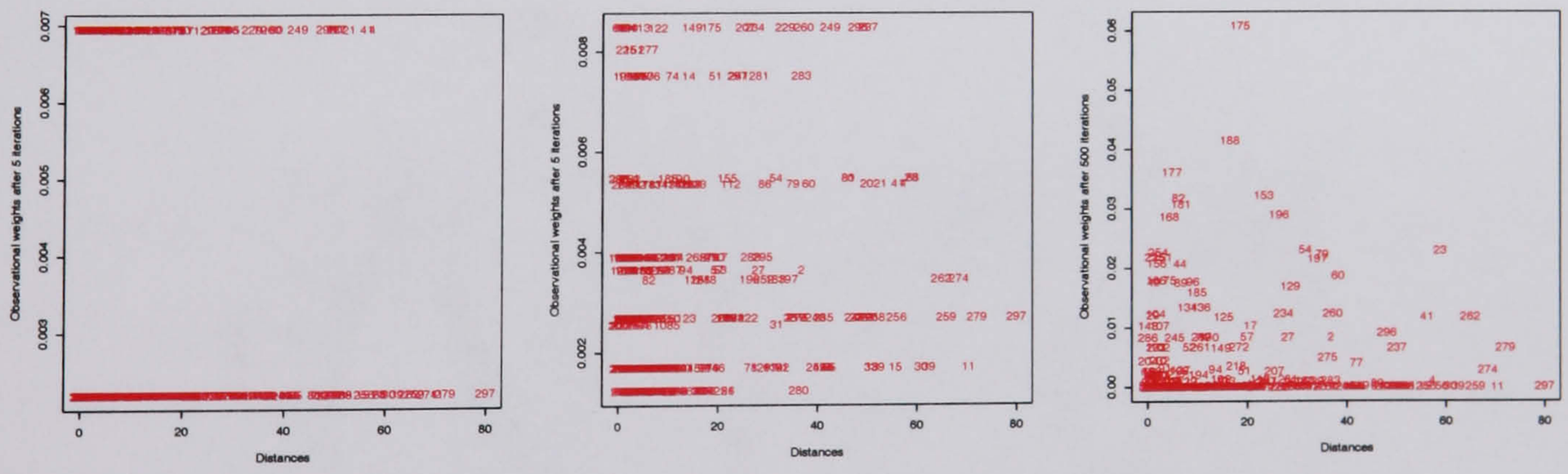


Figure 5.9: *The three panels, left-to-right, represent plots of distances from the decision boundary against observational weights after 1, 5 and 500 iterations respectively, corresponding to the data shown on the LHS panel of Figure 5.8. As the number of iterations increases observations with higher weights seem to be clustered near the south-west corner — that is, very close to the decision boundary.*

Most of the high observational weights in the RHS panel in Figure 5.9 are clustered near the decision boundary. The fact that after a large number of iterations the misclassification error stabilises, implies that the inverse relationship between distances

and weights should simultaneously stabilise. Although the two plots only partially describe the patterns the algorithm generates, they are still good enough to show the changing weight-distance relationship in the course of the algorithm.

5.5.3 A simple example involving noisy data

In this Section we re-run some of the examples in section 5.5.2 based on a similar data set with added noise. Noise was introduced by simply perturbing the dependent variable in (5.6) with a standard random normal component. At low values of m (the number of iterations), the relationship between distances and observational weights was similar to the one in Figure 5.9, with the algorithm focusing mainly on both the misclassified and badly-located cases. However, the added noise made it harder for the case weights to stabilise even at very large values of m — there was still no clear sign of stabilisation, with over 50% of the observations having weights above average.

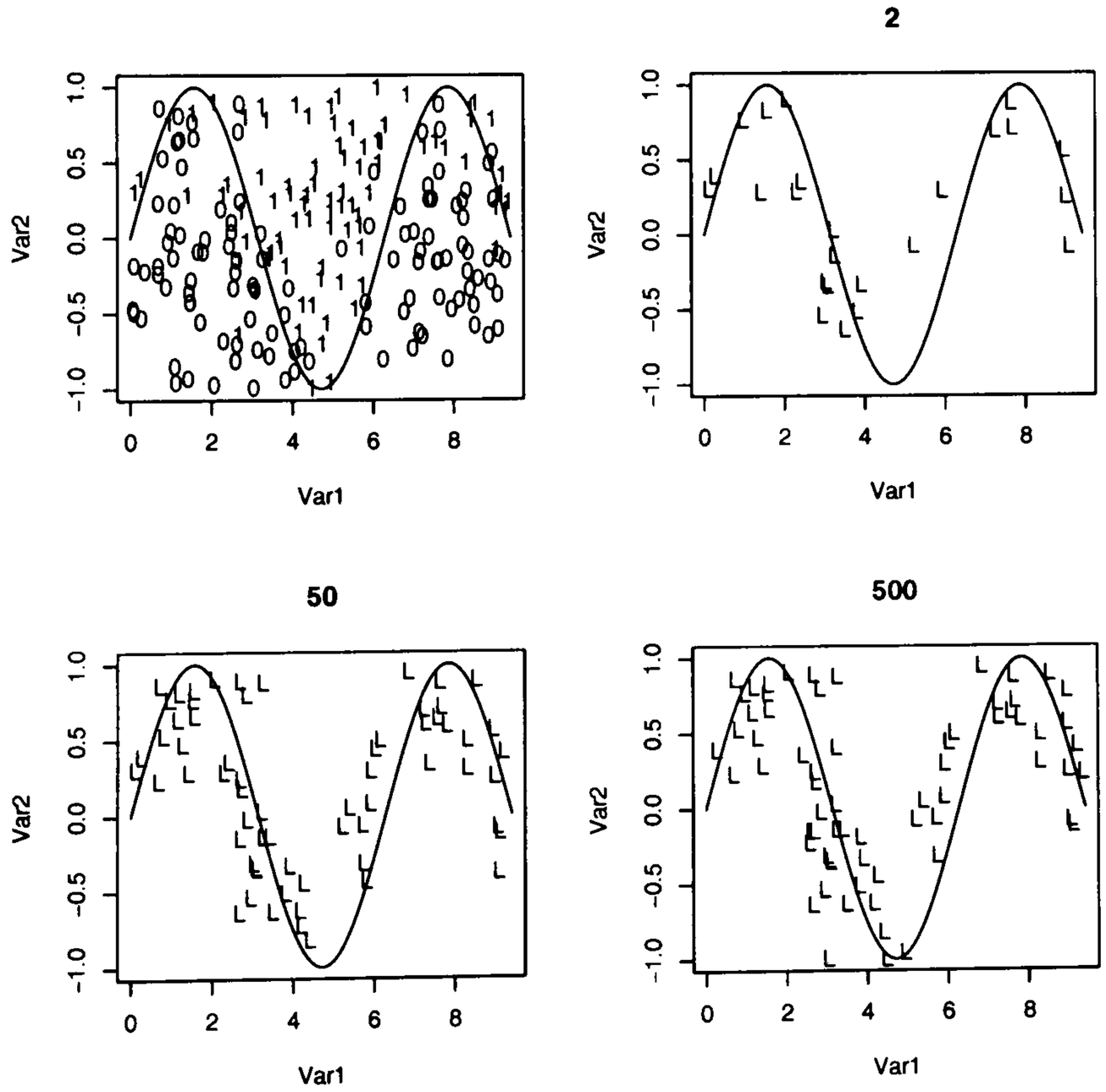


Figure 5.10: *The north-western panel exhibits a noisy sine-wave dataset. The remaining three panels — the north-eastern, south-western and south-eastern exhibit an increasing intensity about the decision boundary for maximum observational weights exceeding the mean weight through iterations 2, 50 and 500 respectively.*

Figure 5.10 portrays a clearer picture of the efforts made by the algorithm to get correct allocations for the "badly-located" and misclassified cases. We use the label

"L", short for "large" to denote maximum case weights above average. Effectively, over a number of iterations, the algorithm records the maximum case weight reached and plots it at its appropriate geometrical location in the 2-D plot. The examples show the pattern after 2, 50 and 500 iterations, all of which clearly indicate maximum weights assigned to cases about the boundary as well the misclassified cases.

Finally, we compared the maximum weights exceeding the mean and those in excess of the 3rd quartile after 100 iterations. The results indicated that large weights are clustered around the decision boundary. The plots looked almost identical as the difference between the mean (0.0209) and the third quartile (0.0299) was trivial.

5.6 Concluding remarks

This chapter focused on a general analysis of the boosting algorithm. An early example from this chapter was to illustrate "weak learnability", which we did by isolating the set of true partitions from that of all partitions. The restricted set, i.e. the set of all partitions minus the true partitions, we identified with a "weak" learner while the set of all partitions was identified with "strong" classes of partitions. We showed that under certain learning conditions, the two sets yield the same results — that is, in the absence of the true partitions from the scene, available partitions may be combined to yield a good result.

Of particular importance to the performance of the algorithm were the misclassification error and the case weights. Working with the sine-wave data with a likelihood ratio a constant function of the distance from the boundary we noted how case weights are recursively updated in accordance with the observations' geometrical location. For a closer investigation into this behaviour we used a non-overlapping uniform model with $n = 300$ to track observational weights of selected "mis-labelled cases" throughout the course of boosting. To demonstrate the impact of the distance from the boundary, we hand-picked observations both closest to the decision boundary and farthest from it and switch their class labels.

5.6.1 Ambiguity of observational locations

The main message was that classification difficulty arises not only from the amount of noise, but also from the geometrical location of the noisy data points. One of the factors affecting boosting is the handling of ambiguously located observations, in particular those observations closest to the decision boundary. In its course, the algorithm attempts to get such observations correctly classified by assigning them higher weights. Even in the absence of class overlap, the cases closest to the decision boundary will typically carry higher weight than those farthest from it. It is reasonable to expect "mis-labelled" cases to be heavily weighted as the algorithm struggles to get them correctly classified.

5.6.2 Potential effect of labelling errors

We also considered "labelling errors" in data handling and saw that changing class labels for the observations closer to the decision boundary seems to have minimal effect, apparently because the algorithm pays more attention to ambiguous observations. However, by avoiding "labelling errors" in some parts of the data, the test error may be brought to a minimum. That is, errors in the test data would have a smaller impact on the algorithm if they occur close to the decision boundary than away from it. In practice we encounter various cases of class overlaps and if we have a rough idea of what the error distribution is to be, the error bands can be reduced or increased by exercising greater care in handling the cases within the locations that are likely to yield higher impact on the algorithm. This can be achieved by employing some data validation techniques.

5.6.3 Can we determine an optimal tree to boost?

In this Chapter, we raised the question as to what are the requirements on the classifier to ensure that boosting does not over-fit? From the classification trees context, this leads to the question whether or not it is possible to determine the optimal complexity of the classifier to boost. This is probably a hard question, to which, unfortunately, there is no clear-cut answer in the literature.

If the the recent research findings that the method works so well with trees are to be

believed, then seeking the answer to the question as to whether an optimal model to boost can be determined is a problem worth pursuing. As a step towards that end, we started off by trying boosting on a variety of tree sizes from the same data and the results suggest that a larger than a stump size does well, although further down the line the performance deteriorates. This may also be a function of the data used. Indeed, the search for an optimal model to boost is very much part of future directions.

Since its inception, the boosting method has been applied on a variety of datasets using both trees and neural networks. Its performance with trees led Leo Breiman to refer to it as "the best classifier enhancer in the world". Indeed, our applications here have been confined to a single dataset, but the nature of the data and the mechanics of the algorithm provide a general guideline into its strength. In the following Chapter, we delve further into the performance of boosting with trees and consider a variety of factors that affect its performance.

Chapter 6

Factors affecting boosting

6.1 Introduction

Two approaches to assess the performance of boosting can be distinguished. The first involves a controlled simulation procedure in which the class priors and conditional probabilities are specified. Data, on which to train and test the algorithm, can then be simulated in accordance with this model. The training sample size, state of the priors — balanced or imbalanced, and the densities constitute a natural part of the assessment. The second approach involves numerical analysis, in which, starting with a fixed dataset, a number of perturbations to the dataset are introduced and the behaviour of the algorithm is monitored.

This chapter looks at a number of factors that affect the performance of the boosting algorithm. Typically, most of the factors would have some kind of a joint impact on the algorithm and in order to determine the impact of each on the estimated error, we seek to isolate and investigate each one of them in some detail. We use two different different of simulated data with a fixed sine-wave boundary. Specifically, the chapter explores the impact of the following factors on the algorithm.

1. The population densities — typically, these determine the Bayesian error. We consider two densities, a piece-wise constant over an area and a conditional normal with different means and variance.

2. The class priors — both equal and unequal.

3. The levels of class overlap — both equal and unequal.

4. The size of the training dataset, n.

5. The number of boosting iterations.

The layout of this chapter is as follows. The design simulation study is described in Section 6.2 and Section 6.3 provides boosting results for the two models under different conditions. In Section 6.4, we give a comparative account of the two models. Concluding remarks and potential future directions are given in Section 6.5.

