Weighted Classification Tree-based Ensemble Methods

Amirah Saeed Alharthi

School of Mathematics / Department of Statistics

University of Leeds

A thesis submitted for the degree of

Doctor of Philosophy

November 2020

The candidate confirms that the work submitted is her own and that appropriate credit has been given where reference has been made to the work of others.

This thesis is dedicated to my parents who always believe in me.

أهدي هذا العمل لوالدي سعيد الحارثي ووالدتي عائشة غالب ... شكراً لكما ...

Acknowledgements

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible to do without Allah who gives me strength to take this journey, and the support and guidance I received from many people.

I would like to say a very big thank you to my supervisor Prof. Charles C Taylor. Charles, thank you very much for guiding me through my PhD, listening and advising me. You have been a great source of support, guidance and encouragement. Another person deserves my sincere thanks is my co-supervisor Dr. Jochen Voss. Jochen, I always have been able to rely on your explanations and help. My gratitude goes to Prof. John T Kent for offering some ideas in some of the theory parts of weighted Gini.

I gratefully acknowledge the funding received during my PhD from the Saudi government for sponsoring my living costs, training courses, conference attendance and tuition fees.

I would also like to say a heartfelt thank you to my mum, dad and my siblings for always believing and encouraging me to follow my goals, and without whom, I would not have had the courage to embark on this journey in the first place.

I greatly appreciate the support received from friends, school staff and colleagues from inside and outside The School of Mathematics.

This thesis was supported in some part through computational resources (Advanced Research Computing-ARC3) provided by The University of Leeds.

Abstract

This thesis explores the role and selection of weighted data in classification trees, including ensemble applications that is tackling datasets with weights. In order to achieve this goal, we will combine decision trees as base classifiers with some ensemble methods. Decision trees are quite an easy to interpret method but their performance is not usually the most accurate performance among all other machine learning algorithms. However, a way to overcome this problem is to combine many decision trees together, i.e. an ensemble of trees. Ensemble methods are popular in statistical computing for their powerful performance.

In order to focus our methods, we started by investigating the effect on the tree first splitting variable and split point when the re-sampling method changes from bootstrapping to reweighting the observations according to a distribution. We found that this change does not have a major impact on the distribution of the first split variable and split point in most cases we investigated. Then, we introduced two methods: "bagging with stumps method" and "Gini stumps method". Bagging with stumps method is decision stumps fitted on different sample sizes while Gini stumps method is a new way to generate split points with probabilities proportional to Gini gain. There are two sub-methods of Gini stumps method, Gini-sampled stumps method and Gini-midpoints stumps method. After that, these methods with two aggregation methods (majority vote and weighted vote) are applied to simulations and real-world datasets to measure their performance accuracies. Gini stumps method has faster computational time than bagging with stumps when using an ordinary tree as the base

classifier. Also, Gini stumps method with especially weighted vote has promising results with most simulations and real-world datasets.

Gini stumps method was adjusted to tackle weight updates and then combined with AdaBoost methods. This combination has accurate results especially in discrete AdaBoost.

We found the expected value of Gini index theoretically in case of two classes, and the final formula is the squared difference between the two cumulative distributions divided by the variance of the weighted sum of the two cumulative functions. This final result is the same as the distance that the Anderson-Darling test is based on.

List of Notation

Ω	measurement space
x_{ir}	represents the i^{th} observation of the r^{th} variable
y_i	label of the i^{th} observation
C	a set has $j \in \{1, 2, \ldots, J\}$ classes
π	prior probability
$d(x)$	a classifier
B	number of bootstraps
ϕ	impurity function
$\mathcal{P}, \mathcal{L}, \mathcal{R}$	parent, left and right nodes of a tree
w_i	the weight of observation x_i
$\Gamma(\cdot)$	Gini index function
κ	power used with Gini function

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The research focus is developing classification tree-based ensemble methods in which we consider various aspects of weighted observations. Changing observation weights increases the importance of these observations as well as makes the tree more biased towards them. Adjusting observation weights can overcome the problem when a sample is not representative of the population and due to that, this change can lead to an improvement in the method accuracy. Another useful use of the weight adjusting is to reduce the effect of outliers (1). In this chapter, we introduce decision trees as a classification algorithm and the type of data they are used with. Decision trees are simple classifiers, and they are usually combined in ensembles to enhance their performance. We will introduce the ensemble methods briefly especially bagging and boosting as ways of combining many classifiers. We will discuss the motivation of this thesis and the importance of applying weight to observations that could improve the accuracy of the model. Finally, we have a detailed outline of the thesis.

1.1 Introduction to Decision Trees

Decision trees are a decision tool which use a tree-like graph of decisions represented by classification trees and regression trees. Decision trees are popular methods in statistical learning and analysing data as well as in expert systems.

Data comes in from many sources, some forms of this data do not have structured attributes, for example in city planning: identifying groups of houses according to their house type, value, size and geographical location, we know there are similarities between them, but they are not identical to each other and we cannot easily assign a class. This kind of data is called unlabelled data, and the algorithms that deals with unlabelled data are known as unsupervised learning (Page 1, (2)). The other form of data has specific labels, and we can recognise every labelled observation. As an example, a credit card company typically receives a large number of applications from new potential customers, and these applications contain information regarding several different features, such as annual salary, age, outstanding loans etc. The data class is collected as an outcome of the credit card being approved and each observation of this data can be categorised into one of several classes; good credit or bad credit or requiring further analysis. Data of this form is called labelled or categorical data and the type of algorithm to deal with this is known as supervised learning.

Decision trees seek to find a relationship between measurement variables and classes in a group of observations that form the dataset. They can be used with categorical (qualitative) or numeric (quantitative) data and they are called classification and regression trees (CART), where the classification tree maps the input space into predefined classes and the regression tree maps the input space into real values. The term CART is an umbrella term, as well as a specific algorithm, used to indicate both of the above procedures, and was first introduced by Breiman et al.(3). We will focus on classification tree-based methods in this thesis.

1.2 Introduction to Ensemble Methods

Over the last couple of decades, ensemble methods have received well deserved attention within statistical learning models. Ensemble methods reduce the variability and improve the accuracy of decision making models (4). There are some popular ensemble methods such as: bagging and boosting.

Bagging stands for bootstrap aggregating and it was proposed by Breiman (5). Bagging is one of the simplest and yet effective ensemble methods and it has some versions as: bagging predictors that is generating multiple versions of a predictor and aggregating them (5), and random forest algorithm is another creative version of bagging and it was introduced by Breiman (6). Random forests create more diversity in the multiple predictors which improves the prediction and decrease the variance of the model. We will talk in more detail about bagging in Chapter 2 and the random forest algorithm in Chapter 3.

Boosting was first introduced in (7) as an iterative way for generating a strong classifier from a classifier that can barely be better than random guessing. Boosting focuses each iteration on misclassified observations by the previous classifier. We will explain boosting in Chapter 7.

1.3 Motivation

Classification is increasingly an important area in statistics and computer science research. Tree-based ensemble methods are popular type of classification algorithms. The base of these methods is decision trees which are easy to understand and interpret but they perform much better within ensembles. One of the very powerful tree-based ensemble methods is random forests by Breiman (6). So far, however, random forests method has not been explored to change observation weights or give extra weight to misclassified observations. For that, the ultimate aim of this research is to find a tree-based ensemble method where observation weights can be changed and updated.

One of the key ingredients in ensemble learning is to obtain samples which are less correlated, so there will be more variance in the models which leads to decreasing the misclassification rate. This led to an investigation into the use of weighted data as a means to reduce the correlation between samples. This investigation started from simple cases, to try to further understand the role of weights, finally leading to an attempt to utilise some of the results in bagging and boosting of stumps.

1.4 Thesis Outline

Throughout this thesis we work towards having a method to deal with data with weight. We begin Chapter 2 by introducing some classification concepts like: classification applications, types, error and algorithms. Also, we highlight a common problem that classification has which is overfitting, show cross validation as a way of evaluate performance and finally discuss bagging.

Classification decision trees are explored in Chapter 3. We discuss constructing trees and splitting criterion (impurity-based criteria); the default splitting criteria is Gini gain in CART (3). Then, we find the formula for the expected value of Gini gain theoretically. We also review tree pruning to overcome the problem of overfitting and show that decision trees are invariant to monotonic transformations of the data. Some decision aggregation methods are explained in this chapter including the random forest method.

We carried out an investigation of the distribution of the first split when the observation weights are changed in Chapter 4, then show the results of the investigation on one/multiple-explanatory-variable simulations and real datasets.

In Chapter 5, we introduce two schemes of forests of stumps, bagging with stumps and Gini stumps method. While bagging with stumps method focuses on fitting stumps to different size of samples that are drawn from the data, Gini stumps method is a new way of generating the split of variables and points. We here apply these two methods on one/multiple-explanatory-variable simulations to estimate their performance. We also compare between bagging with stumps and Gini stumps method in terms of their consumed time to generate a set of split points and the distribution of the split points from both methods with some simulations.

To confirm our results, in Chapter 6 we apply these methods on real datasets. One of the real datasets has a different performance than what we expect and that data is investigated further which leads us to investigate different assumptions about this performance.

We explore boosting classification stumps in Chapter 7. This method combines AdaBoost with Gini stumps methods. Also, we try to find the expected value of the weighted Gini gain and reach a very complicated integrals. To conclude, Chapter 8 summarises our research, contributions and future work.

Chapter 2

Classification

Classification is one of the most studied topics in many branches of science and that is because of its fundamental rule in solving problems in statistics, biology, medicine, artificial intelligence, engineering etc. In this chapter, we will introduce some classification applications which appear in some of the programs that most people use daily. We also will talk briefly about data types and the suitable classification type for the data as well as algorithms. Classification error and Bayes rule are discussed here in detail alongside giving examples of the maximal accuracy for different models. We highlight one of the most common problems in classification that is overfitting, and the reason behind it. Overfitting causes a poor accuracy in the testing data and the most widely used method for estimating and evaluation prediction error is cross validation. Finally, bagging is introduced as a classification ensemble method and we explained how it works.

2.1 Applications

Collaborative filtering is one of the classification applications, and video rental sites such as "Netflix" is an example of collaborative filtering. Netflix uses customer preferences to attract users by suggesting films and TV shows that could be of interest. To explain that, if there is no search history, then the website suggests random contents but if there is a previous history of searching or watching, then

it uses the content attributes, like comedy, drama, horror, science fiction,...etc, to recommend new contents that match these attributes. Figure 2.1 shows the recommended contents based on my previous history.

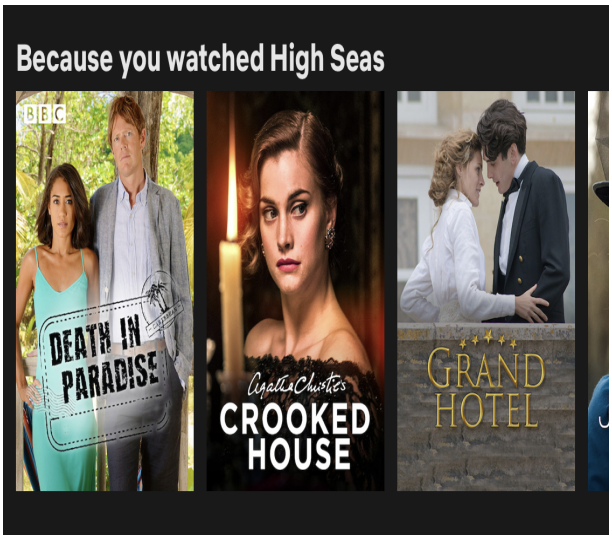


Figure 2.1: Videos advertised to me based on my previous watching on Netflix.

There are many other applications that use classification such as: named entity recognition, e.g. recognising spam emails, speech recognition, e.g. Amazon Alexa and car Bluetooth, text prediction, e.g. Gmail Smart Compose and automatic translation, e.g. Google Translate.

From all these applications, we can see that classification, in statistics, means determining to which categories a new observation belongs to. To be able to do that, we want to construct a system or "classifier", where this system requires data containing observations with known categories. We will refer to this data as training data afterwards, that is because this data is used to construct and train the classifier. This classifier needs to have the ability to learn how to classify new observations, i.e. "observations with unknown categories", and that has the same probability distribution as the training data, and this data will be referred to as testing data (Page 3, (8)).

That leads us to talk about the different types of data we will deal with.

2.2 Data

Usually, similar classifiers solve problems on similar dataset types. So, it is useful to know the different kinds of datasets and characterise learning systems according to them. Classification deals with data that has a categorical response variable and this response variable is one of the following:

Binary variable: that has two values, e.g. true and false or $-1, 1$ or $1, 2$

Numeric variable: such as integers, e.g. a finite set of natural numbers $\mathbb{N}$

Categorical variable: this type of variables take one value of limited possible set of values, e.g. blood types and nationalities

To explain the structure of the data, let n be the number of observations and p be the number of feature variables that are available for making predictions. Let x_{ir} represent the value of the i^{th} observation of the r^{th} variable, where $i = 1, 2, \ldots, n$ and $r = 1, 2, \ldots, p$. This can be represented by an $n \times p$ matrix X, whose $(i, r)^{th}$ element is x_{ir}.

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{bmatrix}. \tag{2.1}$$

Every row of X represents one observation and we write the observations as $x_1, x_2, \ldots, x_n$ and we have the measurement space Ω which contains all possible measurements. Here, x_i is the i^{th} observation and it is a vector of length p containing the p feature variables for the i^{th} observation $x_i^T = (x_{i1}, \ldots, x_{ip})$.

Every column of X represents one variable and we write the variables as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$, and each is a vector of length n. By using this we can also write

this matrix as:

$$X = \left[\mathbf{x}_1 \mathbf{x}_2 \dots \mathbf{x}_p\right]. \tag{2.2}$$

We use y_i to denote the label of the i^{th} observation where $y_i \in C$, such that C is the set of classes $C = \{1, 2, \dots, J\}$. Take $(\mathbb{X}, Y), \in \Omega \times C$, to be random variables from the probability distribution of the training data and these classes have priors π_j, which can be defined as follows:

Definition 1. *The prior probabilities $\pi_j, j = 1, \dots, J$ can be defined as*

$$\pi_j = Pr(Y = j), \text{ such that} \sum_j \pi_j = 1.$$

Definition 2. *The probability distribution of the j^{th} class measurement vector is given by*

$$Pr(\mathbb{X} = x|Y = j) = \frac{Pr(\mathbb{X} = x, Y = j)}{\pi_j}, \tag{2.3}$$

and so,

$$Pr(\mathbb{X} = x, Y = j) = Pr(\mathbb{X} = x|Y = j)\pi_j,$$

$Pr(\mathbb{X} = x|Y = j)$ is a conditional probability density function for $\mathbb{X}$ which belongs to group j. Since $\mathbb{X}$ is real-valued, we denote the probability density function $Pr(\mathbb{X} = x|Y = j)$ as a probability density function pdf, $f_j(x)$.

From Bayes theorem,

Definition 3. *The posterior probability is defined by*

$$Pr(Y = j|\mathbb{X} = x) = \frac{Pr(\mathbb{X} = x|Y = j)\pi_j}{\sum_{j=1}^{J} Pr(\mathbb{X} = x|Y = j)\pi_j} = \frac{f_j(x)\pi_j}{\sum_{j=1}^{J} f_j(x)\pi_j}. \tag{2.4}$$

There are different types of classification types as follows:

2.3 Classification Types

Binary Classification: is the most often used and studied problem in classification, where $C = \{1, 2\}$, i.e. $|C| = 2$. A large number of algorithms

rely on binary classification. Figure 2.2 shows how to clearly distinguish between two sets of shapes (circles and squares). This technique could re-label observations based on the absence or presence of a label or category. For example, e-mail is: spam or not and the status of participant: healthy or sick.

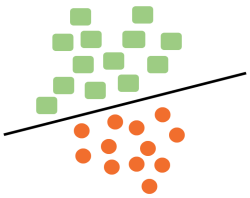


Figure 2.2: Binary classification and this classifier is called a "linear classifier".

Multi-class Classification: this is an extension of binary classification, the difference here is there are more than two categories, for instance, blood types (A, B , AB, O) and top five most spoken languages in the world (Mandarin, English, Hindustani, Spanish, Arabic). Figure 2.3 shows multi-class classification which distinguishes between three classes.

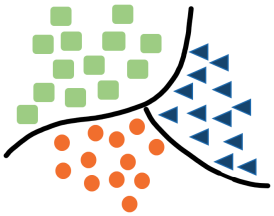


Figure 2.3: Multi-class classification.

To define a classifier:

Definition 4. *A classifier is a function $d(x)$ defined on Ω, where for every $x \in \Omega$: $d(x)$ takes a value in $\{1, 2, \ldots, J\}$. Another way of looking at a classifier is to define Ω_j as the subset of Ω where $d(x) = j$; that is:*

$$\Omega_j = \{x : d(x) = j\}. \tag{2.5}$$

The sets $\Omega_1, \Omega_2, \ldots, \Omega_J$ are disjoint sets and $\Omega = \bigcup_{j=1}^{J} \Omega_j$, so for every $x \in \Omega_j$ the predicted class is j.

$$Pr(\mathbb{X} \in \Omega_j | Y = j) = \int_{\Omega_j} f_j(x)\ \mathrm{d}x, \tag{2.6}$$

where $f_j(x)$ is defined in Equation (2.3).

2.4 Classification Error and Bayes Rule

Definition 5. *The classification error $R(d)$ is defined as the probability that the random variables $(\mathbb{X}, Y) \in \Omega \times C$, is incorrectly classified (9),*

$$R(d) = Pr(d(\mathbb{X}) \neq Y). \tag{2.7}$$

Note that:

- *classification rate = Pr(correct classification)*
- *classification error = 1 - classification rate*

Definition 6. *Bayes rule can be defined as*

$$d^*(x) = \underset{j}{\operatorname{argmax}}\ \pi_j f_j(x), \tag{2.8}$$

Lemma 1. *Denote the Bayes classification error by*

$$R^*(d^*) = Pr(d^*(\mathbb{X}) \neq Y),$$

where $\mathbb{X} \in \Omega$ *is a random variable of any data point from* Ω *and* $Y \in C$ *is a random variable from the classes.Then, for any other classifier* $d(\mathbb{X})$*, Bayes rule* $d^*(\mathbb{X})$ *has the smallest classification error, i.e.* $Pr(d^*(\mathbb{X}) \neq Y) \leqslant Pr(d(\mathbb{X})) \neq Y)$ *(3).*

Proof. To prove it, consider a two-class problem, with $C = \{1, 2\}$, where there are two hypotheses,

$$\begin{aligned} H_1 &: \mathbb{X} \in \text{Class1} \\ H_2 &: \mathbb{X} \in \text{Class2}, \end{aligned} \tag{2.9}$$

and π_1 and $\pi_2 = 1 - \pi_1$ are the prior probabilities of class 1 and 2, respectively. To choose between the two hypotheses in Equation (2.9), accept hypothesis H_j if $x \in \Omega_j$. In testing the hypotheses, there are 2 possible incorrect outcomes,

- $d(x) = 1$ when $y = 2$
- $d(x) = 2$ when $y = 1$,

and these occur with probabilities

- $\int_{\Omega_1} Pr(\mathbb{X} = x|y = 2)\mathrm{d}x$
- $\int_{\Omega_2} Pr(\mathbb{X} = x|y = 1)\mathrm{d}x$

Suppose we have the two errors $\int_{\Omega_2} Pr(\mathbb{X} = x|y = 1)\mathrm{d}x$ and $\int_{\Omega_1} Pr(\mathbb{X} = x|y = 2)\mathrm{d}x$. Our goal now is to find the optimal regions Ω_j so that the average error ζ is minimised. The average error is:

$$\zeta = \pi_1 \int_{\Omega_2} f_1(x)\mathrm{d}x + (1 - \pi_1) \int_{\Omega_1} f_2(x)\mathrm{d}x. \tag{2.10}$$

The average error ζ depends on decision regions Ω_1 and Ω_2. If $L(x) = \frac{f_2(x)}{f_1(x)}$ is the likelihood ratio and $A^* = \frac{\pi_1}{1-\pi_1}$, the average error ζ is minimum when the region Ω_1 contains of values of x for which $L(x) \leqslant A^*$ and $L(x) > A^*$ for the region Ω_2.