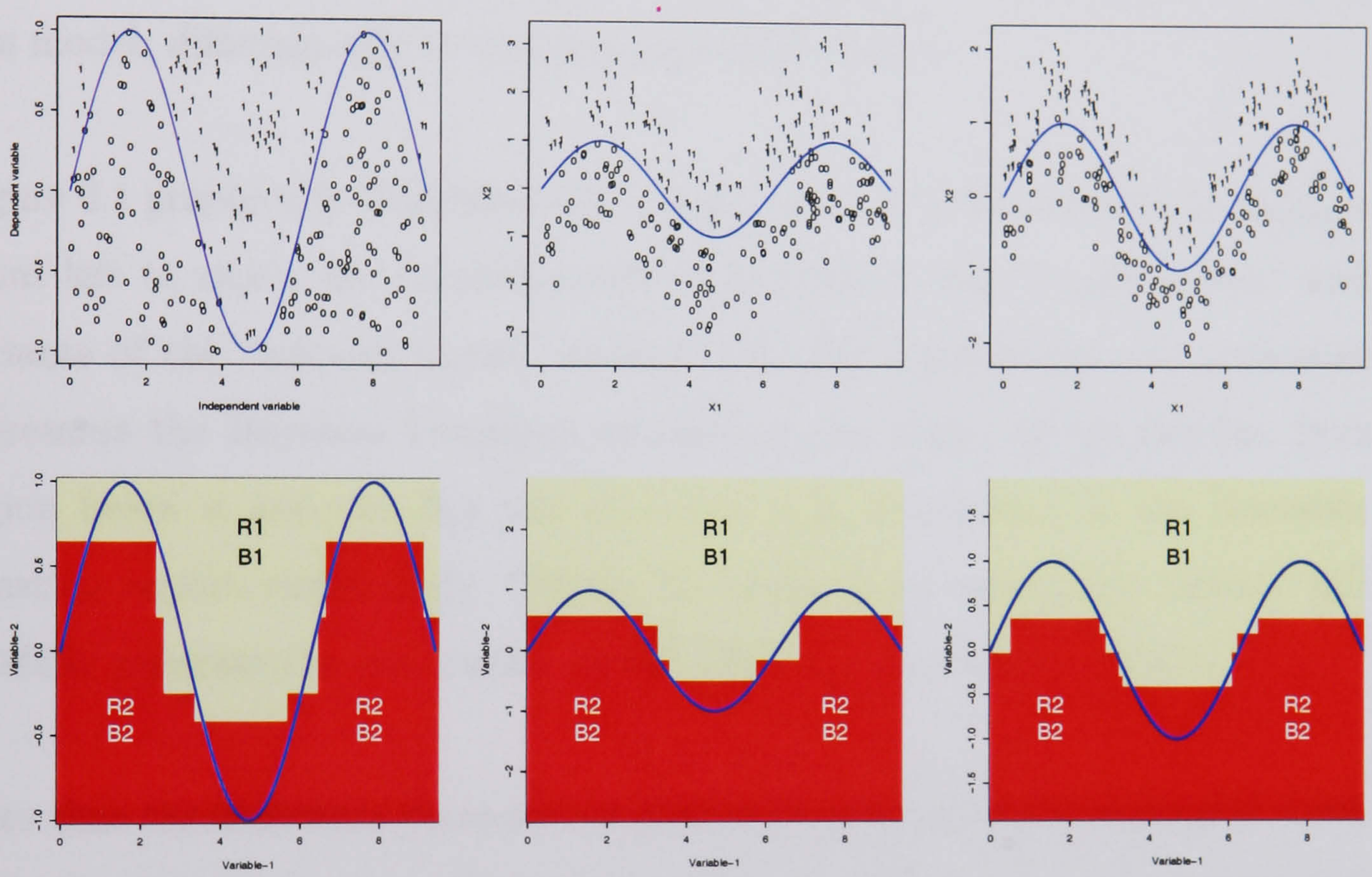


Figure 6.1: *The top and bottom rows provide, respectively, simulated data and the corresponding final partitions for the "rough" and "smoother" probability models for boosting. The LHS column-panels represent the "rough model" and the middle and RHS column-panels illustrate the two variants of the "smoother model" with different means. R1 and R2 are the Bayesian regions, above and below the sine-wave boundary, respectively. The unshaded and shaded regions correspond to the boosting regions B1 and B2 respectively. At $\xi_{B,pop}$ about zero, the middle column-panel provides a typical "counter-boosting" example — with the method performing no better than a plain stump. Typically, any linear method will be equally good.*

6.2 Design of the simulation study

For the purpose of performance assessment, we propose two probability models to boost. The first, termed the "rough model", is the piece-wise constant model of

Chapter 5. The second, termed the "smooth model", is based on a conditional normal model, so that the log-ratio of the densities is a linear function of the second variable given the first, i.e. $x_2|x_1$ — two variants of this model are considered.

The rationale behind the choice of the two models is that they provide a foundation for performance comparison. The former model is piece-wise constant; thus misclassified observations far from the boundary are no less likely to be misclassified than those near the boundary. The latter has a likelihood ratio that is a monotone function of the distance from the sine-wave boundary — we present two versions of this model, differing only in the tightness of the means.

Figure 6.1 graphically illustrates the "rough" and "smooth" models, at $\xi_{B,\text{pop}} = 5\%$. From left to right, the column-panels correspond to the "rough model" and two variants of the "smooth model" respectively. The dark continuous sine-wave line represents the Bayesian boundary, separating the region above the line from the region below it and the R_ks and B_ks, $k = 1, 2$, correspond to the Bayesian and boosting regions respectively. Clearly, no single linear separation method will successfully separate the two classes on the LHS with or without noise.

Note that the sine-wave boundary is presented as a natural challenge to the linear methods of discrimination. We vary the levels of overlap to provide further challenge to the boosting algorithm. At $\xi_{B,\text{pop}} \approx 0$, the middle panel of Figure 6.1 provides a "counter-boosting" illustration, in which boosting would typically perform no better than a plain stump or any horizontal linear discriminant rule. In this case, a single linear separation for the two classes in the middle panel is possible with $\xi_{B,\text{pop}} \approx 0\%$. Indeed, a close to zero level of noise would unambiguosly separate the two classes, which can then be split by a straight line. However, as we shall soon see, this condition ceases to hold as soon as the two classes start to overlap.

6.2.1 Formulation of the "rough" model

If we denote the areas corresponding to the R_k regions by A_k and the class labels by C_k, $k = 1, 2$, we can then write the overlapping versions of (5.7) and (5.8) as

$$\text{Given } C_1 \quad f_1(x) = \begin{cases} c_{11}/A_1 & \text{for} \quad x \in R_1 \\ & \\ c_{12}/A_2 & \text{for} \quad x \in R_2 \end{cases} \qquad c_{11} + c_{12} = 1 \tag{6.1}$$

and

$$\text{Given } C_2 \quad f_2(x) = \begin{cases} c_{21}/A_1 & \text{for} \quad x \in R_1 \\ & \\ c_{22}/A_2 & \text{for} \quad x \in R_2 \end{cases} \qquad c_{21} + c_{22} = 1 \tag{6.2}$$

respectively, where the probability density functions are piece-wise constant. To keep the Bayes' allocation rule the same, for both overlapping and non-overlapping settings, we need only to add the two constraints $c_{11} > c_{12}$ and $c_{22} > c_{21}$. In a typical setting, where $\pi_1 c_{12} < \pi_2 c_{22}$, we write down the Bayesian error as follows

$$\xi_{B,\text{pop}} = P(x \in R_2 | y = C_1)\pi_1 + P(x \in R_1 | y = C_2)\pi_2 = c_{12}\pi_1 + c_{21}\pi_2, \tag{6.3}$$

where $\pi_1 = P(y = C_1)$ and $\pi_2 = P(y = C_2)$. In the case of grossly unequal priors, the error rate will generally be the minimum of the two priors, i.e. (6.3) reduces to

$$\xi_{B,\text{pop}} = \min(\pi_1,\ \pi_2,\ \pi_1 c_{12} + \pi_2 c_{21}). \tag{6.4}$$

The formulations in (6.3) and (6.4) describe the population error as a function of the class priors and the levels of overlap. Different values of c_{12} and c_{21} represent different levels of overlap, while different values of π_ks provide different prior information on class membership. We can vary these values to see what impact they make on $\xi_{B,\text{pop}}$.

Pixelisation and calculation of the error

Consider the calculation of the boosting error in (6.11). This error is a complex function of priors, conditional probabilities, and the training data. Although boosting yields the allocation rule, $\phi(x) \rightarrow \{C_k\}$, it does not directly produce allocation regions in an analytic form. Consequently, in order to plot the allocation regions $B_k = \{x : \phi(x) = C_k\}$, we need to discretise $R_1 \cup R_2$. We calculate $\phi(x)$ at the centre of each pixel on a $c \times c$ grid, where c is a constant, and extrapolate to the whole

pixel. We then count the number of pixels above and below the sine-wave boundary, to obtain the proportion $P(x \in B_2 | y \in C_1) = \frac{\#\text{ of shaded pixels above the curve}}{\#\text{ of all pixels above the curve}}$.

The estimated misclassification error is the sum of the proportions of the unshaded pixel areas below the sine-wave line and the shaded pixel areas above the line, given the class priors π_1 and π_2. Note that the number of pixels depends on M. Based on the formulation in (6.11), we can do calculations for any shaded and unshaded regions. However, the c_{ij}s, $i, j = 1, 2$ will usually affect the resulting B_ks and, consequently, the four main components outlined above.

6.2.2 Formulation of the "smoother" model for boosting

Our second model for (y, x_1, x_2) is defined as follows. Let $x_1 \sim \mathrm{U}(0, 3\pi)$ independent of the class label $y = \pm 1$ Then given y and x_1, let $x_2 \sim N\big[\sin(x_1) + y\mu, \sigma^2\big]$. For two normal densities with μ_k and $\sigma_k^2 = \sigma^2$, $\xi_{B,\text{pop}}$ is determined by the intersection region of the two normals. In this case, the decision boundary is a sine-wave.

We use the same notation R_k as before, to denote the two regions separated by the sine-wave boundary. The two misclassification probabilities are defined as follows

$$P(x \in R_1 | y \in C_2) = \int_{R_1} f_2(x)dx = \int_0^{3\pi} \int_{\sin(x_1)}^{\infty} f^{(2)}(x_2|x_1)dx_2 f^{(2)}(x_1)dx_1 \qquad (6.5)$$

$$P(x \in R_2 | y \in C_1) = \int_{R_2} f_1(x)dx = \int_0^{3\pi} \int_{-\infty}^{\sin(x_1)} f^{(1)}(x_2|x_1)dx_2 f^{(1)}(x_1)dx_1 \qquad (6.6)$$

where $f^{(1)}$ and $f^{(2)}$ correspond to the two classes 1 and 2 respectively. Note that the inner integrals in (6.6) and (6.5) are equivalent to the two probabilities

$$P\big[x_2 < \sin(x_1) \big| y \in C_1 | x_1\big] \quad \text{and} \quad P\big[x_2 > \sin(x_1) \big| y \in C_2 | x_1\big] \qquad (6.7)$$

respectively and can be evaluated as standard normals between $\pm\mu/\sigma$ and $\pm\infty$ to correspond to the standard forms

$$P[\frac{x_2 - [\sin(x_1) + \mu]}{\sigma} < \frac{\sin(x_1) - [\sin(x_1) + \mu]}{\sigma} | y, x_1] = P(Z < -\frac{\mu}{\sigma}) = \Phi(-\frac{\mu}{\sigma}) \qquad (6.8)$$

and

$$P[\frac{x_2 - [\sin(x_1) - \mu]}{\sigma} > \frac{\sin(x_1) - [\sin(x_1) - \mu]}{\sigma} | y, x_1] = P(Z > \frac{\mu}{\sigma}) = \Phi(-\frac{\mu}{\sigma}) \qquad (6.9)$$

where Z is a standardised normal random variable. Consequently, given equal priors, $\pi = 1 - \pi = 1/2$, the model's Bayesian error can now be worked out by adding up (6.5) and (6.6), weighted by the corresponding class priors, thus yielding

$$\xi_{B,\text{pop}} = \frac{1}{2}P[Z < \frac{-\mu}{\sigma}] + \frac{1}{2}P[Z > \frac{\mu}{\sigma}] = \Phi(-\frac{\mu}{\sigma}). \tag{6.10}$$

Estimation of the population error

We propose a numerical integration method for estimating the population error, similar to (6.11). We assume equal priors, $\pi_1 = \pi_2$, thus leaving us focusing on the data variability as the main factor on the misclassification error. We provide a general consideration of "widened" and "tightened" class mean intervals by comparing results from the mean scenarios involving $\mu = \pm 1$ and $\mu = \pm 1/2$ respectively. Intuitively, the algorithm can be expected to find the latter scenario easier to handle than the former. To ensure a fair assessment of the performance of the boosting algorithm under each of the two scenarios, the Bayesian error is kept the same.

Analogous to (6.3), the boosting error, after M iterations, can be written as follows

$$\xi^M_{D,\text{pop}} = \pi_1 P(x \in B_2 | y \in C_1) + \pi_2 P(x \in B_1 | y \in C_2) = \pi_1 b_{12} + \pi_2 b_{21}, \text{ say} \tag{6.11}$$

where the B_ks denote the boosting regions shown in Figure 6.1. The regions are random, i.e. they depend on the particular training data, and, consequently, $\xi^M_{D,\text{pop}}$ is also random. Based on the assessment criterion in (4.13), we define

$$E[\Delta^*] = \{E[\xi^M_{D,\text{pop}}] - \xi_{B,\text{pop}}\}/\{E[\xi^{m=1}_{D,\text{pop}}] - \xi_{B,\text{pop}}\}, \tag{6.12}$$

expressed in percentage, as the algorithm's error reduction measure. Certainly, $E[\Delta^*] \geq 0$ and only extreme over-fitting will result in $E[\Delta^*] > 1$.

Comparing the performance of boosting on the two models requires that σ be chosen in such a way that the two errors in (6.3) and (6.10) are equal — that is, for a given error rate calculated in (6.3), we choose σ such that the same error rate is obtained in (6.10). To assess the performance of boosting under the new model we need to compare its estimated error to (6.10). Calculation of the boosting population error, after M iterations, follows the same logic as that of (6.11), only now the two

probabilities, $P(x \in B_k | y \in C_{\bar{k}})$, are a function of the distance from the boundary, with observations closest to the boundary being more likely to be misclassified than those farthest from it.

The lower middle and RHS panels in Figure 6.1 represent pictorial forms of the final partition after M boosting iterations for the "smoother model" with "widened" and "tightened" means respectively. Misclassified cases are denoted by the shaded and unshaded regions above and below the sine-wave boundary respectively. The pixels in the panels are equally spaced over the interval $[0, 3\pi]$ in the horizontal direction and over minus and plus an adopted value of σ in the vertical direction.

To calculate $\xi_{D,\text{pop}}$, we pixelise $[0, 3\pi] \times [-\mu - 3\sigma, \mu + 3\sigma]$ for the centre of each pixel on a $c \times c$ grid and extrapolate to the whole pixel. Horizontally, each pixel has width $w = 3\pi/c$ and vertically the length is $l = (6\sigma + 2\mu)/c$. We then run the boosting algorithm to classify each pixel and obtain the B_k regions.

To estimate the population error, we first calculate $f_k(x_{1i}, x_{2i})$ for each pixel, i, which is equivalent to a class membership weight. The two functions are given as

$$f_1(x_{1i}, x_{2i}) = \frac{1}{3\pi\sigma\sqrt{2\pi}} \exp\left\{ - \frac{\left(x_{2i} - \left[\sin(x_{1i}) + \mu\right]\right)^2}{2\sigma^2} \right\} \quad \text{and} \tag{6.13}$$

$$f_2(x_{1i}, x_{2i}) = \frac{1}{3\pi\sigma\sqrt{2\pi}} \exp\left\{ - \frac{\left(x_{2i} - \left[\sin(x_{1i}) - \mu\right]\right)^2}{2\sigma^2} \right\}. \tag{6.14}$$

Each of the resulting probabilities is then multiplied by the appropriate pixel area to obtain a volume for that particular observation. We can then approximate the level of overlap by $P(x \in B_k | y \in C_{\bar{k}})$. The estimated boosting misclassification error, after M iterations, can then be written as

$$\xi^M_{D,\text{pop}} = \pi_1 \sum_{x_{.i} \in B_2} f_1(x_{1i}, x_{2i}) * a + \pi_2 \sum_{x_{.i} \in B_1} f_2(x_{1i}, x_{2i}) * a \tag{6.15}$$

where $a = w \times l$ denotes the pixel area. The area for each pixel is multiplied by the appropriate pixel height and then summed up to yield the volumes. The population error estimate is obtained by summing up the volumes, weighted by the priors.

6.3 Results from boosting the two models

In this section we present boosting results for the two models based on the factors outlined in Section 6.1. We make a two-fold comparison of the performance of boosting by first comparing the two models then looking at the inter-model comparison based on the "tightness" of the means. In both cases, we are interested in how fast boosting drives the error down and how close its estimate gets to the Bayesian error.