For any classifier $d(x)$ with two decision regions Ω_1^* and Ω_2^*, and an average error $\zeta(d)$ and the Bayes classifier $d^*(x)$ with a average error $\zeta(d^*)$, the subtraction of a Bayes classifier average error from any other classifier's average error

according to Equation (2.10) will be as follows,

$$\begin{aligned}
\zeta(d) - \zeta(d^*) &= \pi_1 \int_{\Omega_2^*} f_1(x)\mathrm{d}x + (1-\pi_1)\int_{\Omega_1^*} f_2(x)\mathrm{d}x - \\
&\quad \pi_1 \int_{\Omega_2} f_1(x)\mathrm{d}x - (1-\pi_1)\int_{\Omega_1} f_2(x)\mathrm{d}x \\
&= \pi_1\Big(\int_{\Omega_2^*} f_1(x)\mathrm{d}x - \int_{\Omega_2} f_1(x)\mathrm{d}x\Big) + \\
&\quad (1-\pi_1)\Big(\int_{\Omega_1^*} f_2(x)\mathrm{d}x - \int_{\Omega_1} f_2(x)\mathrm{d}x\Big) \\
&= A^*(1-\pi_1)\Big(\int_{\Omega_2^*} f_1(x)\mathrm{d}x - \int_{\Omega_2} f_1(x)\mathrm{d}x\Big) + \\
&\quad (1-\pi_1)\Big(\int_{\Omega_1^*} f_2(x)\mathrm{d}x - \int_{\Omega_1} f_2(x)\mathrm{d}x\Big)
\end{aligned} \tag{2.11}$$

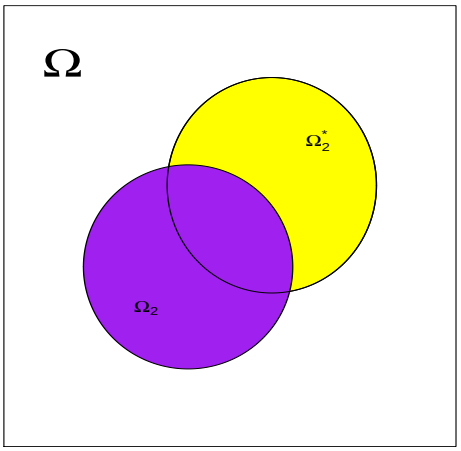


Figure 2.4: The measurement space and decision regions. There are two regions for Bayes classifier $d^*(x)$ that are Ω_1 and Ω_2, such that $\Omega = \Omega_1 \cup \Omega_2$. Also, there are two regions for any classifier $d(x)$ that are Ω_1^* and Ω_2^*, such that $\Omega = \Omega_1^* \cup \Omega_2^*$

From Figure 2.4, $\Omega_2^c = \Omega_1$, $\Omega_1^c = \Omega_2$, $\Omega^{*c}_2 = \Omega_1^*$, and $\Omega^{*c}_1 = \Omega_2^*$.

$$\begin{aligned}
\Omega_2^* &= \Omega_2^* \cap \Omega_2^c = \Omega_2^* \cap \Omega_1, \\
\Omega_1^* &= \Omega_1^* \cap \Omega_1^c = \Omega_1^* \cap \Omega_2, \\
\Omega_2 &= \Omega_2 \cap \Omega^{*c}_2 = \Omega_2 \cap \Omega_1^*, \\
\Omega_1 &= \Omega_1 \cap \Omega^{*c}_1 = \Omega_1 \cap \Omega_2^*.
\end{aligned} \tag{2.12}$$

By substituting Equations (2.12) in Equation (2.11), we have:

$$\begin{aligned}
\zeta(d) - \zeta(d^*) &= A^*(1-\pi_1)\Big(\int_{\Omega_2^* \cap \Omega_1} f_1(x)\mathrm{d}x - \int_{\Omega_2 \cap \Omega_1^*} f_1(x)\mathrm{d}x\Big) + \\
&\quad (1-\pi_1)\Big(\int_{\Omega_1^* \cap \Omega_2} f_2(x)\mathrm{d}x - \int_{\Omega_1 \cap \Omega_2^*} f_2(x)\mathrm{d}x\Big) \\
&= (1-\pi_1)\Big(\int_{\Omega_1 \cap \Omega_2^*} A^* f_1(x)\mathrm{d}x - \int_{\Omega_1^* \cap \Omega_2} A^* f_1(x)\mathrm{d}x + \\
&\quad \int_{\Omega_1^* \cap \Omega_2} f_2(x)\mathrm{d}x - \int_{\Omega_1 \cap \Omega_2^*} f_2(x)\mathrm{d}x\Big)
\end{aligned}$$

$$\begin{aligned}
\zeta(d) - \zeta(d^*) = (1-\pi_1)\Bigg\{ &\int_{\Omega_1 \cap \Omega_2^*} \Big(A^* f_1(x) - f_2(x)\Big)\mathrm{d}x + \\
&\int_{\Omega_1^* \cap \Omega_2} \Big(f_2(x) - A^* f_1(x)\Big)\mathrm{d}x\Bigg\},
\end{aligned} \tag{2.13}$$

when $x \in \Omega_1$, $L(x) \leqslant A^*$,

$$A^* f_1(x) - f_2(x) \geqslant 0,$$

and when $x \in \Omega_2$, $L(x) > A^*$ and $f_2(x) - A^* f_1(x) > 0$. Always, the integrals in Equation (2.13) are greater than or equal to zero,

$$\zeta(d) - \zeta(d^*) > 0,$$

and $\zeta(d^*)$ is the minimum average error (Page 18, (10)) □

The Bayes classification error is

$$\begin{aligned}
R^*(d^*) &= Pr(d^*(\mathbb{X}) \neq Y) \\
&= 1 - Pr(d^*(\mathbb{X}) = Y) \qquad \text{from Lemma 1} \\
&= 1 - \sum_{j=1}^{J} Pr(\mathbb{X} \in \Omega_j, Y = j) \\
&= 1 - \sum_{j=1}^{J} \pi_j Pr(\mathbb{X} \in \Omega_j | Y = j) \\
&= 1 - \sum_{j=1}^{J} \pi_j \int_{\Omega_j} f_j(x)\, \mathrm{d}x \qquad \text{from Equation (2.6)} \\
&= 1 - \left[\int_{\Omega_1} f_1(x)\pi_1\, \mathrm{d}x + \int_{\Omega_2} f_2(x)\pi_2\, \mathrm{d}x + \int_{\Omega_3} f_3(x)\pi_3\, \mathrm{d}x + \ldots \right]
\end{aligned}$$

Since $\Omega_j = \{x : \ f_j(x)\pi_j = \max_{j'} f_{j'}(x)\pi_{j'}\}$

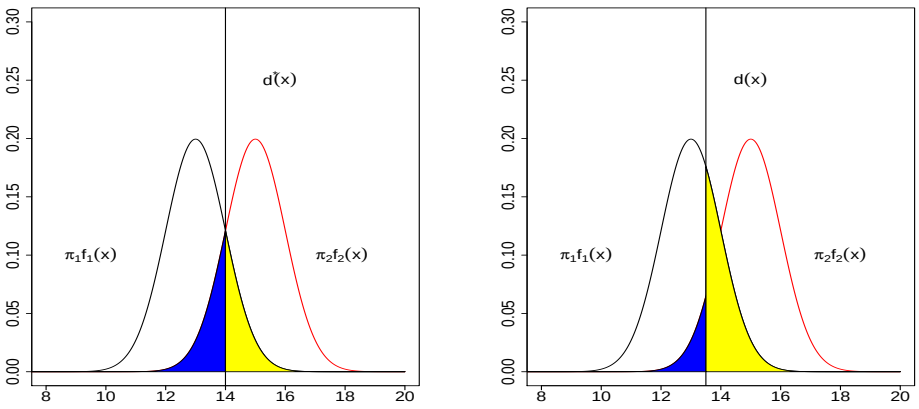
$$R^*(d^*) = 1 - \sum_{j=1}^{J} \int_{\Omega_j} \max_{j'} f_{j'}(x)\pi_{j'} \mathrm{d}x. \tag{2.14}$$

Figures 2.5 and 2.6 show two normal probability density function, with equal and different priors, respectively.

Figure 2.5 shows Bayes classifier $d^*(x) = \begin{cases} 1, & \text{if } x < 13.99 \\ 2, & \text{otherwise} \end{cases}$, while the other classifier $d(x) = \begin{cases} 1, & \text{if } x < 13.5 \\ 2, & \text{otherwise} \end{cases}$. The maximal accuracy for Bayes classifier $d^*(x)$ is 84.13% and 81.23% for the classifier $d(x)$.

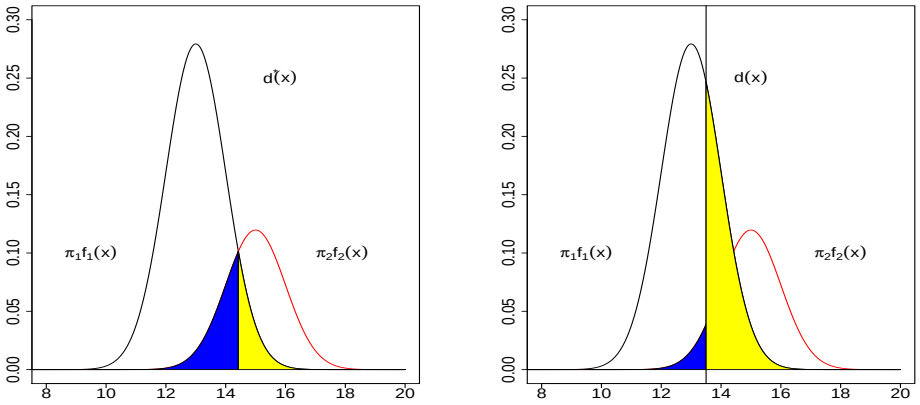
Figure 2.6 shows Bayes classifier $d^*(x) = \begin{cases} 1, & \text{if } x < 14.42 \\ 2, & \text{otherwise} \end{cases}$, while the other classifier $d(x)$ as above. The maximal accuracy for Bayes classifier $d^*(x)$ is 86.12% and 76.40% for the other classifier $d(x)$.

Here, $x = 13.99$ and $x = 14.42$ are the intersection points of the two curves f_1 and f_2 weighted by equal and different priors in Figures 2.5 and 2.6, and $x = 13.5$ is a chosen point.



(a) $d^*(x)$ (b) $d(x)$

Figure 2.5: Two normal probability density functions $f_1 \sim N(13, 1)$ and $f_2 \sim N(15, 1)$ with equal priors. There are two classifiers $d^*(x)$ and $d(x)$ and they split observations into two classes. The blue shaded area indicates the observations from class 2 which are incorrectly classified as class 1 while the yellow shaded area indicates the observations from class 1 which are incorrectly classified as class 2.