6.3.1 The "rough" model

In this section we look at the impact of fixed and variable c_{ij}s, balanced and imbalanced π_ks and fixed and optimal M. The case of fixed M is trivial, as we just pick a value and compare the performance for different c_{ij}s. Optimal M depends on unavailable knowledge, in much the same way as does $\xi_{B,\text{pop}}$. However, both are still useful benchmarks in investigation. We start off by considering the effects of $c_{12} = c_{21} = \xi_{B,\text{pop}}$, the sample size n, class priors, $\pi_1 = 0.39$ and $\pi_2 = 0.61$ (obtained as a consequence of simulation) and fixed M on the measure $E[\Delta^*]$. We investigate

1. What happens to $E[\Delta^*]$ as $\xi_{B,\text{pop}}$ goes up.

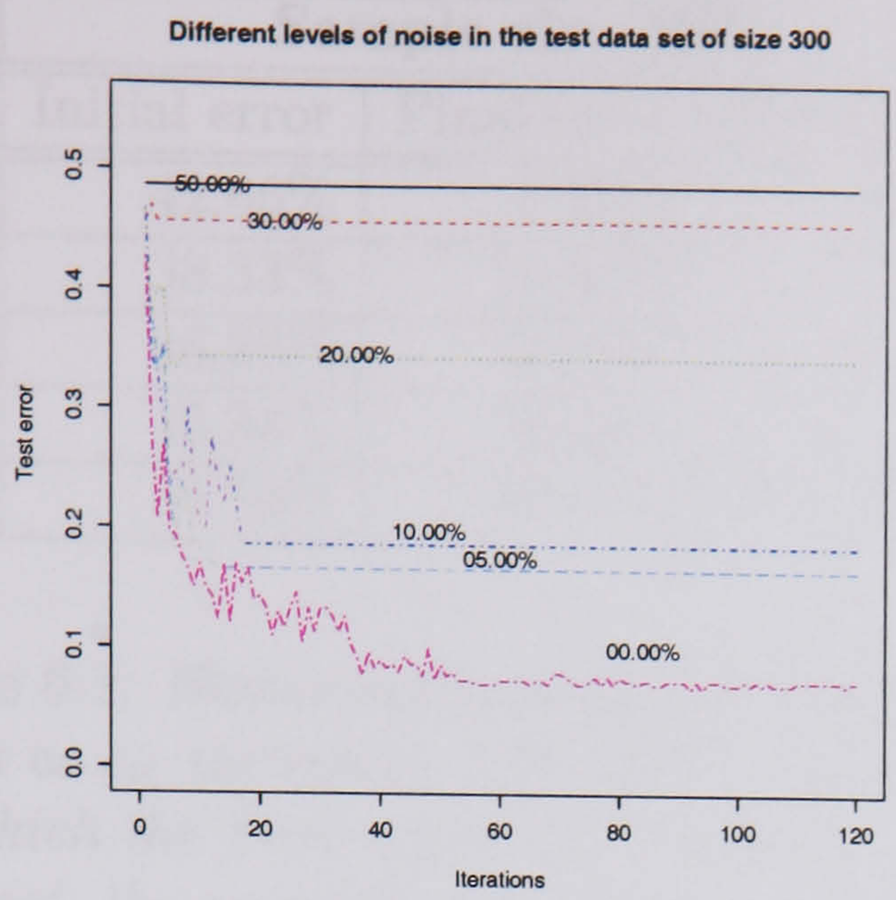
2. The behaviour of both the initial and final errors.

Data simulation is based on the densities (6.1) and (6.2). The chosen π_ks are, in fact, a consequence of the data simulation. We mainly use two different data sizes $n = 300$ and $n = 60$, but to rule out the sample-specific behaviour, we also use $n = 5000$. The data points $x_i,\ i = 1, ..., n$ are simulated uniform over the area defined as Variable-1 × Variable-2, in Figure 6.1, with a sine-wave boundary.

The case of fixed M and fixed imbalanced priors

In this example, we set $M = 120$ and proceed by systematically inflating $\xi_{B,\text{pop}}$, from 0% through 30%, inclusive. The main purpose is to observe the behaviour of boosting, in particular the number of iterations necessary to optimise the misclassification error and the resulting initial and final error patterns for both datasets.

Boosting results, based on the foregoing settings, are presented in Figure 6.2 and summarised in Table 6.1 for the two datasets. Some of the boosting lines in Figure



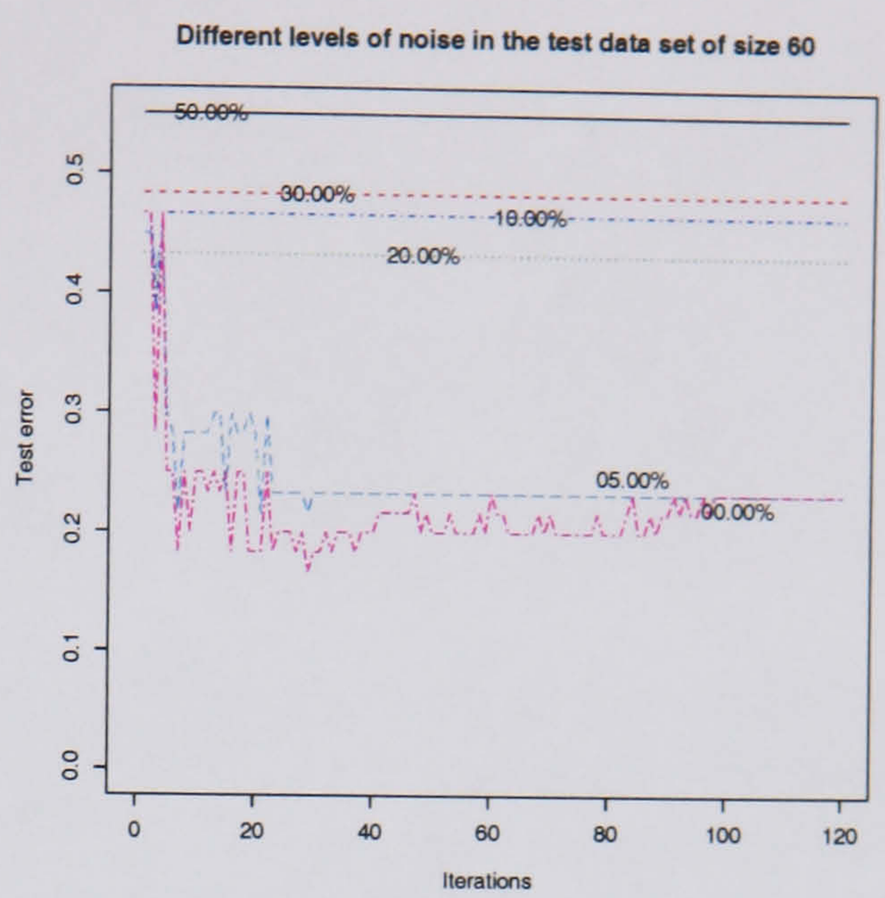


Figure 6.2: *Test error plots associated with different amounts of noise in test data of sizes $n = 300$ and $n = 60$ on the LHS and RHS respectively. In both cases, the algorithm was trained and tested on datasets having the same error distribution. Sampling variation is clearly higher in the smaller dataset.*

6.2 look anomalous. In particular, the misclassification error for the larger dataset without overlap is 42.66% at the first iteration but drops by 34.36% to 28.00% in the next iteration. Similarly, misclassification error for the same set, with 30.00% added noise, increases from 45.66% to 47.00% in one iteration, then reverts to the initial value in the next iteration. Indeed, as we noted earlier, boosting "struggles" as the level of noise goes up. The misclassification error for the smaller dataset with 10.00% noise, starts at 46.66%, drops to 38.33% in three iterations, then goes on to stabilise at the same value of 46.66% in the next iteration.

Note that the majority of the lines in both panels of Figure 6.2 did not need multiple iterations, i.e. a large M is not necessarily useful to the algorithm. Such circumstances are common, for small data sets, when classes overlap and a general remedy would usually be to stop boosting early. In the next exposition we do investigate this problem further, by running multiple simulations, to see if the foregoing findings are typical.

Optimal M and fixed imbalanced priors

Can we obtain an optimal value for M, for all simulations, that minimises misclassification? Keeping the same settings, as in the foregoing experiment, we run 50 simulations of each for four of the six scenarios, namely, the no-overlap and the

$c_{12} = c_{21}$	**Sample size 300**			**Sample size 60**		
	Initial error	Final error	Iterations	Initial error	Final error	Iterations
0%	42.66%	7.33%	75	42.66%	23.33%	99
5%	38.33%	16.66%	13	45%	23.33%	23
10%	46.66%	18.66%	17	46.66%	46.66%	1
20%	46.33%	34.33%	7	43.33%	43.33%	1
30%	45.66%	45.66%	1	48.33%	48.33%	1

Table 6.1: *Numerical summary of Figure 6.2. Note the decreasing number of iterations as c_{ij} increases. The entries in columns 4 and 7 correspond to the value of m at which the final error was recorded — for the zero-overlap scenario of the larger dataset, the recorded error remains unchanged as m increases — but not in the case of $n = 60$. For all values of $\xi_{B,pop}$, the smaller sample yielded a higher final error.*

05.00%, 10.00% and 20.00% levels of overlap. We reduce the number of iterations to 100, instead of the 120 used earlier. We then proceed by averaging the errors over all simulations as follows. We denote the error at the m^{th} iteration of the s^{th} simulation by $\xi_{s,m}$, where $s = 1, ...S$. Its average over all simulations is

$$\frac{1}{S}\sum_{s=1}^{S}\xi_{s,m}. \tag{6.16}$$

The averaged lines for each of the c_{ij} scenarios are shown in Figure 6.3, for $S = 50$. The figure is a multiple simulation version of Figure 6.2, for the same levels of overlap. The error patterns in the figure smoothly stabilise, thus readily providing the average initial and final errors, as summarised in Table 6.2. Note the dramatic change in the number of iterations, particularly for the higher levels of overlap.

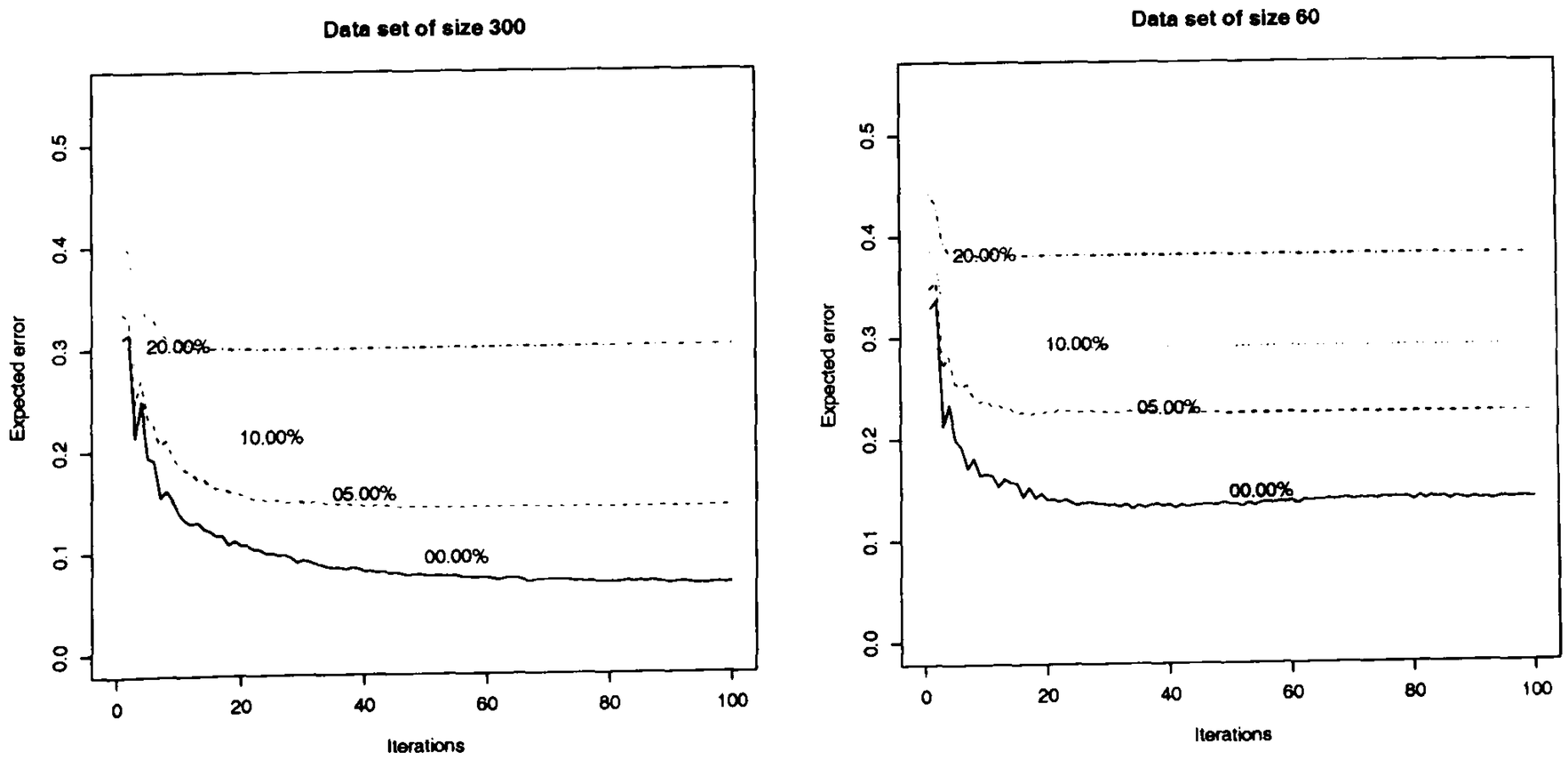


Figure 6.3: *The two panels are a simulation version of Figure 6.2, obtained by averaging over 50 simulations. The higher level overlap, $c_{ij} = 30.00\%$, has been omitted. Note the over-fitting behaviour of the bottom line in the RHS panel.*

$c_{12} = c_{21}$	**Sample size 300**			**Sample size 60**		
	Initial error	Final error	Iterations	Initial error	Final error	Iterations
0%	31.12%	6.6%	84	32.93%	13%*	35
5%	33.5%	14.2%	55	34.83%	22.33%	22
10%	35.68%	20.5%	35	38.5%	29%	18
20%	40.39%	30%	20	44.16%	38%	8

Table 6.2: *The table provides a numerical summary of Figure 6.3. The entries are obtained by averaging the error estimates over 50 test data simulations. The asterisk indicates that the error goes up again after that point. This is particularly interesting as it exhibits over-fitting for the smaller sample size with no overlap.*

It is clearly evident, in both cases, that both the initial and final boosting errors are an increasing function of the level of overlap, as opposed to the number of iterations, which change inversely with it. At the same time, as the level of overlap increases, the error reduction power of boosting deteriorates. Table 6.3, provides a summary of the algorithm's error reduction measure for the given two sets of data and c_{ij}s. Both c_{ij} and data size clearly appear to have an impact on the error reduction rate.

$c_{12} = c_{21}$	**Error reduction - 300**	**Error reduction - 60**
00.00%	21.21%	39.47%
05.00%	32.28%	58.09%
10.00%	40.88%	66.66%
20.00%	49.04%	74.50%

Table 6.3: *The error reduction measure, $E[\Delta^*]$, is computed in percentage for the different values of c_{ij}. As to be expected, the measure increases with c_{ij} in both cases, but assumes larger values when $n = 60$ than in the case of $n = 300$.*

Figure 6.4 provides a more comprehensive analysis of the effect of inflating c_{ij}, involving the two datasets. The top row panels, left-to-right, correspond to the bottom-up plots of the LHS panel of Figure 6.3, while the bottom row of Figure 6.4 corresponds to the bottom-up plots of the RHS panel in Figure 6.3. Further, ± 1 standard deviation lines have been added to each of the plot, and clearly show how the variation increases horizontally, as the level of overlap increases, and vertically, as the training dataset becomes smaller.

If M is optimally chosen for each of the two datasets, the plot of $\xi_{B,\text{pop}}$ against $E[\Delta^*]$ would be informative — graphical relationships of the two quantities, for different sample sizes, appear in Section 6.4.1, in which we compare the two models. A slightly

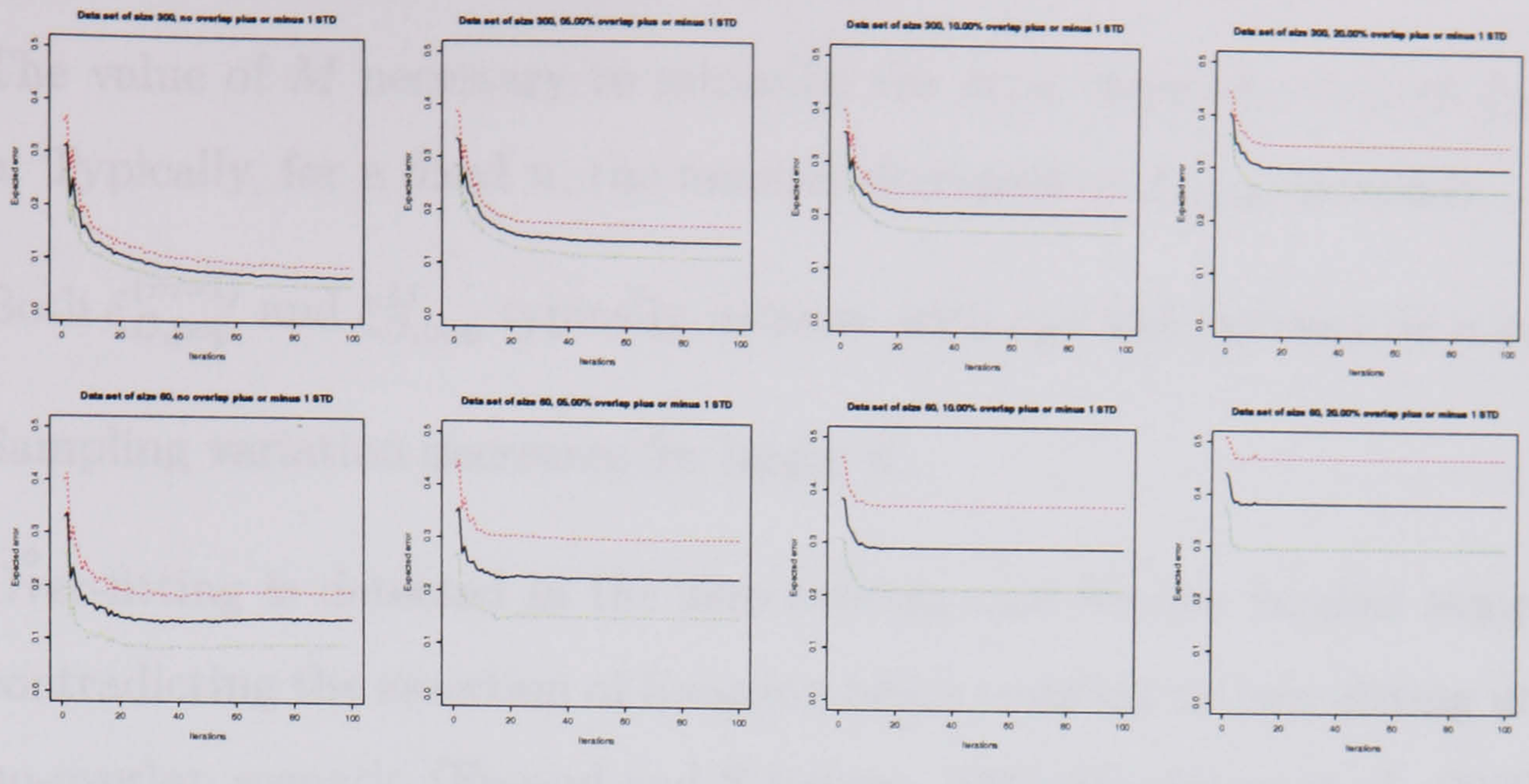


Figure 6.4: *The top row panels represent the simulation averaged boosting lines for the dataset of size 300, while the bottom row panels are based on the smaller dataset of size 60. In both cases, the c_{ij}s in Table 6.2, top-down, are in the left-to-right direction and ± 1 standard deviation lines have been added. Variation increases horizontally, as c_{ij} increases and vertically as the training dataset decreases. Left-to-right the panels correspond to levels of overlap 0%, 5%, 10% and 20% respectively.*

different approach would be to extract the minimum values and the corresponding number of iterations for each course of boosting over all the simulations, i.e., find

$$m_s^* = \arg\min_m \xi_{s,m}$$

and average the minima to obtain the "final error" of boosting as $\frac{1}{S}\sum_{s=1}^{S}\xi_{s,m_s^*}$. It seems reasonable to adopt this quantity as the algorithm's final error.

To determine whether or not the foregoing behaviour was a consequence of the sample size, we ran multiple simulations for a much larger sample size, $n = 5000$ and $c_{12} = c_{21} = 10\%$. Compared to the results based on the same error rate and $n = 300$, the performance in this case was much better, resulting into a decrease in both the initial and final error estimates to 33.3% and 15.92% respectively, yielding $E[\Delta^*] = 25.41\%$ — a dramatic improvement in the algorithm's error reduction measure and a confirmation of its dependence on both $\xi_{B,\text{pop}}$ and n. However, the improvement was attained at the cost of increased number of iterations to about 80.

Summary of some important observations

The following are a number of observations deriving from the examples in the section.

1. For fixed n, the performance of boosting deteriorates with increasing $\xi_{B,\text{pop}}$ and the plot of $\xi_{B,\text{pop}}$ against $E[\Delta^*]$ will monotonically increase with $\xi_{B,\text{pop}}$.

2. The value of M necessary to minimise the error depends on both $\xi_{B,\text{pop}}$ and n. Typically, for a fixed n, the number decreases as $\xi_{B,\text{pop}}$ increases.

3. Both $\xi_{D,\text{pop}}^{(m=1)}$ and $\xi_{D,\text{pop}}^{M}$ typically increase with c_{ij}s and decrease as n increases.

4. Sampling variation decreases for larger n.

5. Over-fitting is detected in the zero overlap case for the smaller sample case, contradicting the assertion of boosting being resistant to over-fitting under the no-overlap scenario (Freund and Schapire, 1997; Friedman *et al.*, 2000).

6. Increasing n improves $E[\Delta^*]$, by reducing both the initial and final error estimates. Further, a large n also entails a large M.

Constant priors and variable overlap

Again, we use the class priors $\pi_1 = 0.39$ and $\pi_2 = 0.61$, the R_ks and C_ks are the predicted and true classes respectively as defined earlier in Chapter 4 Section 4.1. The total number of observations is denoted by n and n_k, $k = 1, 2$, are the total number of data points belonging to region R_k and n_{ij} denotes the number of cases from class i that are currently classified as class j. Thus, the quantity n_{11}, representing the C_1-labelled data points in R_1, can then be obtained by rule (6.1), i.e. by recognising that $c_{11} + c_{12} = 1$, which implies that n_{11} corresponds to c_{11}. Consequently, $n_{12} = n_1 - n_{11}$ and $n_1 = n_{11} + n_{12}$. That is, n_{12} cases belonging to class 2 will typically be classified to class 1. The C_2-labelled data in R_2 and R_1 can be obtained in exactly the same way. The number of cases above and below the boundary are $n_1 = n_{11} + n_{12}$ and $n_2 = n_{22} + n_{21}$ respectively.

Clearly, given n_k, the Bayesian error is a function of the c_{ij}s through the n_{ij}s. Similarly, given a fixed $c_{12} = c_{21}$ and variable π_ks, we can monitor the behaviour of the estimated error as a function of the priors. In this example, given the class priors, π_ks, we explore the impact of six different c_{ij}s between two extreme cases, as we keep the Bayesian error fixed at $\xi_{B,\text{pop}} = 10.00\%$.

Table 6.4 summarises results from the six different overlapping scenarios, for a training dataset of size 300 with fixed $\xi_{B,\text{pop}} = 10.00\%$. It provides estimates of $\xi_{B,\text{pop}}$

by boosting with tree stumps as well as $E[\Delta]^*$, for each set of the c_{ij}s. Note that the entries in columns 1 and 2 are based on $\xi_{B,\text{pop}} = 10.00\% = c_{12}\pi_1 + c_{21}\pi_2$, with $\pi_1 = 0.39$ and $\pi_2 = 0.61$, while those in columns 3, 4 and 5 were obtained by averaging over 50 different simulations. If we hold $\xi_{B,\text{pop}}$ fixed, we can compute the average value of delta over a number of samples. Typically, we would expect the quantity $E[\Delta]^*$ to be larger in the case of the smaller than in the larger dataset.

c_{12}	c_{21}	$\xi_{D,\text{pop}}^{(m=1)}$	$\xi_{D,\text{pop}}^{(M=250)}$	$E[\Delta^*]$
0%	16.39%	34.39%	15%	20.50%
5%	13.20%	34.26%	14.8%	19.79%
7%	11.92%	33.98%	14.56%	19.02%
14.69%	7%	33.42%	13.8%	16.23%
17.82%	5%	33.46%	12.9%	12.36%
25.64%	0%	31.28%	11.5%	7.05%

Table 6.4: *The table displays various scenarios of class overlap, c_{ij}s, and fixed class priors, π_ks, for a training dataset of size 300 with fixed $\xi_{B,pop} = 10.00\%$. The discernible pattern in columns 3, 4 and 5 is attributable to the nature of the π_ks.*

It can be seen from Table 6.4 that a clear pattern emerges as c_{12} goes up and c_{21} goes down. The composite effects of the imbalanced priors and c_{ij}s lead to the clear trend in the initial and final errors as well as the expected reduction measure. Figure 6.5 provides boosting results, for each of the scenarios in Table 6.4, averaged over 50 simulations. The larger sample performs better than the smaller, both in terms of closeness to the Bayesian error and resistance to over-fitting. It is quite evident, that in the case of the smaller dataset, it would be wise to stop boosting early. In this case, the choice of $M < 50$ seems plausible.

Again, we can look at the error pattern variation for the two training sets. Figure 6.6 exhibits a side-by-side comparison of the error variability between the two datasets, as the level of overlap goes up. With fixed priors and a fixed $\xi_{B,\text{pop}} = 10.00\%$, variability in the larger dataset is almost unaffected by the changing c_{ij}. With $n \to \infty$, we can expect the variability to disappear. Conversely, as n becomes increasingly small, we can expect a large variability, both within and between overlap scenarios.

The emerging patterns in Table 6.4 and Figure 6.5, suggest that a higher error proportion of error arising from class two, bears more serious consequences to the

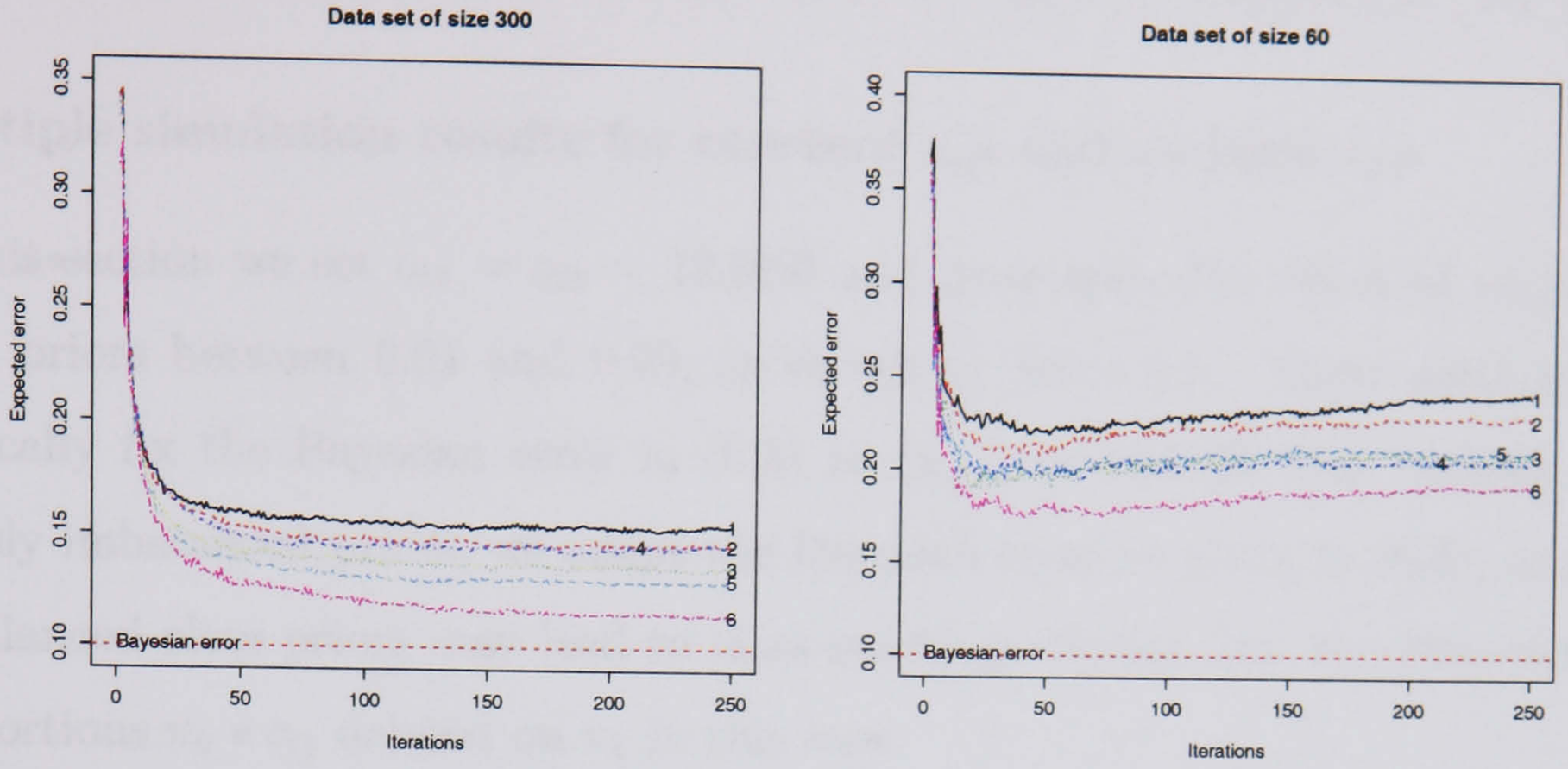


Figure 6.5: *The LHS and RHS panels exhibit boosting results for* $n = 300$ *and* $n = 60$ *respectively, based on Table 6.4. The numbered lines correspond to the rows of the table, top-down. Clear patterns of over-fitting on the RHS imply that boosting would best be stopped after about 50 iterations. Note that all boosting scenarios on the LHS, get closer to the Bayesian error that any of the scenarios on the RHS.*