(a) $d^*(x)$ (b) $d(x)$

Figure 2.6: Two normal probability density functions $f_1 \sim N(13, 1)$ and $f_2 \sim N(15, 1)$ with two priors $\pi_1 = 0.7$ and $\pi_2 = 0.3$. There are two classifiers $d^*(x)$ and $d(x)$ and they split observations into two classes. The blue shaded area indicates the observations from class 2 which are incorrectly classified as class 1 while the yellow shaded area indicates the observations from class 1which are incorrectly classified as class 2.

2.5 Classification Algorithms

As we have mentioned earlier classification is a supervised learning method. There are many classification algorithms such as:

- Linear classifiers: naive Bayes classifier, logistic regression
  Naive Bayes classifier
  Naive Bayes classifier is a probabilistic classification algorithm based on Bayes' theorem and a set of independence assumptions between features. The naive Bayes classifier makes a (naive) conditional assumption that each feature variable given the class is independent of each of the other feature variables. This algorithm has been studied since 1960s and it is the most popular method for text categorisation (11).

  Logistic regression
  Logistic regression is a classification algorithm which basically uses a logistic function. The logistic regression classifier is used to explain the relationship between the binary response variable and the explanatory feature variables. Moreover, if the data has a categorical response variable with more than two classes, a multinomial logistic regression is used to deal with this data. The multinomial logistic regression is used to predict the probabilities of the non-binary response variable given a set of explanatory variables (12).

- k-nearest neighbours algorithm
  The k-nearest neighbours algorithm (k-NN) is a non parametric method used for classification and regression. k-NN for classification works in a way such that it finds the k closest training observations to an object, which we want to find its class, then it allocates this object to the most popular class of the object's k-nearest neighbours. One of the useful technique is used with k-NN method is to take the weight of observation into consideration, so, the closest observations contribute more to the vote than the further away ones (13).

- Neural networks
  A neural network is based on a group of connected units "neurons". Each neurone receives a signal, processes it then transfer it to other neurons that are connected to it. The "signal" or activation at each neurone is a real number and each neuron's output is computed by a non-linear function of the sum of its inputs. The neural network has input layer, hidden layers and output layers (14). Each layer has a collection of neurons and each of which connects to all neurons in the next layer and so on as in Figure 2.7.

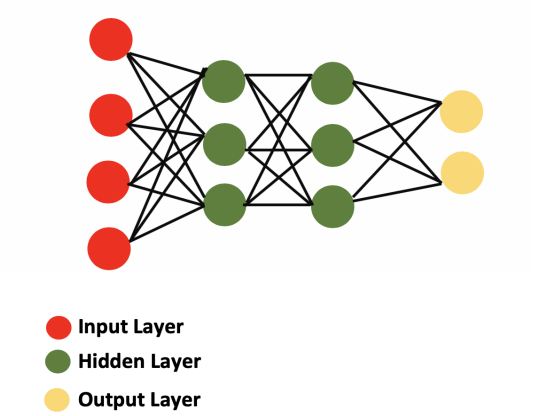


Figure 2.7: A simple neural network that has different layers.

- Support vector machines
  A support vector machine is a supervised learning algorithm and it constructs a set of hyperplane in a space. The support vector machine can be used for classification and regression and a desirable hyperplanes has the largest distance "margin" to the closest data points. A large margin produces a more robust learner. (Page 418, (15)).

- Decision Trees (Decision Stumps, Boosted Stumps and Random Forest)
  We will focus in our work on trees generally and tree-based algorithms are explained later in the thesis, in Chapter 3.

2.6 Overfitting

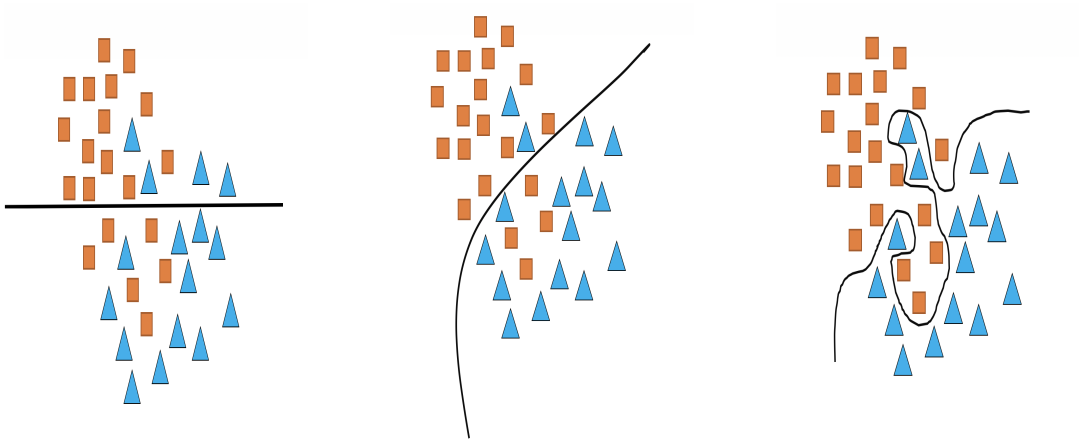
We have said the main aim of classification is to use the classifier to predict the classes for new observations which are drawn from the same distribution as the training dataset. There are two types of errors committed by a classification model: training errors and generalisation errors. The training error is the proportion of classification errors committed on the training set, and generalisation error is the expected error of the model on previously unseen observations or a testing dataset.

A good classification model must both appropriately fit the trained observations and accurately predict the class labels of a testing dataset. Thus, a key purpose of the learning algorithm in general is to build models that correctly predict the class labels. A situation like this is known as overfitting and it occurs when a model is very complex. A model that has been overfitted will have weak predictive performance on the testing data, as it overreacts to minor changes in the training data and considers outlier values and the noise. Zhu and Wu (16) give two sources of the noise or the random noise that are: the way that this data gathered or from the measurement tool is used to measure the features as useful information. The complex model will fit the training data very well but will result in the loss of its predictive capacity and be a very poor predictive classifier on the testing data.

As can be seen in Figure 2.8, there are three plots, 2.8(a) shows an underfitting model which poorly fits the training dataset and it is expected to perform inaccurately on the testing data, while 2.8(c) model performs very accurate in the training data but it is not expected in the testing data and that because it picks even noises in the training data. However, model in Figure 2.8(b) does not perform as accurate as model in Figure 2.8(c) in the training data but it is expected to have the ability to overcome generalisation error and doing better performance in the testing data.

In order to avoid overfitting, it is often very useful to consider a few different techniques. Examples of such techniques include pruning and cross validation

that can denote when further training is not resulting in better prediction. The basis of these techniques is either to reduce model's complexity, or test its ability to generalise by evaluating model's performance on a set of data not used for training this model.



(a) Underfitting (b) Desired (c) Overfitting

Figure 2.8: Different classifiers fitted to training data. Underfitting where the classifier is simple and poorly classifies the training data, desired where the model classifies the training data accurately but it is not very complex, and overfitting where the classifier is very complex and pick the noises in the training data.

We will talk briefly about cross validation as a method of testing a model's ability in the next section and about pruning in Section 3.4.

2.7 Cross Validation

The most widely used method for estimating prediction error is cross validation (Chapter 7, (17)). Cross validation provides a simple and effective method for both model selection and performance evaluation. Prediction capability on independent testing data relates to the generalisation performance of a learning method. If we have a classifier $d(x_i)$ and it is defined as in Definition 4 on Page 11, the classification error from Equation (2.7) is $R(d) = Pr(d(\mathbb{X}) \neq Y)$ and for

n observations the classification error can be estimated by

$$R(d) = \frac{1}{n}\sum_{i=1}^{n} I\Big(d(x_i) \neq y_i\Big), \tag{2.15}$$

for the classifier d, where $I(\cdot)$ is the indicator function which is 1 if the statement inside the parentheses is true and zero otherwise.

There are two main types of cross validation: K-fold cross validation and stratified cross validation (18).

1. K-fold cross validation
   The observations in the data are randomly divided into K parts and these parts are of approximately equal sizes $n_{(K)} \approx n/K$. We fit a classifier $d(x)$ to $K-1$ parts and predict on the K^{th} part of the data removed.

   The cross validation error of the testing data (K^{th} part removed) $R^{cv}(d^{(K)})$ is:

   $$R^{cv}(d^{(K)}) = \frac{1}{n_{(K)}}\sum_{i=1}^{n_{(K)}} I\big(d^{(K)}(x_i^{(K)}) \neq y_i\big), \tag{2.16}$$

   where $d^{(K)}$ is the K^{th} part classifier and $x_i^{(K)}$ is the i^{th} observation in the K^{th} part, then the overall error is obtained by averaging over all the errors. n-fold cross validation or "leave-one-out" is a special case of K-fold cross validation, where for each i, $i = 1, \ldots, n$, the i^{th} observation is left out and the classifier is learned using the remaining $n-1$ observations. The i^{th} observation is then used as a single-case test sample and the cross validation estimate is the average of i estimates (3).

   The cross validation error $R^{cv}(d^{(n)})$ is:

   $$R^{cv}(d^{n}) = \frac{1}{n}\sum_{i=1}^{n} I\big(d^{(n)}(x_i) \neq y_i\big). \tag{2.17}$$

2. In stratified cross validation, the data is stratified and every fold contains roughly the same proportions of labels as the original dataset (18).

Cross validation uses sampling without replacement, i.e. once the observation is selected, it cannot be drawn again in a particular training/testing datasets and it is used for model selection and performance evaluation. However, bootstrap aggregation, or bagging, uses sampling with replacement from the original data. Bagging is used to create similar bootstrap samples to the training data which gives more variation. We will introduce bagging in more detail in the next section.

2.8 Bagging

Bagging is an ensemble method designed to improve the classification by combining the classifiers of randomly generated bootstrap samples of the training set. Figure 2.9 shows how bagging technique works.