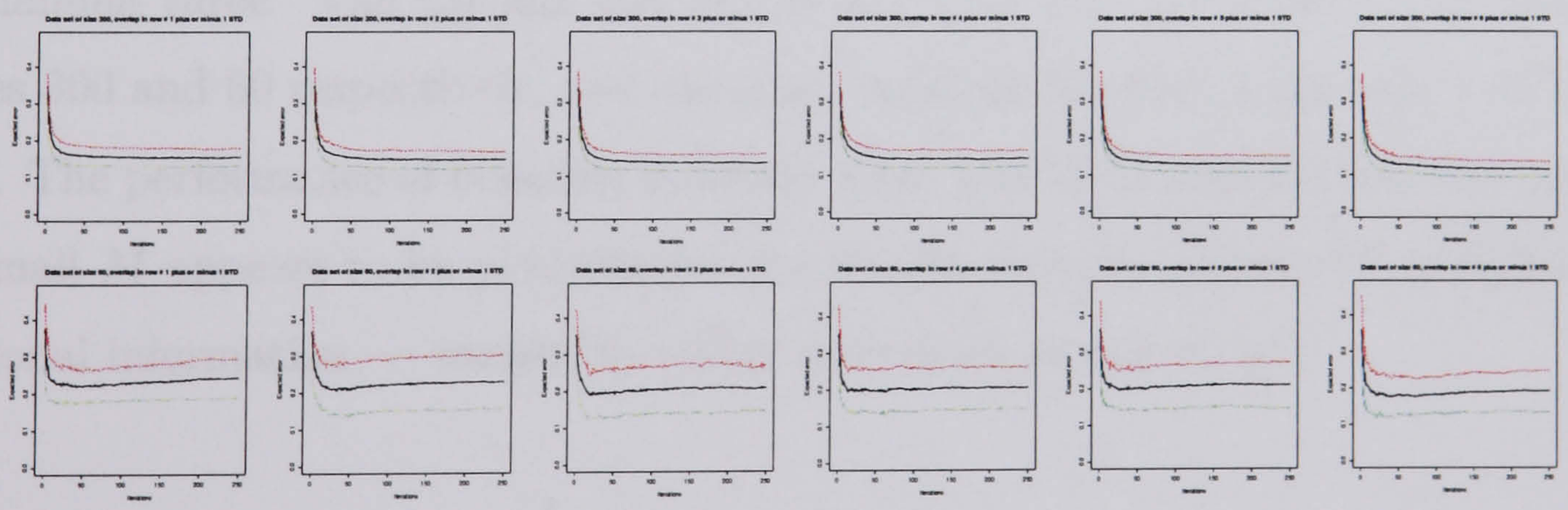


Figure 6.6: *The upper panels correspond to the boosting error patterns of rows 1 through 6 in Table 6.4, with two* ± 1 *standard deviation lines added. The lower panels are based on the smaller dataset of size 60 with the same error scenarios. Variability remains constant, with constant* $\xi_{B,pop}$*, particularly in the case of the larger dataset. The smaller dataset exhibits some variability, both within and between scenarios.*

estimated error than the other way round. In other words, the scenario is more favourable to a low level scenario for c_{21} than for c_{12}. Two possible causes of this situation are the nature of the decision boundary and the imbalanced class priors.

Multiple simulation results for constant c_{ij}s and variable π_ks

In this section we set $c_{12} = c_{21} = 12.00\%$ and investigate the effect of varying the class priors between 0.01 and 0.99, as shown in Table 6.5. These settings automatically fix the Bayesian error in (6.3) to $\xi_{B,\text{pop}} = 12.00\%$. But because of the grossly imbalanced priors, we adapt the Bayesian error as given in (6.4), as grossly imbalanced class priors may lead to class masking. Notice how the Bayesian error proportions $\pi_k * c_{ij}$ depend on π_k in this case.

π_1	$\min(\pi_1, \pi_2, \pi_1 c_{12} + \pi_2 c_{21})$
1%	1%
30%	12%
50%	12%
60%	12%
99%	1%

Table 6.5: *Class overlap is kept constant at $c_{12} = c_{21} = 12.00\%$ but the π_ks are varied between 01.00% and 99.00% on either sides of the sine-wave decision boundary.*

For comparison purposes, we separate the first and final rows of Table 6.5 from the remaining three. The LH and RH panels in Figure 6.7 correspond to datasets of sizes 300 and 60 respectively, and are both based on the rows 2 through 4 of Table 6.5. The performance of boosting is clearly good with the larger dataset and, again, a small M appears to be plausible for the smaller dataset. Figure 6.8 provides additional information — variability stably goes down as n goes up.

The case of variable $c_{ij}s$ and balanced priors

In this section, we consider the scenario of a fixed $\xi_{B,\text{pop}} = 12.00\%$, balanced priors and variable $00.00\% \leq c_{ij} \leq 24.00\%$. With balanced priors the error becomes solely a function of the densities, making the assessment of the impact of the c_{ij}s on the error more sensible. But for the priors and the imbalanced c_{ij}s, this scenario is similar to the previous one. Parameters of interest include M, the initial and final

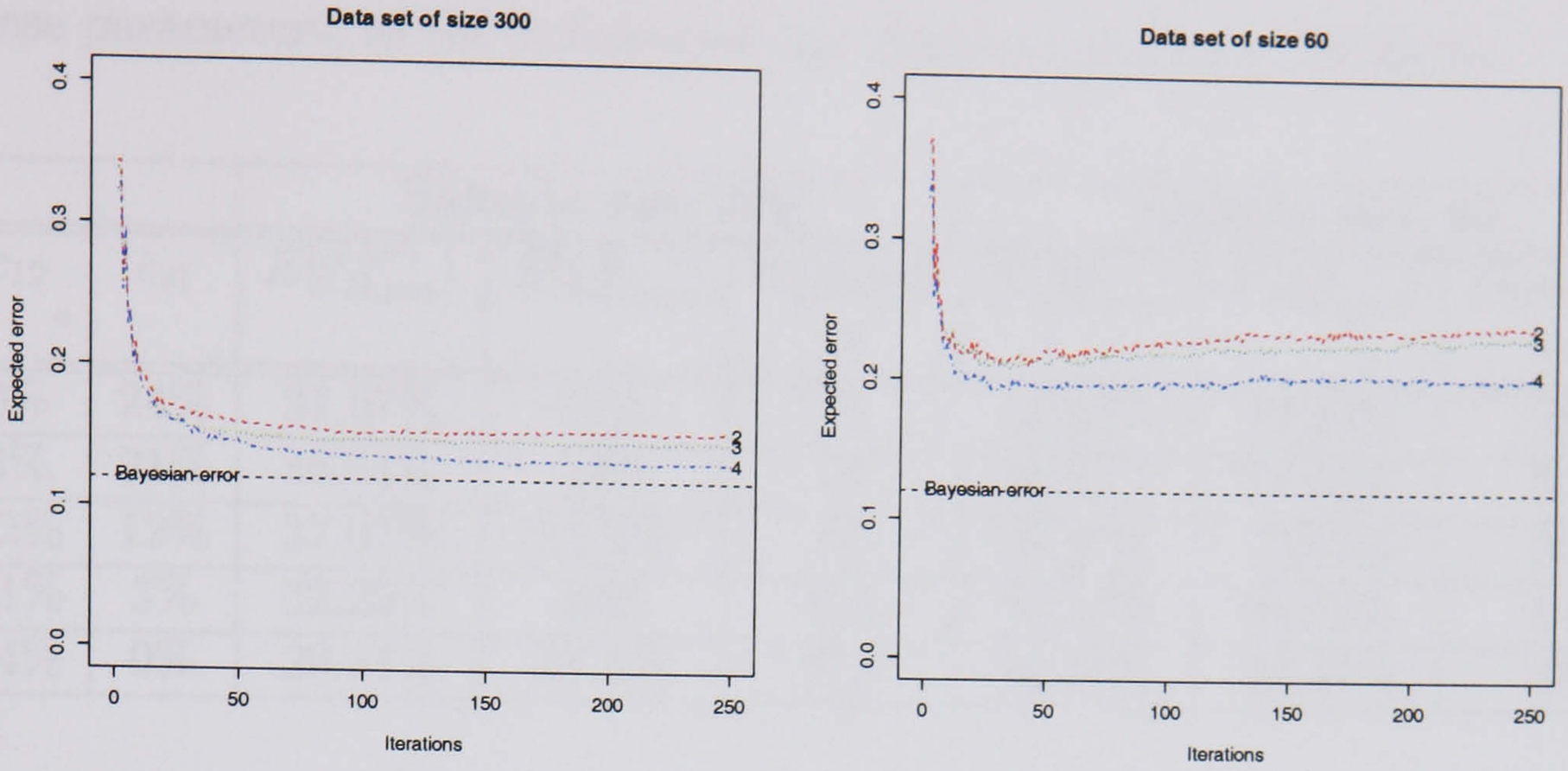


Figure 6.7: *Top-down, the lines correspond to the rows 2 through 4 of Table 6.5. The emerging over-fitting pattern for the smaller sample case, suggests small M.*

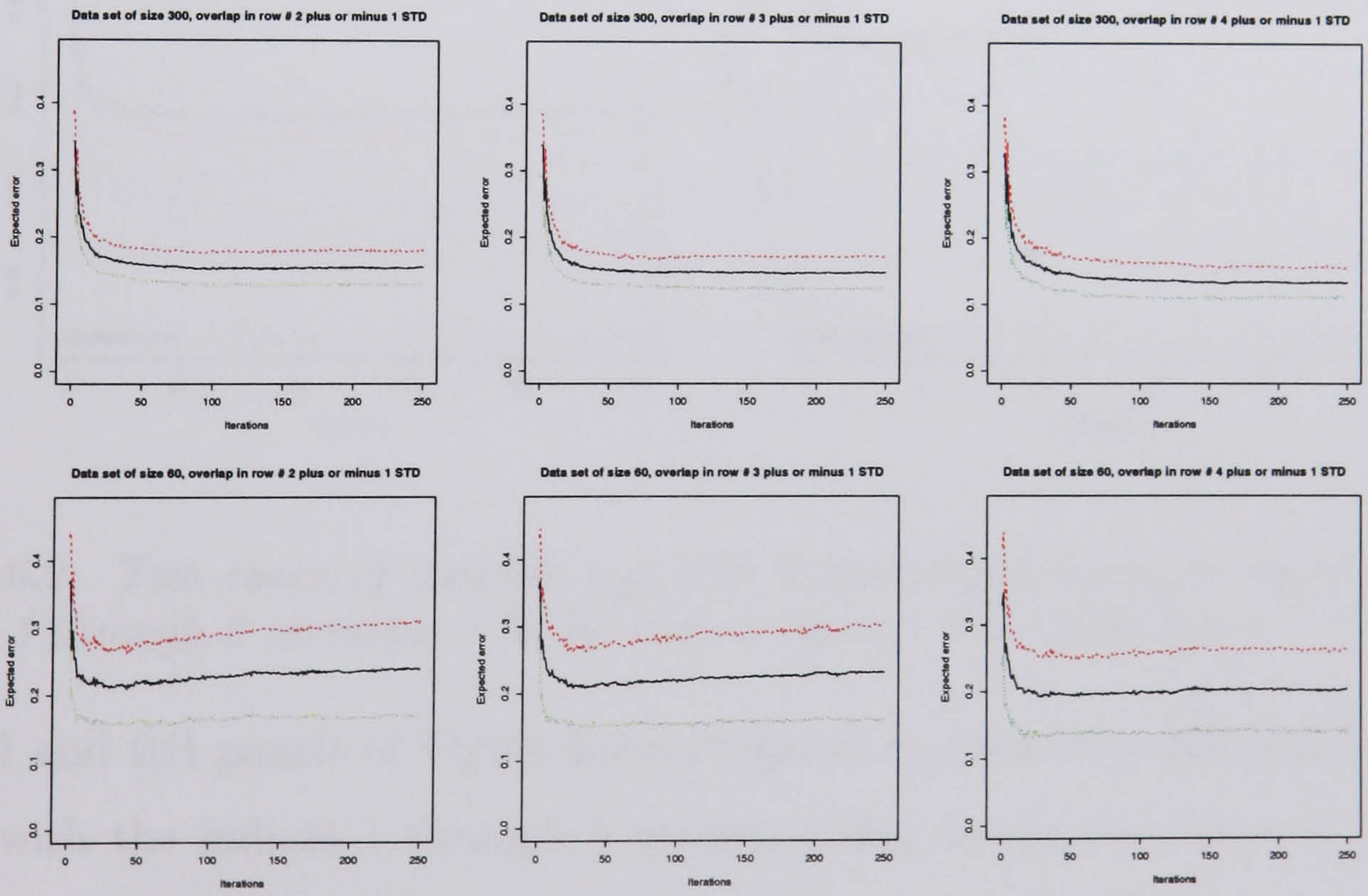


Figure 6.8: *The top row panels correspond to the larger dataset, based on the scenarios in rows 2 through 4 in Table 6.5. The bottom row panels are based on the smaller dataset, with the same settings. Again, column-wise comparison clearly reveals that variability is more noticeable in the bottom than in the top row.*

error estimates for each simulation, $\xi^M_{D,\text{pop}}$ and $E[\xi^M_{D,\text{pop}}]$ over all simulations and the algorithms error reduction measure. We wish to investigate the behaviour of some of these parameters, as the imbalanced c_{ij}s change as shown in Table 6.6.

			Sample size 300			**Sample size 60**		
Row	c_{12}	c_{21}	$E[\xi^{m=1}_{D,\text{pop}}]$	$E[\xi^M_{D,\text{pop}}]$	Optimal M	$E[\xi^m_{D,\text{pop}}]$	$E[\xi^M_{D,\text{pop}}]$	Optimal M
1	0%	24%	37.07%	24%	72	38.43%	27.43%	35
2	3%	21%	36.51%	23%	76	38.50%	27.63%	30
3	12%	12%	37.07%	21.5%	75	38.97%	26.57%	30
4	21%	3%	32.25%	19%	110	35.77%	24.03%	14
5	24%	0%	29.31%	17.5%	215	32.87%	20.47%	35

Table 6.6: *Selected scenarios of c_{ij} for a fixed population error of $\xi_{B,pop} = 12.00\%$ and balanced class priors, $\pi_k = 50.00\%$. Note the comparatively higher effect of c_{21} on the initial and final errors, particularly for the larger dataset. The smaller dataset exhibits high variability in almost all the parameters of interest.*

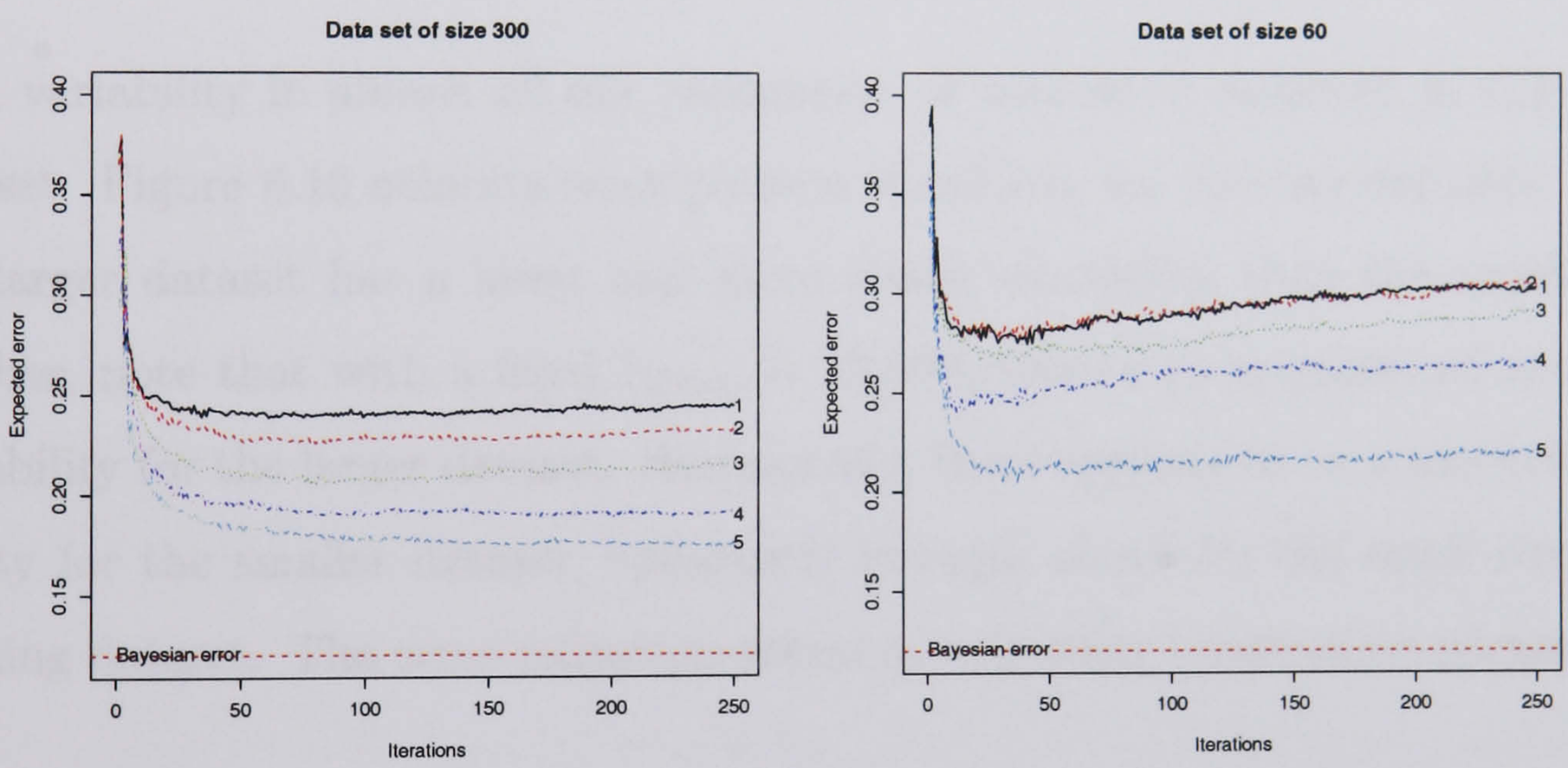


Figure 6.9: *Two cases of variable c_{ij}s with balanced priors, $\pi_k = \pi_{\bar{k}} = 0.5$. The indices 1 through 5 correspond to the rows 1 through 5 in Table 6.6.*

The LH and RH panels of Figure 6.9 correspond to data sizes 300 and 60 respectively, with the indices 1 through 5 corresponding to the five rows of Table 6.6 top-down. Each of the selected scenarios, was averaged over 50 simulations. The lines in each case exhibit the expected error patterns, based on the expected values for all $m = 1$ through $M = 250$ over all simulations. All these multiple simulation plots can clearly be seen to be smoother than the plots based on single boosting cycles seen earlier in Figure 6.2. The summary in Table 6.6 and the nature of the sine-wave boundary together indicate that the overlap c_{21} has more influence on our

parameters of interest than c_{12}. Note, in the case of the larger dataset, how c_{21} grows with both the initial and final errors and inversely with M.

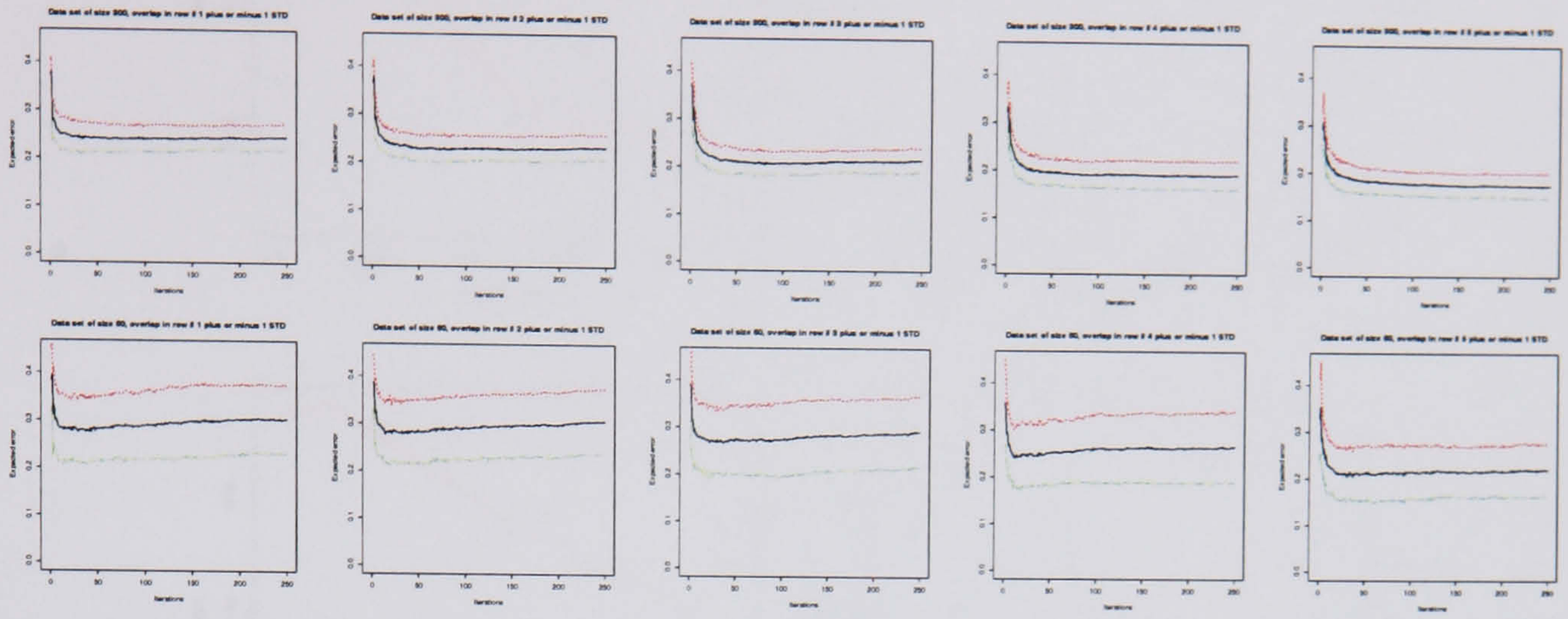


Figure 6.10: *The two plot-columns correspond to the two scenarios of variable c_{ij}s with balanced priors, $\pi_k = \pi_{\bar{k}} = 0.5$ as given in Table 6.6. Left-to-right the panels correspond to the table's second and third columns top-down. The top row panels, with low and stable variability, represent the larger dataset while the bottom row panels correspond to the smaller dataset. Note the high variability in the latter case.*

High variability in almost all our parameters of interest is observed in the smaller dataset. Figure 6.10 exhibits error pattern variability for the two datasets. Again, the larger dataset has a lower and more stable variability than the smaller one. Further, note that with a fixed $\xi_{B,\text{pop}} = 12.00\%$ there's no pronounced horizontal variability for the larger dataset. Horizontally, there appears to be a moderate variability for the smaller dataset, apparently brought about by the small size of the training dataset. The error reduction measure is another informative parameter.

Figure 6.11 depicts two mirror-image plots of the relationship between the c_{ij}s and $E[\Delta^*]$. The top two panels correspond to the larger dataset of size $n = 300$ while the bottom two represent the smaller dataset. In each case, the plotted relationships, left to right, are for $E[\Delta^*]$ against c_{12} and c_{21} respectively.

6.3.2 The "smoother" model

This section presents boosting results for the two variants of the "smooth model", i.e., the "widened" and "tightened" means. The estimation of the population error, for $\mu = \pm 1$, was done for both the 300 and 60 data point samples. An image grid of 200×200, each defining a pixel and various values of σ yielding overlaps between 0%

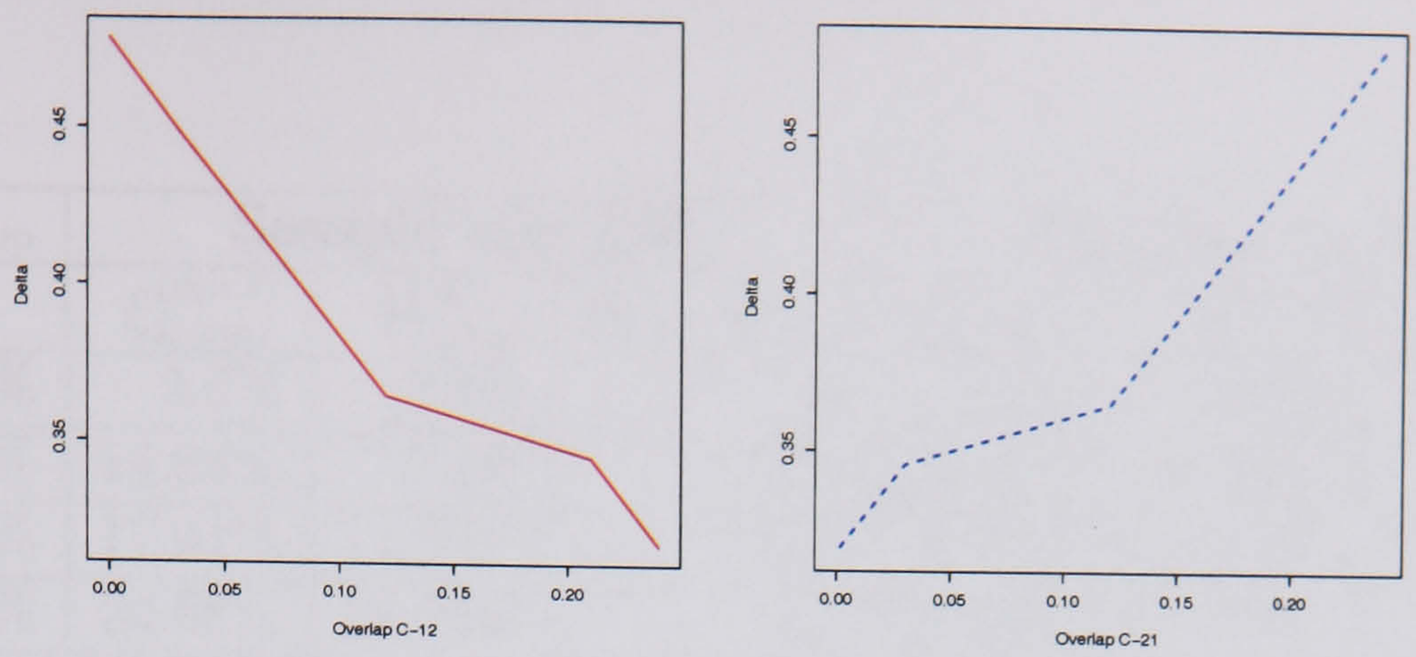


Figure 6.11: *The LHS and RHS column-panels display mirror-image relationships between $E[\Delta^*]$ and c_{12} and c_{21} for the larger and smaller datasets respectively. The mirror-images are due to the fact that the overlaps are plotted against Delta.*

and 20% were used. The area for each pixel was multiplied by the appropriate pixel height and then summed up to yield the volumes, overlaps as well as the population error estimate. Table 6.7 summarises the results for single samples by giving the initial and "best" final error estimates as well as the number of iterations to obtain the minimum error. Note the relationship between the number of iterations and σ. The two panels in Figure 6.12 graphically reproduce the results of line 2 in Table 6.7.

For a fixed n, low levels of σ are associated with a large number of iterations, decreasing as σ increases. Particularly for the smaller sample size, boosting beyond the "best" error levels leads to over-fitting. For instance, the error after $M = 100$ and $\sigma = 1.19$ is 28%, even higher than the initial estimate — that is, $E[\Delta]^* > 1$. Experiments with much larger values of σ confirmed that as σ increases, $M \to 1$.