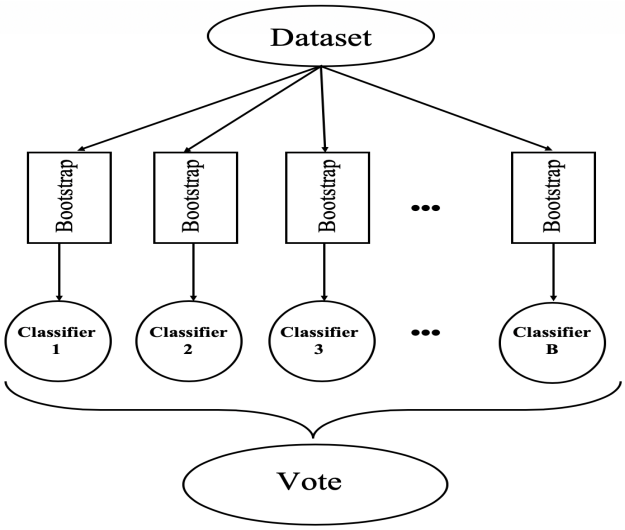


Figure 2.9: Bagging combines many classifiers that are fitted on different bootstrap samples.

We take a classifier $d(x)$, which is defined as in Equation (2.5), with replacing $d(x)$ with $d_b(x)$ to denote a classifier on a bootstrap b. The training dataset is divided to $b = \{1, 2, \ldots, B\}$ bootstraps, and if $d(x)$ predicts a class $j \in \{1, \ldots, J\}$,

then the method of aggregating the decision of $d_b(x)$ is by voting as follows:

$$d(x) = \underset{1 \leqslant j \leqslant J}{\arg\max} \sum_{b=1}^{B} I(d_b(x) = j), \tag{2.18}$$

In a sampling process with replacement, the probability of an observation x to be selected from a set has n observations is $\frac{1}{n}$. The expected number of non-repeated values in the bootstrap sample of size n is $\frac{1}{n}$ and the probability of not selecting the observation x is $(1 - \frac{1}{n})^n$ in the bootstrap sample of size n. The expected number of selecting x is

$$n(1 - (1 - \frac{1}{n})^n) \approx n(1 - \frac{1}{e}),$$

in the bootstrap sample of size n. where $\lim_{n\to\infty}(1 - \frac{1}{n})^n = \frac{1}{e}$. The expected percentage of selecting unique observations is $\frac{n(1-\frac{1}{e})}{n} = 1 - \frac{1}{e} = 63.2\%$, so, we expect to have 63.2% of the unique observations of the training dataset in each bootstrap sample. That means almost two thirds of the training data points are unique and the observations of the last third are repeated (19).

The overlaps between the bootstrap samples creates larger correlation between the classifiers which reduces the model variance. That is an advantage of bagging but at the same time bagging causes a slight increase in the model bias (Page 125, (20)). Due to this reason, bagging works well with some kind of classifiers like tree-based methods as they have low bias and high variance, and this method fixes the high variance problem. We will explain this in more detail in Section 3.7.

Chapter 3

Classification Trees and Random Forests

One of the most widely used classification algorithms are tree-based methods. In this chapter, we will focus on classification decision trees and how to construct them. Section 3.1 explains the way a tree is constructed by splitting recursively the data into two subsets (nodes) until these nodes satisfy specific splitting criterion. These criterion are called impurity functions and there are three impurity functions we are going to discuss in this chapter which are: entropy, Gini index and classification error. Splitting criterion and impurity functions are introduced in Section 3.2. Gini index is the default impurity function used with CART (3), hence, we investigate Gini index and find the expected value of the change in Gini theoretically, and show that the Gini index curve exactly matches the Bayes classifier in case of equally weighted priors as in Section 3.3. Then in Section 3.4, we talk about the tree pruning to find the optimal size of a tree by reducing the number of split levels and to overcome overfitting problem as was discussed in the previous chapter. Following this, we show in Section 3.5 that the monotonically transformed data does not have an impact on decision trees. Finally, we will talk about decision aggregation as a way of aggregate the decision in Section 3.6 and a brief summary of random forests in Section 3.7.

3.1 Constructing Classification Trees

Classification trees are a unique method of modelling multi-dimensional data (21). They use a set of explanatory measurement variables to predict classes. A tree is constructed by recursively partitioning a dataset into purer subsets by using a splitting criterion (22) which will be discussed in more detail in Section 3.2. Each criterion considers a single variable, and it does this in such a way that the dataset is partitioned according to the feature variables to minimise a measure of node heterogeneity (23).

A classification tree consists of nodes (a root node, internal nodes and terminal nodes). An observation is classified by starting at the root node, testing the feature corresponding to the root node, then moving down the tree branch corresponding to the value of the feature variable in the given observation. This procedure continues until the observation reaches a terminal node "leaf node" (24).

Example 1

We have a dataset that has 50 observations, two explanatory variables and a response variable that has three classes A, B and C. Figure 3.1 shows a decision tree where the first node is the root node and it has no incoming edges, this node is at the top of the diagram and it has 21 observations in class A, 14 in class B and 15 in class C. The dataset is split at the root node, and each internal node, into two subsets. This way of splitting the data is a linear classifier and it is made at each split if observations at that node satisfy a query and the children (internal or terminal) nodes are actually more homogeneous than their parent node.

The query at the root node is "Is $V2 \geq 43$?" if yes, then observations go to the left child node, if no then proceed to the right child node. Internal and leaf nodes have one incoming edge. Every node with two outgoing edges is called an internal node, for example, the node in the second level which has 11 observations in class A, 11 in class B

and 13 in class C. Each internal node contains a split which specifies a test based on a single variable. Every node that does not have outgoing edges is called a leaf, for instance the rectangle box that has 10 observations in class A, 3 in class B and 2 in class C, and it is represented by the most frequent class A. The tree continues splitting until it reaches the leaf that has one dominating class, i.e. most observations belong to that class. Alternatively, the leaf may hold a probability vector indicating the probability of the classes having a certain value. Observations are classified by directing them from the root of the tree down to the appropriate leaf. At these leaves, each observation is allocated to a specific class and each leaf node has a class label. The leaf node is described as a pure node if all of its observations belong to one class.

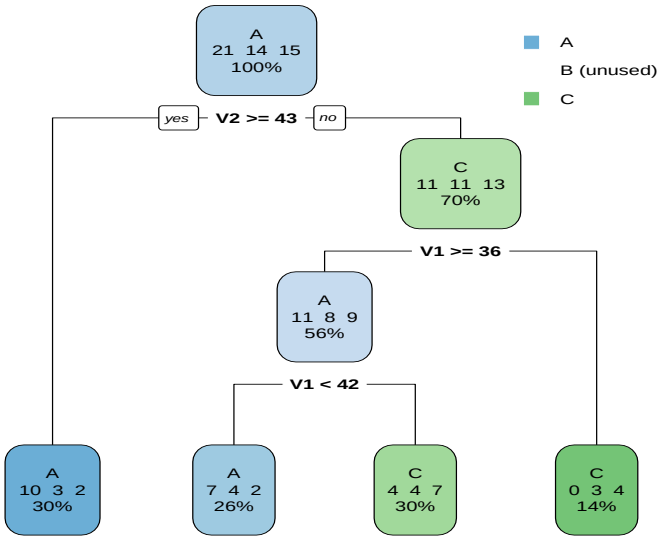


Figure 3.1: A classification tree for dataset of 50 observations. This dataset has two feature variables $V1$ and $V2$, and a response variable with three classes A, B and C. Each internal node represents a split and the numbers in the rectangle boxes are the number of observations in each class and each terminal node is labelled based on the class that is most frequent. The percentage in each node shows the percentage of the total observations in that node.

One can ask how does the classification tree choose a feature variable to split the data at each node? And the answer of this question is by using the classification trees splitting criterion which will be explained in the next section.

3.2 Splitting Criterion (Impurity-based Criteria) and Impurity Functions

3.2.1 Splitting Criterion (Impurity-based Criteria)

Binary tree classifiers are constructed by recursively splitting subsets of the training dataset at the root node into two child nodes. To split the data points at any node into two child nodes, we have to select the splits in a way such that the child nodes are "purer" than their parents. Thus a "goodness of split" criterion is introduced, and this is derived from the notion of an impurity function.

An impurity function is a function ϕ defined on the set of all proportions of number in J possible classes. For a given node c with $n_c > 0$ observations, let $n_{cj} \geq 0$ denote the number of observations in class j, and $j \in \{1, 2, \ldots, J\}$, where $n_c = \sum_{j=1}^{J} n_{cj}$. Let $c \in \{\mathcal{P}, \mathcal{L}, \mathcal{R}\}$, to denote $\mathcal{P}$ (parent), $\mathcal{L}$ (left) and $\mathcal{R}$ (right) nodes, where $\frac{n_{cj}}{n_c}$ is the proportion of observations that are in class j. The impurity function has the following properties:

(a) $\phi \geq 0$.

(b) ϕ is a symmetric function of $(\frac{n_{c1}}{n_c}, \frac{n_{c2}}{n_c}, \ldots, \frac{n_{cJ}}{n_c})$, i.e., if we permute n_{cj}, then ϕ remains constant.

(c) ϕ achieves its maximum at the point $(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J})$, i.e., when the proportion of observations in each class are equal.

(d) ϕ achieves its minimum at the points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)$, i.e., when the proportion in a single class is 1 and 0 for all remaining classes.

Given an impurity function ϕ, the impurity measure of any node c is

$$i(c) = \phi(\frac{n_{c1}}{n_c}, \frac{n_{c2}}{n_c}, \ldots, \frac{n_{cJ}}{n_c}).$$

If a split point $t \in \mathbb{R}$ in a node c divides all observations into two nodes $\mathcal{L}$ and $\mathcal{R}$ with proportions $\frac{n_\mathcal{L}}{n_\mathcal{P}}$ and $\frac{n_\mathcal{R}}{n_\mathcal{P}}$, then the decrease of impurity is defined as:

$$\triangle i(t, c) = i(\mathcal{P}) - \Big(\frac{n_\mathcal{L}}{n_\mathcal{P}} i(\mathcal{L}) + \frac{n_\mathcal{R}}{n_\mathcal{P}} i(\mathcal{R})\Big), \tag{3.1}$$

where $\triangle i(t, c)$ is the difference between the impurity measure for the node c. The classification tree algorithm searches through all possible candidate splits of $\mathcal{P}$ to select the one with the maximum $\triangle i(t, c)$. Examples of impurity functions are Entropy function $N(c)$, Gini index $\Gamma(c)$ and Classification error $C(c)$, and more detail about these functions can be found in Subsection 3.2.2.

The weight of an observation is a measure of the importance of an observation compared to other observations. The more important observations receive more weights and vice versa (Page 179, (25)). Let w_i be the weight of observation x_i, then the weighted version of impurity functions as in Equation (3.1) takes the following form:

$$\triangle i_w(t, c) = i_w(\mathcal{P}) - \left\{ \frac{\sum_{j=1}^{J} W_{\mathcal{L}j}}{W_\mathcal{P}} i_w(\mathcal{L}) + \frac{\sum_{j=1}^{J} W_{\mathcal{R}j}}{W_\mathcal{P}} i_w(\mathcal{R}) \right\}. \tag{3.2}$$

For a given node c, the weight of observations at that node and class $j \in \{1, 2, \ldots, J\}$ is W_{cj} and $W_{cj} = \sum_{x_i \in c} w_i I(y_i = j)$. The weight of observations at node c is $W_c = \sum_{j=1}^{J} \sum_{x_i \in c} w_i I(y_i = j)$, where y_i is the label of i^{th} observation.

3.2.2 Impurity Functions

We will discuss some of impurity functions as follows:

- **Entropy function:** $N(c) = -\sum_{j=1}^{J} \frac{n_{cj}}{n_c} \log_2(\frac{n_{cj}}{n_c})$.

  The change in entropy is

$$\triangle_N = N(\mathcal{P}) - \Big(\frac{n_\mathcal{L}}{n_\mathcal{P}} N(\mathcal{L}) + \frac{n_\mathcal{R}}{n_\mathcal{P}} N(\mathcal{R})\Big). \tag{3.3}$$

The weighted Entropy function is

$$N_w(c) = -\sum_{j=1}^{J} \frac{W_{cj}}{W_c} \log_2(\frac{W_{cj}}{W_c}). \tag{3.4}$$

Entropy function $N(c)$ satisfies impurity function properties as follows:

(a) $N(c) \geq 0$ because $\log_2(x) \leq 0, \forall x \in (0,1)$ and $\frac{n_{cj}}{n_c} \log_2(\frac{n_{cj}}{n_c}) = 0 \log_2 0 \to 0$ when $\frac{n_{cj}}{n_c} \to 0$, property (a)

(b) $N(c)$ has a summation function and the summation function is symmetric, property (b) of ϕ properties

(c) $N(c)$ achieves its maximum at $\log_2(\frac{1}{J})$, property (c)

(d) $N(c)$ achieves its minimum when the node is pure as $\log_2(1) = 0$ and $N(c) = 0$, property (d)

- **Gini index:** $\Gamma(c) = 1 - \sum_{j=1}^{J} (\frac{n_{cj}}{n_c})^2$ and the change in Gini is

$$\triangle_\Gamma = \Gamma(\mathcal{P}) - \left(\frac{n_\mathcal{L}}{n_\mathcal{P}} \Gamma(\mathcal{L}) + \frac{n_\mathcal{R}}{n_\mathcal{P}} \Gamma(\mathcal{R})\right), \tag{3.5}$$

The weighted Gini index is given by

$$\Gamma_w(c) = 1 - \sum_{j=1}^{J} \left(\frac{W_{cj}}{W_c}\right)^2 \tag{3.6}$$

Gini index function $\Gamma(c)$ satisfies impurity function properties as follows:

(a) $\Gamma(c) \geq 0$ because $0 \leq \frac{n_{cj}}{n_c} \leq 1$, property (a)

(b) $\Gamma(c)$ is a symmetric function because it has an addition function and the addition function is symmetric, property (b) of ϕ properties

(c) $\Gamma(c)$ reaches its maximum at $1 - \sum_{j=1}^{J} (\frac{1}{J})^2$, property (c)

(d) $\Gamma(c)$ achieves its minimum when the node is pure as $\frac{1}{j} = 1$ and $\Gamma(c) = 0$, property (d)

- **Classification error:** $C(c) = 1 - \max_j (\frac{n_{cj}}{n_c})$, the classification error is the proportion of the misclassified observations that do not belong to the most common class and the change in classification error is

$$\triangle_C = C(\mathcal{P}) - \left(\frac{n_\mathcal{L}}{n_\mathcal{P}} C(\mathcal{L}) + \frac{n_\mathcal{R}}{n_\mathcal{P}} C(\mathcal{R}) \right). \tag{3.7}$$

The weighted classification error is

$$C_w(c) = 1 - \max_j (\frac{W_{cj}}{W_c}), \tag{3.8}$$

Classification error $C(c)$ satisfies impurity function properties as follows:

(a) $C(c) \geq 0$ because $0 \leq \max_j (\frac{n_{cj}}{n_c}) \leq 1$, property (a)

(b) $C(c)$ is a symmetric function because it has an addition function and the addition function is symmetric, property (b) of ϕ properties

(c) $C(c)$ reaches its maximum at $1 - \sum_{j=1}^{J} (\frac{1}{J})^2$, property (c)

(d) $C(c)$ achieves its minimum when the node is pure as $\frac{1}{j} = 1$ and $C(c) = 0$, property (d)

The most common used impurity function is Gini index and we can rewrite Equation (3.5) in case of two classes, i.e. $J = 2$, so $N_\mathcal{L} = N_{\mathcal{L}1} + N_{\mathcal{L}2}$.

$$\begin{aligned} \frac{n_\mathcal{L}}{n} \Gamma(\mathcal{L}) &= \frac{1}{n} \left\{ n_\mathcal{L} \left(1 - \frac{\sum_{j=1}^{J} \left(n_{\mathcal{L}j} \right)^2}{n_\mathcal{L}^2} \right) \right\} \\ &= \frac{1}{n} \left(n_\mathcal{L} - \frac{\sum_{j=1}^{J} \left(n_{\mathcal{L}j} \right)^2}{n_\mathcal{L}} \right). \end{aligned}$$

$$\begin{aligned}\frac{n_{\mathcal{L}}}{n}\Gamma(\mathcal{L}) &= \frac{1}{n}\left(n_{\mathcal{L}} - \frac{n_{\mathcal{L}1}^2 + (n_{\mathcal{L}} - n_{\mathcal{L}1})^2}{n_{\mathcal{L}}}\right) \\ &= \frac{1}{n}\left(n_{\mathcal{L}} - \frac{n_{\mathcal{L}1}^2 + n_{\mathcal{L}}^2 - 2n_{\mathcal{L}}n_{\mathcal{L}1} + n_{\mathcal{L}1}^2}{n_{\mathcal{L}}}\right) \\ &= \frac{2}{n}\left(n_{\mathcal{L}1} - \frac{n_{\mathcal{L}1}^2}{n_{\mathcal{L}}}\right).\end{aligned}$$

The second term in Equation (3.5) is:

$$\frac{n_{\mathcal{L}}}{n}\Gamma(\mathcal{L}) = \frac{2}{n}\left(n_{\mathcal{L}1} - \frac{n_{\mathcal{L}1}^2}{n_{\mathcal{L}}}\right), \tag{3.9}$$

and the third term is:

$$\frac{n_{\mathcal{R}}}{n}\Gamma(\mathcal{R}) = \frac{2}{n}\left(n_{\mathcal{R}1} - \frac{n_{\mathcal{R}1}^2}{n_{\mathcal{R}}}\right). \tag{3.10}$$

Rewriting the changing in Gini by substituting Equations (3.9) and (3.10) in Equation (3.5), we will have

$$\triangle_{\Gamma} = \Gamma\left(\mathcal{P}\right) - \frac{2}{n}\left[\left(n_{\mathcal{L}1} - \frac{n_{\mathcal{L}1}^2}{n_{\mathcal{L}}}\right) + \left(n_{\mathcal{R}1} - \frac{n_{\mathcal{R}1}^2}{n_{\mathcal{R}}}\right)\right]. \tag{3.11}$$

Example 2

Figure 3.2 shows the normalised curves of the three impurity functions for the same explanatory variable of a simulation which has two equally likely classes and $n = 1000$ observations, such that

$$\text{class } 1 \sim \begin{cases} N(5, 1.8^2), \text{probability } 0.45 \\ N(20, 0.6^2) \end{cases} \quad \text{and class } 2 \sim N(13, 4^2).$$

The choice of the model is done in a way to achieve an overlapping area between the two classes, so we will have a sense of complexity to test the performance of the method we are using.

There are small differences between these impurity function values and have the same minimum point which is the blue dashed line,

however, Gini index and classification error functions have the same maximum point which is the black dashed line while entropy function has a slightly different maximum which is the red dashed line in Figure 3.2.

Decision trees are greedy algorithms which means that they make a sequence of choices. At each choice, they select the best choice among all other available choices at the moment. That is, the greedy algorithm is choosing the optimal solution locally in hope that will lead to the a solution that is good globally. This kind of algorithms rarely provides optimal solutions (26).

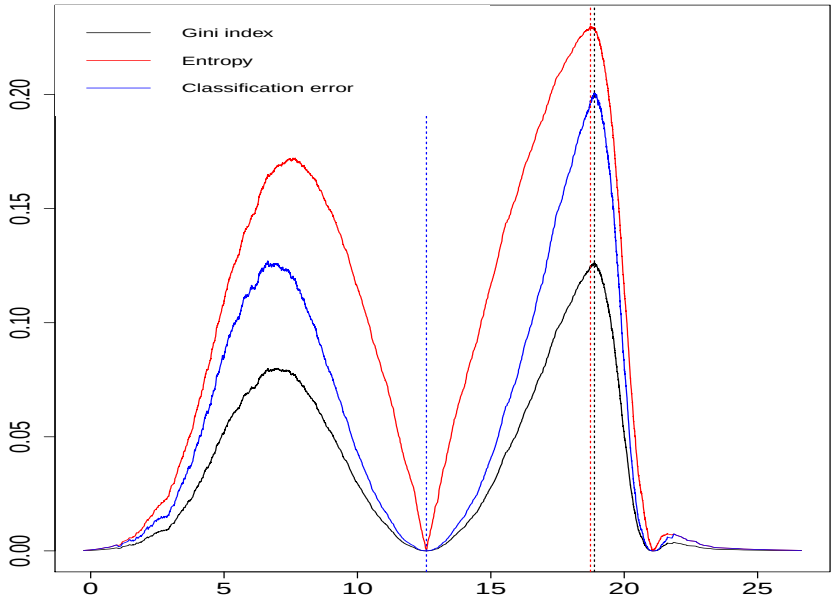


Figure 3.2: Different impurity functions as a function of split points. The vertical lines show minimum and maximum points of the three curves. The blue dashed line shows the minimum point of all three curves while Gini index and classification error functions have the same maximum point which is the black dashed line and entropy function has a slight different maximum point which is at the red dashed line.