The case of tightened means interval

In this example we tighten the means by setting $\mu = \pm 1/2$, which means that it is now impossible to attain a pure partition by a single horizontal split as was the case

$(\sigma) \Rightarrow \xi_{B,\text{pop}}$	**Sample size 300**			**Sample size 60**		
	$\xi_{D,\text{pop}}^{(m=1)}$	$\xi_{D,\text{pop}}^{(M)}$	Iterations	$\xi_{D,\text{pop}}^{(m=1)}$	$\xi_{D,\text{pop}}^{(M)}$	Iterations
$(0.008) \Rightarrow 0\%$	2.4%	2.4%	1	2.4%	2.4%	1
$(0.608) \Rightarrow 5\%$	14.88%	6.1%	72	18.74%	10.68%*	10
$(0.781) \Rightarrow 10\%$	17.01%	11.39%	65	18.98%	14.66%*	12
$(1.19) \Rightarrow 20\%$	24.58%	22.05%	50	25.82%	22.16%*	5

Table 6.7: *Summary of pixelisation with boosting used in estimating the population error for the $\mu = \pm 1$ scenario. Except for the noise-free scenario, low levels of overlap are associated with a large number of iterations, comparatively higher in the larger than in the smaller sample size and drops as $\sigma \to \infty$. The asterisk indicates that the error estimate goes up again after that point, suggesting a relationship between over-fitting and the data size.*

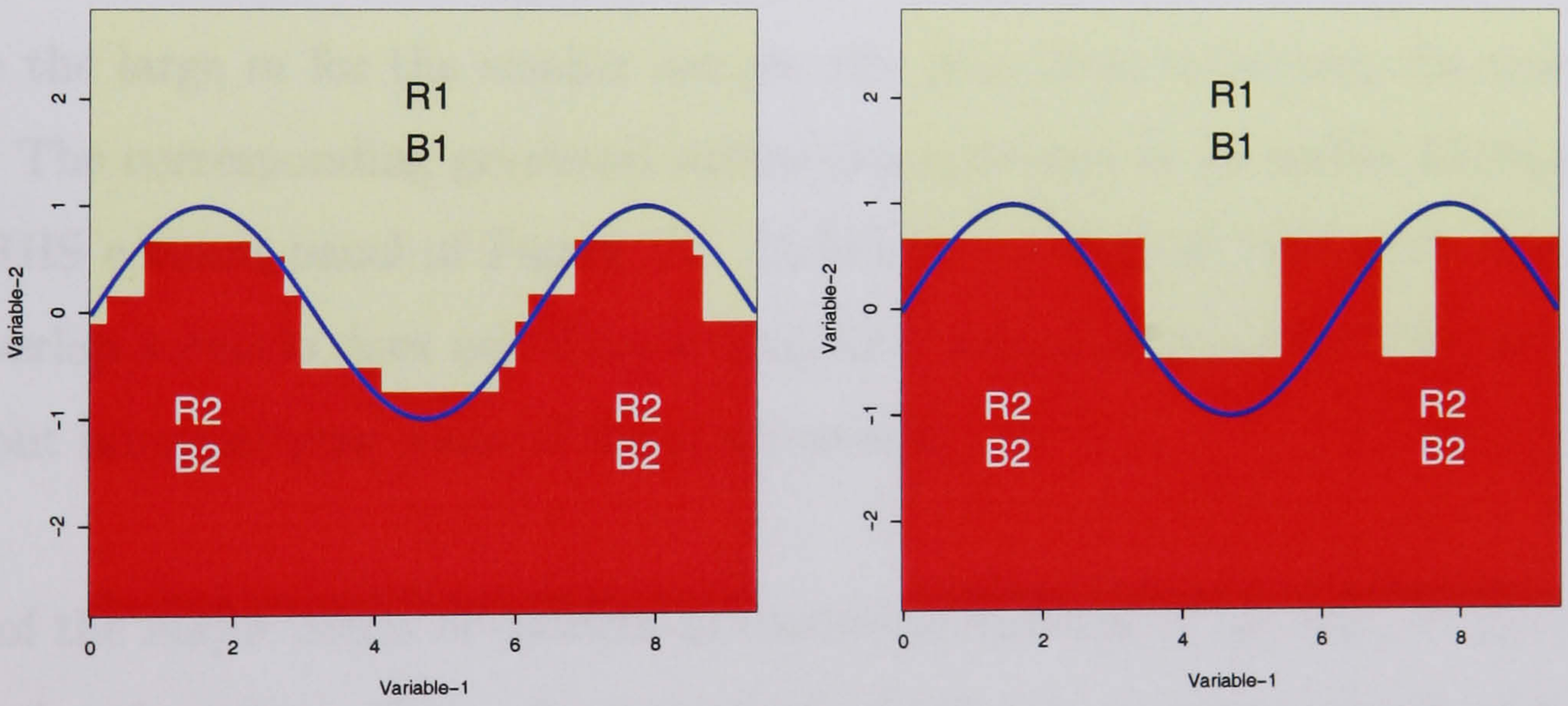


Figure 6.12: *The two panels are based on $\mu = \pm 1$ and $\sigma = 0.608$. They both represent the final partitions after 100 boosting iterations. The LHS panel corresponds to the dataset of size 300 with an estimated population error of 6.1%, while the estimated population error for the RHS panel, corresponding to the dataset of size 60, is 10.68%. Estimation of the theoretical error by pixelisation entails calculating the shaded as well as the unshaded sub-regions above and below the sine-wave respectively. That is, pixelising Variable-1 $\times$ Variable-2 for the centre of each pixel on a $c \times c$ grid and extrapolating to the whole pixel. The misclassification error is then obtained by averaging over the shaded and unshaded pixels.*

under $\mu = \pm 1$. The RHS of equations (6.8) through (6.10) now become $\Phi(-1/2\sigma)$. i.e., to maintain the same levels of overlap, we need half the value of σ used under the $\mu = \pm 1$ scenario, which also changes the class membership density in (6.13).

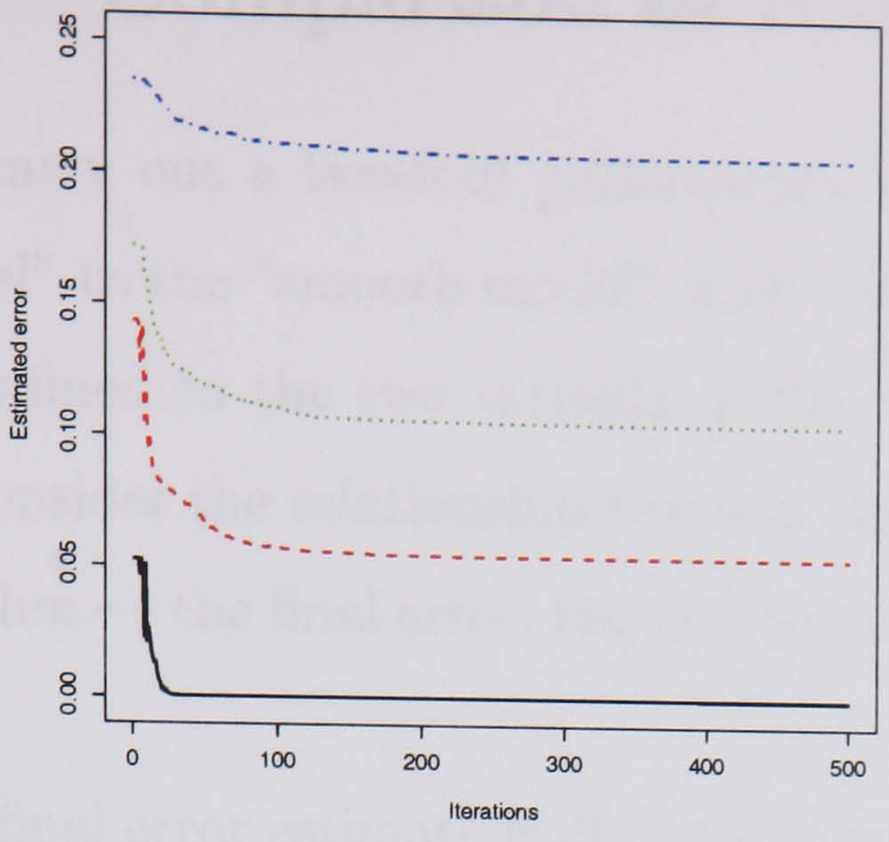
$(2\sigma) \Rightarrow \xi_{B,\text{pop}}$	**Sample size 300**			**Sample size 60**		
	$\xi_{D,\text{pop}}^{(m=1)}$	$\xi_{D,\text{pop}}^{(M)}$	Iterations	$\xi_{D,\text{pop}}^{(m=1)}$	$\xi_{D,\text{pop}}^{(M)}$	Iterations
$(0) \Rightarrow 0\%$	18.47%	0.63%	5	18.34%	8.45%	18
$(0.608) \Rightarrow 5\%$	29.88%	22.69%	60	32.81%	24.78%	25
$(0.781) \Rightarrow 10\%$	32.52%	27.6%	50	35.98%	34.09%	32
$(1.19) \Rightarrow 20\%$	40.97%	36.11%	30	46.22%	42.75%	8

Table 6.8: *The table provides a summary of the population estimation based on pixelisation with boosting, for $\mu = \pm 1/2$. The near-perfect partition observed in Table 6.7 is attained for the larger sample only with an increased number of iterations, while the large m for the smaller sample size does little to recover the true partition.*

Table 6.8 provides numerical details of the performance of boosting under the tightened means scenario. It exhibits a summary of the population estimation based on pixelisation with boosting, for $\mu = \pm 1/2$. The near-perfect partition observed in Table 6.7 is attained for the larger sample only with an increased number of iterations, while the large m for the smaller sample size does little to recover the true partition. The corresponding graphical summaries appeared in an earlier illustration in the RHS column panel of Figure 6.1. Unlike in the case of "widened" means, the no-overlap scenario does not allow a horizontal demarcation of the two two classes without misclassifying some of the observations.

One of the major issues of concern in statistical learning is the sample size used to train the algorithm. Given the same level of overlap, a large n would yield better results than a small n. The following simple example uses $n = 5000$. The main objective of the example is to determine whether or not the foregoing behaviour of the algorithm was characteristic of "small" samples.

Figure 6.13 provides a graphical summary of the performance of the algorithm with $n = 5000$. The LHS and RHS panels correspond to the $\mu = \pm 1$ and $\mu = \pm 1/2$ scenarios respectively. The two panels are based are based on the same $\xi_{B,\text{pop}}$ rates as given in Table 6.7 and Table 6.8 respectively. An internal comparison reveals that



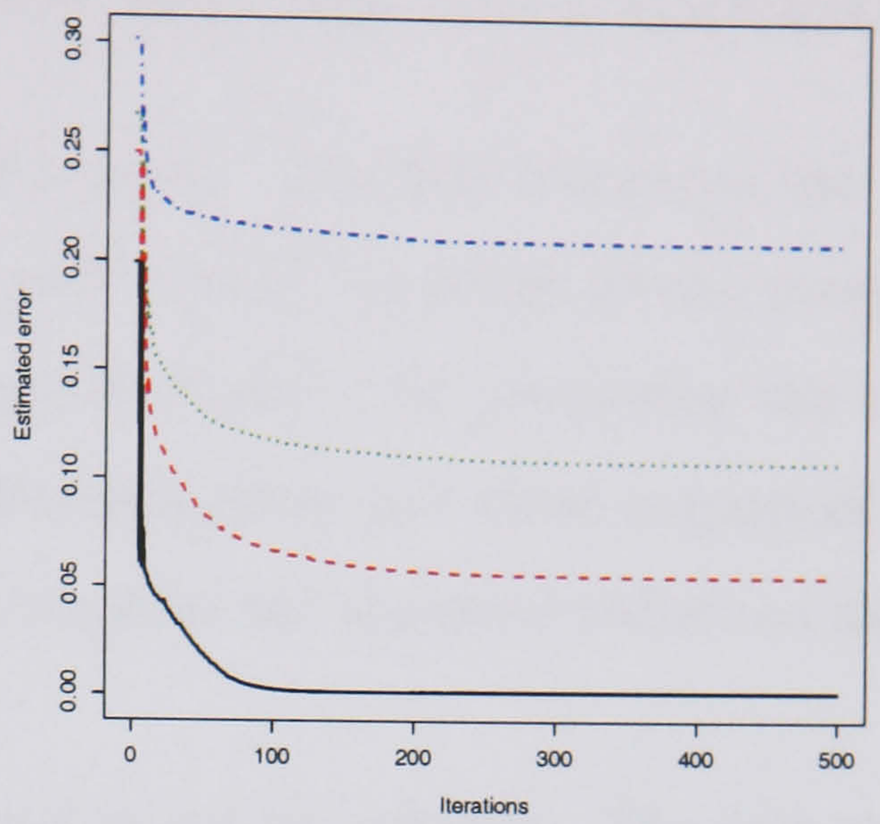


Figure 6.13: *The LHS and RHS panels exhibit boosting plots for the cases* $\mu = \pm 1$ *and* $\mu = \pm 1/2$ *respectively, based on the sample size* $n = 5000$. *The boosting lines, bottom up, correspond to the Bayesian errors 0%, 5%, 10% and 20% respectively. Large* n *gets boosting very close to recovering the Bayesian error in each case.*

the case of $\mu = \pm 1$ out-performs that of $\mu = \pm 1/2$ in each case. However, judging by the magnitude of $E[\Delta^*]$ in each case, it can be inferred that the latter case is more tolerant to noise than the former. Table 6.9 provides a numerical summary of Figure 6.13 as well as $E[\Delta^*]$ for each $\xi_{B,\text{pop}}$.

	Case of $\mu = \pm 1$				**Case of** $\mu = \pm 1/2$			
$\xi_{B,\text{pop}} \rightarrow$	0%	5%	10%	20%	0%	5%	10%	20%
Init.	5.23%	14.32%	17.18%	23.5%	19.93%	24.96%	26.73%	30.16%
Fin.	0%	5.44%	10.56%	20.73%	0.12%	5.50%	10.81%	20.97%
m	26	282	315	324	107	251	429	460
$E[\Delta]^*$	0.00%	4.72%	7.8%	20.86%	0.602%	2.50%	4.84%	9.54%

Table 6.9: *Summary of performance of the algorithm for both the "widened" and "tightened" mean scenarios, with* $n = 5000$. *Both scenarios of the model appear to be getting close enough to the Bayesian error, with the case of* $\mu = \pm 1$ *performing superiorly. Its superiority is also manifested in the number of iterations needed in each case. Other than in the increased number of iterations, the two cases here out-perform the corresponding scenarios in Table 6.7 and Table 6.8 by far.*

Comparing Table 6.9 to Tables 6.7 and 6.8 shows a clear superiority of boosting with a large n, albeit being associated with large m. With an ever increasing sample size, the algorithm gets increasingly closer to capturing the Bayesian error. Clearly, the error margin is much higher in the two tables with a smaller sample size. Increasing the sample size to $n = 5000$, with $\xi_{B,\text{pop}} = 5\%$, for instance, improves the final error from 22.69% down to 5.5%, a really desirable performance.

6.4 Comparison of results for the two models

We carry out a two-fold performance comparison. The first compares the "rough model" to the "smooth model" with "widened" means, while the second comparison is confined to the two variants of the "smooth model". In comparing the models, we consider the relationship between the Bayesian error and three outputs of the algorithm — the final error, the number of iterations and the error-reduction measure.

The final error estimate is obviously the most plausible criterion. The rate at which the algorithm drives the error down constitutes a general criterion for performance comparability, while the resulting error reduction measure provides information as to how fast a model becomes "hopeless" for boosting as $\xi_{B,\text{pop}} \to 50\%$.