An important topic is how to obtain the optimal size of a tree. We will discuss a method to reduce the size of a tree when it becomes too complicated, which is tree pruning in Section 3.4.

3.3 Expected Value of Gini Gain

From the previous section, we have the formula of Gini index at a node c is given as

$$\Gamma(c) = 1 - \sum_{j=1}^{J} \left(\frac{n_{cj}}{n_c}\right)^2,$$

where n_{cj} is the number of observations in class j at node c. The node c is the region of space defined by the split point $t \in \mathbb{R}$, $c = \{\mathcal{P}, \mathcal{L}, \mathcal{R}\}$, where $\mathcal{P}, \mathcal{L}, \mathcal{R}$ denote parent, left and right nodes, respectively.

$$\triangle_\Gamma = \Gamma(\mathcal{P}) - \left(\frac{n_\mathcal{L}}{n}\Gamma(\mathcal{L}) + \frac{n_\mathcal{R}}{n}\Gamma(\mathcal{R})\right). \tag{3.12}$$

To find the expected value of the change in Gini as in Equation (3.5), take the count to the left and to the right $n_\mathcal{L}, n_\mathcal{R}$, and Gini index at the left node and the right $\Gamma(\mathcal{L}), \Gamma(\mathcal{R})$ to be random variables. Re-writing Equation (3.5) in terms of the random variables and the split point t gives:

$$\triangle_\Gamma = \Gamma(\mathcal{P}) - \left(\frac{N_\mathcal{L}}{n}\Gamma(\mathcal{L}) + \frac{N_\mathcal{R}}{n}\Gamma(\mathcal{R})\right). \tag{3.13}$$

As we know from Definition 2 in Chapter 2, the density function of a random value $\mathbb{X}$ is

$$f_\mathbb{X}(x) = \sum_{j=1}^{J} \pi_j f_j(x),$$

where π_j is the probability of class j and f_j is the probability density function of class j.

The probability of class j for a given point x is

$$Pr(Y = j | \mathbb{X} = x) = \frac{\pi_j f_j(x)}{f_\mathbb{X}(x)}.$$

The count of class j to the left of a split point $t \in \mathbb{R}$ is

$$N_{\mathcal{L},j}(t) = \sum_{i=1}^{n} I_{\{\mathbb{X} \leqslant t, Y_i = j\}} \tag{3.14}$$

The event $I_{\{\mathbb{X} \leqslant t, Y_i = j\}}$ is the outcome of a Bernoulli trial and the probability of that event is:

$$\begin{aligned} Pr(I_{\{\mathbb{X} \leqslant t, Y_i = j\}} = 1) &= Pr(\mathbb{X} = x_i \leqslant t, Y_i = j), \forall i = 1, \ldots, n \\ &= \pi_j F_j(t), \end{aligned}$$

where $F_j(t)$ is the cumulative distribution function of class j. Then the distribution of the sum of n Bernoulli random variables is a binomial distribution, so

$$N_{\mathcal{L},j}(t) = \sum_{i=1}^{n} I_{\{\mathbb{X} \leqslant t, Y_i = j\}} \sim B(n, \pi_j F_j(t)),$$

with probability mass function

$$Pr(N_{\mathcal{L},j}(t) = k) = \binom{n}{k} (\pi_j F_j(t))^k (1 - \pi_j F_j(t))^{n-k}. \tag{3.15}$$

Similarly, the probability of occurrence of an observation to the left of a split point $t \in \mathbb{X}$ is relates to a Bernoulli trial with a random variable $I_{\{\mathbb{X} \leqslant t\}}$ and probability is as follows:

$$\begin{aligned} Pr(I_{\{\mathbb{X} \leqslant t\}} = 1) &= Pr(\mathbb{X} \leqslant t) \\ &= \sum_{j=1}^{J} Pr(Y_i = j) Pr(\mathbb{X} \leqslant t | Y_i = j) \\ &= F(t), \end{aligned}$$

where $F(t) = \sum_{j=1}^{J} \pi_j F_j(t)$ and $\forall i = 1, \ldots, n$.

Then, the distribution of n Bernoulli trials is a binomial distribution, i.e.

$$N_L(t) = \sum_{i=1}^{n} I_{\{\mathbb{X} \leqslant t\}} \sim B(n, F(t)),$$

with probability

$$Pr(N_{\mathcal{L}}(t) = k) = \binom{n}{k} (F(t))^k (1 - F(t))^{n-k}. \tag{3.16}$$

Using standard results for the Binomial distribution, the expected value of the count in Equation (3.14) is:

$$E\Big(N_{\mathcal{L},j}(t)\Big) = n\pi_j F_j(t). \tag{3.17}$$

The count of all classes to the left of the split point has expected value:

$$E\Big(N_{\mathcal{L}}(t)\Big) = nF(t). \tag{3.18}$$

Similarly, the count of class j to the right of the split point $t \in \mathbb{R}$ is $N_{\mathcal{R},j}$ and the expected value is:

$$E\Big(N_{\mathcal{R},j}(t)\Big) = n\pi_j \bar{F}_j(t), \text{where} \quad \bar{F}_j(t) = 1 - F_j(t), \tag{3.19}$$

and the expected value of the number of sample to the right of the split $t \in \mathbb{R}$ is:

$$E\Big(N_{\mathcal{R}}(t)\Big) = n\bar{F}(t), \text{and} \quad \bar{F}(t) = \sum_{j=1}^{J} \pi_j \bar{F}_j(t). \tag{3.20}$$

Recall that the Gini index at a node c is given by Equation (3.5). Here, we have $\Gamma(\mathcal{P})$ is a constant and it depends on the number of observations in each class.

Then by taking the expected value of Equation (3.9) and considering $n_{\mathcal{L}1}$ and $n_{\mathcal{L}}$ are random, we have :

$$E\left(\frac{N_{\mathcal{L}}}{n}\Gamma(\mathcal{L})\right) = \frac{2}{n}\left(E(N_{\mathcal{L}1}) - E\left(\frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}}\right)\right). \tag{3.21}$$

Now, $E(N_{\mathcal{L}1}) = n\pi_1 F_1(t)$ from Equation (3.17), and from the law of total expectation:

$$\begin{aligned} E\left(\frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}}\right) &= \sum_{n_{\mathcal{L}}=1}^{n} E\left(\frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}} \;\middle|\; N_{\mathcal{L}} = n_{\mathcal{L}}\right) Pr(N_{\mathcal{L}} = n_{\mathcal{L}}) \\ &= \sum_{n_{\mathcal{L}}=1}^{n} E\left(\frac{N_{\mathcal{L}1}^2}{n_{\mathcal{L}}} \;\middle|\; N_{\mathcal{L}} = n_{\mathcal{L}}\right) Pr(N_{\mathcal{L}} = n_{\mathcal{L}}) \end{aligned}$$

Since $N_{\mathcal{L}1} \sim B\Big(n_{\mathcal{L}}, p'\Big)$, with $p' = \frac{\pi_1 F_1(t)}{F(t)}$, then

$$E\left(N_{\mathcal{L}1}^2 \;\middle|\; N_{\mathcal{L}} = n_{\mathcal{L}}\right) = n_{\mathcal{L}}p'(1-p') + n_{\mathcal{L}}^2 p'^2,$$
$$E\left(\frac{N_{\mathcal{L}1}^2}{n_{\mathcal{L}}} \;\middle|\; N_{\mathcal{L}} = n_{\mathcal{L}}\right) = p'(1-p') + n_{\mathcal{L}}p'^2,$$

and so,

$$\begin{aligned} E\left(\frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}}\right) &= \sum_{n_{\mathcal{L}}=1}^{n} \left\{p'(1-p') + n_{\mathcal{L}}p'^2\right\} Pr(N_{\mathcal{L}} = n_{\mathcal{L}}) \\ &= p'(1-p') + p'^2 E(N_{\mathcal{L}}) \\ &= \frac{\pi_1 F_1(t)\pi_2 F_2(t)}{F^2(t)} + \frac{\pi_1^2 F_1^2(t)}{F^2(t)} nF(t). \end{aligned}$$

By substituting $E(N_{\mathcal{L}1})$ and $E\left(\frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}}\right)$ in Equation (3.21), then we will have

$$E\left(\frac{N_{\mathcal{L}}}{n}\Gamma(\mathcal{L})\right) = \frac{2}{n}\left\{n\pi_1 F_1(t) - \frac{\pi_1\pi_2 F_1(t)F_2(t)}{F^2(t)} - n\frac{\pi_1^2 F_1^2(t)}{F(t)}\right\}. \tag{3.22}$$

By following the same steps

$$E\left(\frac{N_{\mathcal{R}}}{n}\Gamma(\mathcal{R})\right) = \frac{2}{n}\left\{n\pi_1 \bar{F}_1(t) - \frac{\pi_1\pi_2 \bar{F}_1(t)\bar{F}_2(t)}{\bar{F}^2(t)} - n\frac{\pi_1^2 \bar{F}_1^2(t)}{\bar{F}(t)}\right\}$$

Finally,

$$\begin{aligned} E\big(\triangle_\Gamma\big) &= \Gamma(\mathcal{P}) - \frac{2}{n}\left\{n\pi_1 - \frac{\pi_1 F_1(t)\pi_2 F_2(t)}{F^2(t)} - n\frac{\pi_1^2 F_1^2(t)}{F(t)} - \frac{\pi_1 \bar{F}_1(t)\pi_2 \bar{F}_2(t)}{\bar{F}^2(t)} - n\frac{\pi_1^2 \bar{F}_1^2(t)}{\bar{F}(t)}\right\} \\ &= \Gamma(\mathcal{P}) - \frac{2\pi_1}{n}\left\{n - \pi_2\left(\frac{F_1(t)F_2(t)}{F^2(t)} + \frac{\bar{F}_1(t)\bar{F}_2(t)}{\bar{F}^2(t)}\right) - n\pi_1\left(\frac{F_1^2(t)}{F(t)} + \frac{\bar{F}_1^2(t)}{\bar{F}(t)}\right)\right\} \end{aligned} \tag{3.23}$$

Within the curly brackets, the second and third terms are $O(n)$ and the fourth and fifth terms are $O(1)$. Hence, for large n we get asymptotic approximation gives:

$$\begin{aligned} E\big(\triangle_\Gamma(t)\big) &\approx \Gamma\left(\mathcal{P}\right) - 2\pi_1\left\{1 - \pi_1\left(\frac{F_1^2(t)}{F(t)} + \frac{\bar{F}_1^2(t)}{\bar{F}(t)}\right)\right\} \\ &= \Gamma\left(\mathcal{P}\right) - 2\pi_1\left\{\pi_2 - \frac{\pi_2\big(\pi_1\pi_2 F_1^2(t) - F_1(t)F_2(t) + F_2^2(t)\big)}{F(t)(1-F(t))}\right\} \\ &= \Gamma\left(\mathcal{P}\right) - 2\pi_1\pi_2\left\{1 - \frac{\pi_1\pi_2\big(F_1(t) - F_2(t)\big)^2}{F(t)(1-F(t))}\right\}. \end{aligned} \tag{3.24}$$

$F(t)$ can be considered as a binomial random variable (27), which gives that $E[F(t)] = F(t)$ and $Var[F(t)] = F(t)(1-F(t))$. The term $\frac{\big(F_1(t)-F_2(t)\big)^2}{F(t)(1-F(t))}$ is the squared difference between the two cumulative distributions divided by the variance of the weighted sum of the two cumulative functions $F(t) = \sum_{j=1}^{J} \pi_j F_j(t)$.