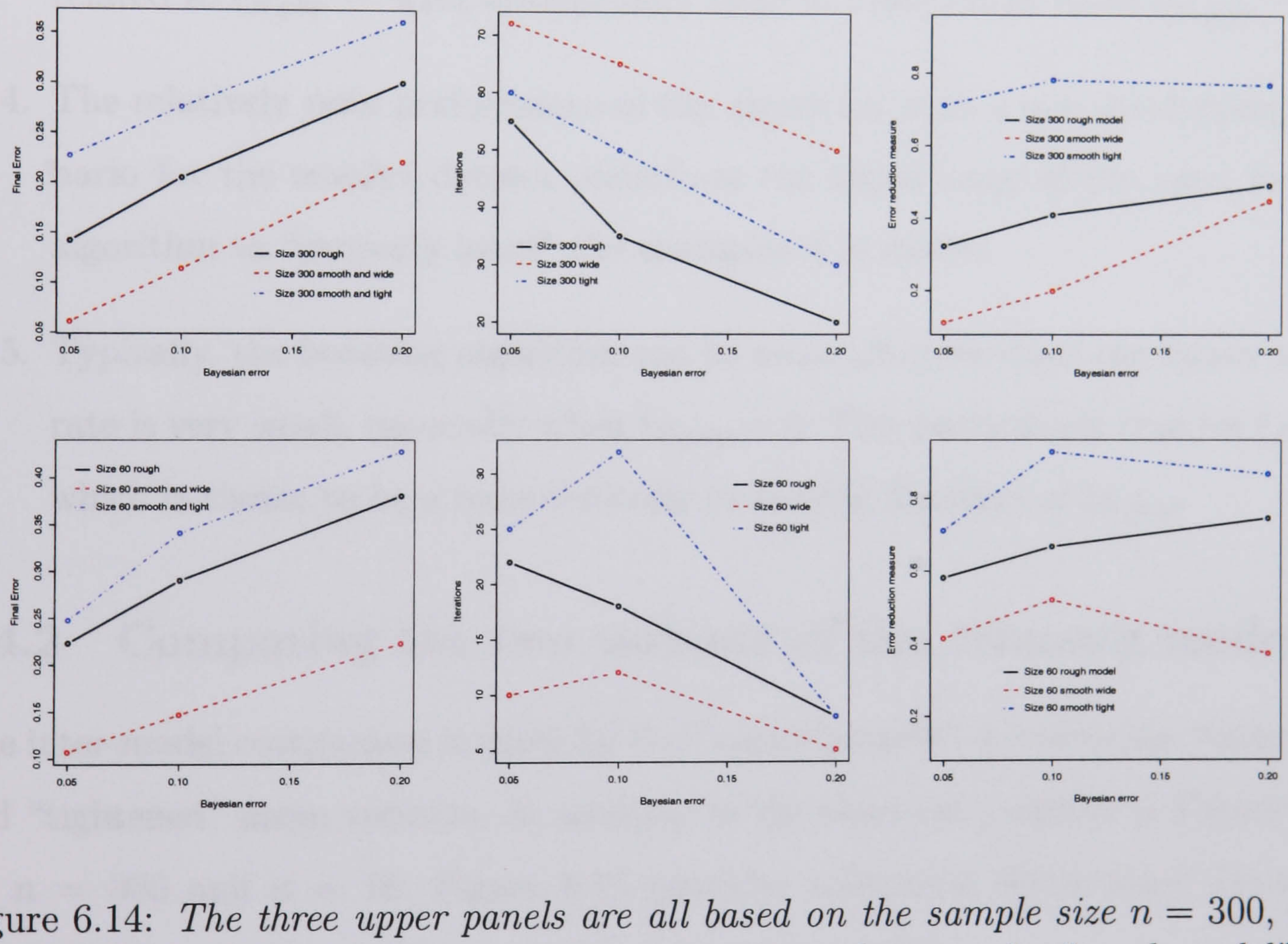


Figure 6.14: *The three upper panels are all based on the sample size* $n = 300$, *and, left to right, they exhibit the performance comparison between the "rough model" and the two variants of the "smooth model", based on the criteria final error estimate, number of iterations and the error-reduction measure respectively. The lower panels provide the same information for the sample size* $n = 60$.

6.4.1 Comparing the "rough" and "smooth" models

Figure 6.14 provides graphical comparison for the "rough" and "smooth" models based on the three criteria listed above. The three upper panels are all based on

the sample size $n = 300$ and, left to right, they correspond to the criteria $\xi^M_{D,\text{pop}}$. m and $E[\Delta]^*$ respectively, while the lower panels, in the same order, correspond to the same criteria for the sample size $n = 60$. The following are some of the conclusions we make from the comparison of the two models.

1. By the crucial criterion $\xi^M_{D,\text{pop}}$, the "smooth model" with "widened" mean yields the best performance, followed by the "rough model" — given large n.

2. Based on $E[\Delta]^*$, boosting under the no-overlap scenario appears to be more reasonable under the "rough model", with not much for it to do under the "smooth model". Large values of $\xi_{B,\text{pop}}$ restore the order of performance in 1.

3. The number of iterations under the "rough model" appears to be inversely related to $\xi_{B,\text{pop}}$ — with dramatically large m observed at lower $\xi_{B,\text{pop}}$.

4. The relatively poor performance of the algorithm with a non-overlapping scenario for the smaller dataset underlines the importance of the need for the algorithm to "properly learn" the examples it is shown.

5. Typically, the boosting algorithm can be more effective when the Bayes' error rate is very small, especially when $\xi_{B,\text{pop}} = 0$. This particularly true for $\xi_{D,\text{pop}}$, which is shown to be a monotonically increasing function of $\xi_{B,\text{pop}}$.

6.4.2 Comparing the two variants of the "smooth model"

The inter-model comparison is made for the "smooth model" between the "widened" and "tightened" mean variants. In addition to the observed patterns in Figure 6.14 for $n = 300$ and $n = 60$, Figure 6.15 provides additional information based on $n = 5000$. Comparison results can be summarised as follows

1. Based on $\xi^M_{D,\text{pop}}$, $n = 300$ and $n = 60$, the model with "widened" mean yields a lower error estimate than its "tightened" mean counterpart. However, as to be expected, the two models approach the same performance as $n \to \infty$.

2. Plots for the error-reduction measure suggest that the model with "tightened" mean is more "error-tolerant" than its "widened" mean counterpart.

3. The complex data structure of the "tightened" mean version of the model entails a much larger number of iterations than the "widened" mean model.

4. The larger the dataset the closer to the Bayesian error the boosting estimate gets and the higher the number of iterations to convergence required.

5. The problem of over-fitting associated with the smaller dataset does appear to be less evident in the case of the "tightened" means.

6. Results from the sample size $n = 5000$ emphasise the importance for "proper training". Increasing n yields an enormous error reduction and does away with the difference in the final error estimate between the two models — implying that the performance of boosting depends on both $\xi_{B,\text{pop}}$ and n.

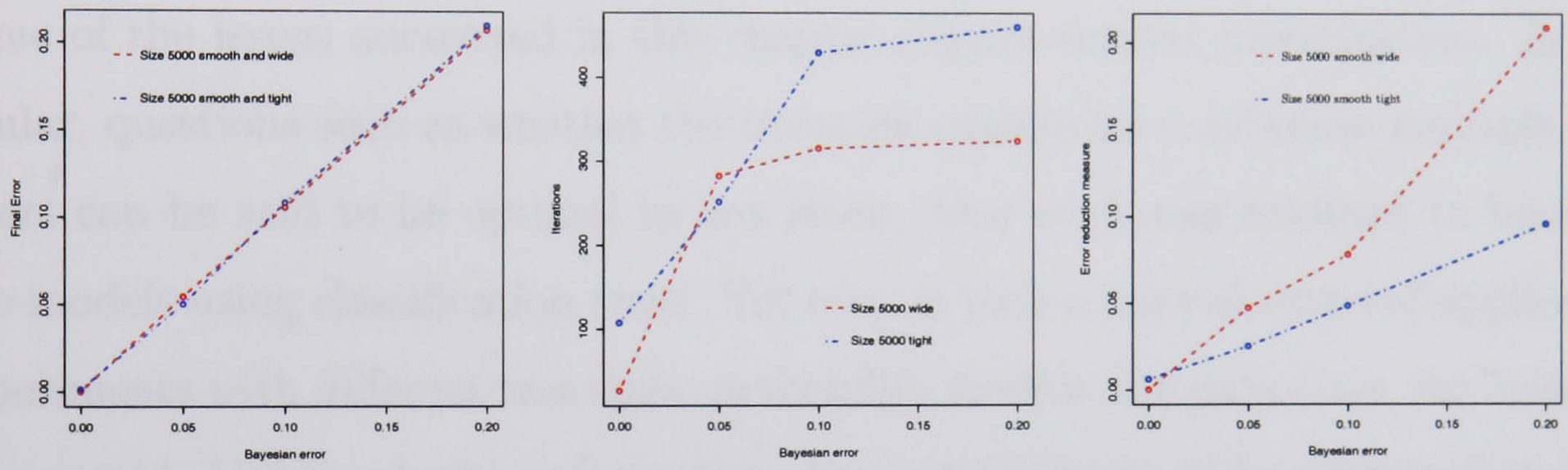


Figure 6.15: *Comparison between the two variants of the "smooth model", based on the sample size $n = 5000$. From left to right, the panels correspond to the criteria $\xi^M_{D,pop}$, m and $E[\Delta]^*$ respectively. The near-identical performance on the LHS is attained at the expense of an enormous difference in m between the two models.*

Note the dramatic difference between the middle and RHS panels of Figures 6.14 and 6.15, with the number of iterations in the former fast decreasing with an increasing $\xi_{B,\text{pop}}$, contrary to the pattern in the latter. As to be expected, the algorithm makes more efforts in learning the rule with a large than with a small sample size.

6.5 Concluding remarks and future directions

The chapter considered various factors that affect the performance of boosting. Based on two probability models and two classes separated by a sine-wave boundary, we ran multiple simulations and showed how boosting is generally hampered by increasing $\xi_{B,\text{pop}}$ and how the location of overlap affects the algorithm. We also

revealed that the performance of boosting depends on the probability model.

The question arises as to why boosting, under the smooth model with widened mean, out-performs boosting under the uniform model. The training process amounts to showing examples to a learning algorithm, thus future success of the algorithm depends much on the "distinctiveness" of the cases it "sees" during training. Consider the non-overlapping scenario under the two models. Although the uniform model is also well-separated, observations closest to the boundary constitute more ambiguity than similar observations under the smoother model. We can therefore conclude that the learning process is less complicated under the smoother model.

6.5.1 Potential future directions

Some of the issues uncovered in this chapter require further investigation. In particular, questions such as whether the boosting scheme for combining multiple classifiers can be said to be optimal in any sense. Our work was confined to boosting two models using classification trees. Yet even in such a limited scope of application experiments with different tree sizes, probability models and data sizes, for instance, may provide highly valuable information. Some of the issues to be addressed include:

1. We need good empirical comparisons between a variety of methods, something which hasn't been widely done. The world of classification uses a wide range of both classical and newly developed discriminant methods — Linear Discriminant Analysis, Classification Trees, Neural Networks, Support Vector Machines and Genetic Algorithms. While enhancement of these methods is commonplace, good empirical comparisons of the methods are still scarce.

2. We need a good collection of datasets for testing the learning schemes, a well-designed environment in which valid tests can be conducted.

3. We need to archive the test results from different sources and methods in order to provide a robust and sensible base for comparison and assessment.

4. We need comprehensive ways of characterising datasets — as exemplified here by the two models and the variants.

Associated issues include data-based choices for M and how to determine model complexity. Indeed, the above-listed issues unanimously seek to address the general problem of defining model complexity. With so many techniques in use on virtually infinite data types some kind of "consensus" would be useful. Attaining that goal, however, remains a subject of many research activities.

Finally, there is still scope for further enhancing the boosting technique. Studies as to why boosting "resists" over-fitting have been widely carried out. Only recently Friedman *et al.* (2000) have shown that boosting, effectively, fits an additive logistic model. Swaying boosting into the Bayesian context, for instance, would make it possible to work provide the model with prior information and probably gain some control over the case re-weighting process, hence the estimated error.

Bibliography

Atkinson, A. C. (1985). *Plots, Transformations, and Regression : an Introduction to Graphical Methods of Diagnostic Regression Analysis.* John Wiley and Sons.

Atkinson, A. C. (1994). Very fast robust methods for the detection of multiple outliers. *Journal of American Statistical Association.* **89**, 1329–1339.

Atkinson, A. C. and Riani, M. (2000). *Robust Diagnostic Regression Analysis.* Springer Verlag.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

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

Cortes, C. and Vapnik, V. (1995). Support vector network. *Machine Learning* **20**, 1–25.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society.* **39**, 1–38.

Dobson, A. J. (1989). *Introduction to Generalized Linear Models.* Chapman and Hall.

Fahmeir, L. and Tutz, G. (1994). *Multivariate Statistical Modelling Based on Generalized Linear Models.* Springer Verlag.

Figueiredo, M. A. T. and Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transcations on Pattern Analysis and Machine Intelligence. 24*(3), 381–396.

Freund, F. and Schapire, R. (1997). A decision-theoretic generalisation of on-line learning and an application to boosting. *Journal of Computer and System Sciences.* **55**, 119–139.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation.* **121**, 256–285.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Machine Learning, Proceedings of the Thirteenth International Conference.*, 13, pp. 148–156.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics.* **28**, 337–407.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

Hadi, A. S. and Simonoff, J. S. (1993). Procedures for the identification of multiple outliers in linear models. *Journal of the American Statistical Association.* **88**, 1264–1272.

Hartigan, J. and Wong, M. (1979). A k-means clustering algorithm. *Applied Statistics.* **28**, 100–108.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning - Data Mining, Inference, and Prediction.* Springer Series in Statistics.

Hazelton, M. L. (2003). Variable kernel density estimation. *Australia & New Zealand Journal of Statistics* **3**, 271.

Kent, J. T., Tyler, D. E., and Yehuda, V. (1994). A curious likelihood identity for the multi-variate t-distribution. *Communication Statistics-Simulation 2*(23), 441–453.

Kiefer, J. and Wolfowitz, J. (1956). Consistency of the maximum likelihood estimates in the presence of infinetely many incidental parameters. *Annals of Mathematical Statistics.* **27**, 887–906.

Lange, K., Little, R., and Taylor, J. (1989). Robust statistical modelling using the t distribution. *Journal of the American Statistical Association.* **84**, 881–886.

Looney, C. G. (1997). *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists.* Oxford University Press.

Mardia, K., Kent, J. T., and Bibby, J. M. (1979). *Multivariate Analysis.* Wiley and Sons.

McLachlan, G. J. and Krishnan, T. (1996). *EM Algorithm and Extensions.* John Wiley and Sons.

McLachlan, G. J. and Peel, D. (2000). *Finite Mixture Models.* Wiley and Sons.

McQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability.*, pp. 281–297.

Mwitondi, K. S., Taylor, C. C., and Kent, J. T. (2002). Using boosting in classification. In *Proceedings of the Leeds Annual Statistical Research Conference*, pp. 125–128. Leeds University Press.

Nakhaeizadeh, G. and Taylor, C. C. (1997). *Machine Learning and Statistics, The Interface.* John Wiley and Sons.

Sakamoto, V., Ishiguro, M., and Kitagawa, G. (1986). *Akaike Information Criterion Statistics.* KTK Scientific.

Schapire, R. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated. *Machine Learning.* **37**, 297–336.

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

Silverman, B. W. (1981). Using kernel density estimation to investigate multimodality. *Journal of the Royal Statistical Society.* **B 43**, 97–99.

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

Taylor, C. C. and Mwitondi, K. S. (2001). Robust methods and data mining - in spatial statistics? In *Proceedings of the Leeds Annual Statistical Research Conference*, pp. 67–70. Leeds University Press.

Titterington, D., Smith, A., and Makov, U. (1985). *Statistical Analysis of Finite Mixture Distributions.* John Wiley and Sons.

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

Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* Springer-Verlag.