By applying this formula on a model 1, where $C_1 \sim N(13, 1^2)$ and $C_2 \sim N(15, 1^2)$, and a model 2, where $C_1 \sim \begin{cases} N(5, 1.8^2), Pr = 0.45 \\ N(20, 0.6^2) \end{cases}$ and $C_2 \sim N(13, 4)$, where C_j denotes class j, the expected values are as in Figure 3.3 and Figure 3.4. The two figures show a good matching between the expected value of Gini index curves and the average of them. The grey lines show the standard error of the average (mean) of all 500 curves.

Figure 3.5 shows Bayes classifiers and the expected value of the change in Gini index for model 1 in case of equal or different priors. It can be seen that the highest point in the expected value of the change in Gini index curve matches Bayes classifier when the priors are equals in both classes and a little different when the priors are unequal. In case of different priors the green curve shifts towards the class that has larger prior. In Figure 3.5(b), the prior of the first class is $\pi_1 = 0.7$ and $\pi_2 = 0.7$ in Figure 3.5(c).

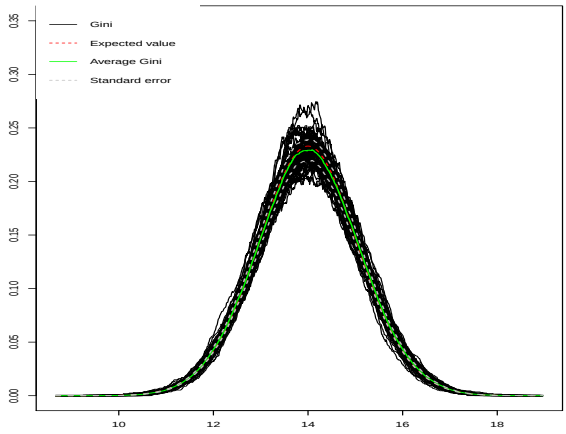


Figure 3.3: The expected value of 500 simulations from model 1. The green line shows the average of 500 Gini curves and the dashed red line shows the expected value.

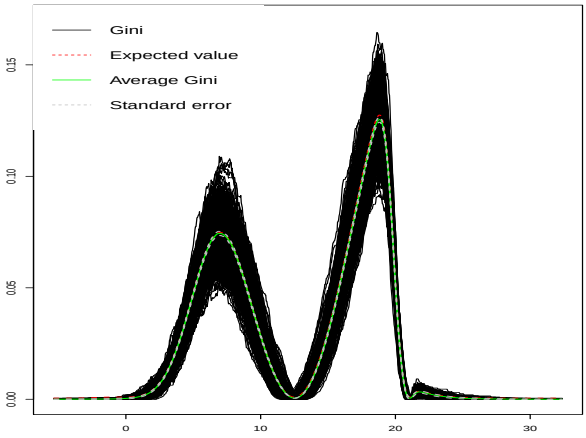
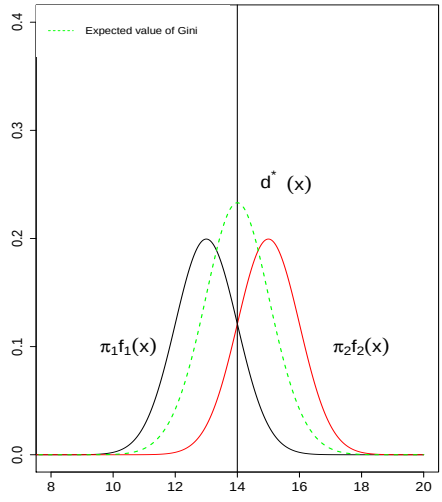
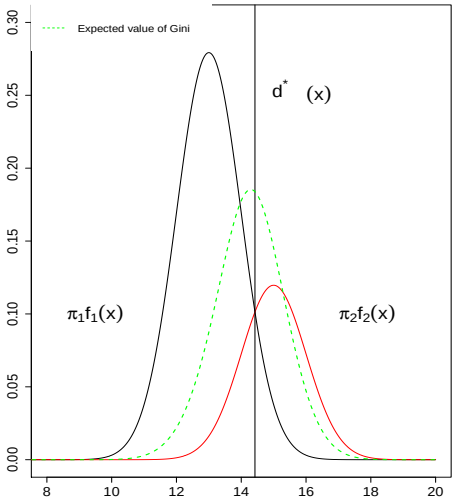


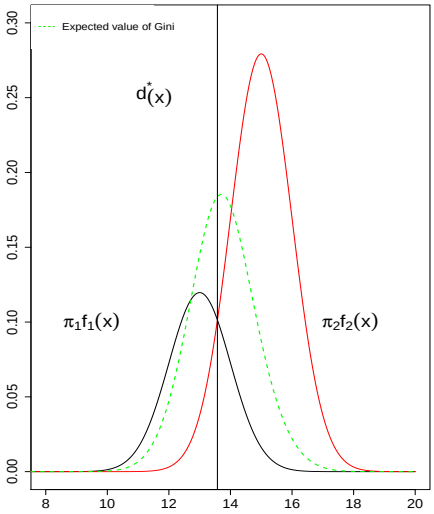
Figure 3.4: The expected value of 500 simulations from model 2. The green line shows the average of 500 Gini curves and the dashed red line shows the expected value.



(a) Equal priors, the Bayes classifier and highest point in the expected value of Gini are at 13.99.



(b) Different priors $\pi_1 = 0.7$, the Bayes classifier is at 14.4 and highest point in the expected value of Gini is at 14.3.



(c) Different priors $\pi_2 = 0.7$, the Bayes classifier is at 13.6 and highest point in the expected value of Gini is at 13.71.

Figure 3.5: These plots show Bayes classifiers $d^*(x)$ and the expected value curves (green dashed lines) in case of equal or different priors in model 1.

3.4 Tree Pruning

Growing a complex tree and pruning it back is the most prevalent approach to constructing trees whilst attempting to limit the effect of overfitting. Pruning is suitable because the resulting tree from the algorithm may overfit the data by growing more structure than is justified by the training dataset. Specifically, if there are no misclassified observations, the training set error of a large tree is zero, whilst the generalisation error "test error" is expected to be larger. To solve the overfitting problem, the overgrown tree is "pruned" back with the aim of determining the tree with the lowest error rate on the testing data (28).

There are different methods of pruning, one such method is known as cost complexity pruning which is based on the minimal cost complexity pruning method of the CART decision tree (3).

We need to minimise the cost complexity function here,

$$R_\alpha(T) = R(T) + \alpha|d(T)|, \tag{3.25}$$

where $R(T)$ is the classification error, $d(T)$ is the decision tree which gives a set of leaves and $\alpha \geq 0$ represents a complexity parameter. The size of a tree depends on the complexity parameter α, when $\alpha \to \infty$, the tree is pruned and the number of leaves reduced until this tree becomes a root, and when $\alpha \to 0$, the tree remains large.

For a given node c, $R(T) = \sum_{c \in d(T)} r(c)Pr(c)$, where $r(c)$ is the classification error for points in node c, $Pr(c)$ is the proportions of data falling into the region corresponding to that node.

Example 3

In USPS dataset (29) which contains handwritten digits obtained from envelopes by the U.S. Postal Service, the digits were written by many different people, using a large variety of writing sizes and styles. These images have been processed and size normalised, resulting in 16 x 16 greyscale images, examples of which can be seen in Figure 3.6.

Furthermore, the grey levels of every image are scaled to fall within the interval $[-1, 1]$ (29). There are 7291 training observations and 2007 test observations, distributed as in Table 3.1:

	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

Table 3.1: The frequencies of the observations in classes $0, 1, \ldots, 9$ in the training and testing datasets from a dataset containing handwritten digits obtained from envelopes by the U.S. Postal Service.

We fit a tree using the `rpart` command from `rpart` package (30) in R (31) and change the complexity parameter `cp` in `rpart.control` for this data and then compare between the percentage of correctly classified observations in the training and testing datasets as in Figure 3.7. We run an experiment to see how changing the complexity parameter affects the tree decision accuracy. The default range of `cp` in `rpart` is `cp` ≥ 0 but we take `cp` $\in (0, 0.0002)$. The percentage of correctly predicted classes in the testing data is around 82% in the first area while the the percentage of correctly predicted classes in training data is 100%. The reason behind this slightly poor performance in the testing data is that the model is too complicated and it causes an overfitting problem due to the fact that the small `cp` values make the model more sensitive to noise. Then, when the percentages of performance accuracy of the training data decreased in the second area to become 94%, the model gives a more accurate performance and the percentages in the testing sets are around 84% in that interval. Finally, the performance decreased in both training and testing data in the third area as the model becomes more simple and do not classify data so accurately.

This illustrates that the more complex the model, the larger difference between train and test errors. The ideal way of choosing α, which is

`cp` in `rpart` command, to perform accurately is to have a model is able to give the highest accuracy error in the testing data and has the least possible complexity. While Figure 3.8 shows a tree is automatically stopped from growing up extra nodes when this tree satisfies a certain splitting criterion, Figure 3.9 shows a tree which is forced to grow to its maximum size by changing `rpart.control` parameters.

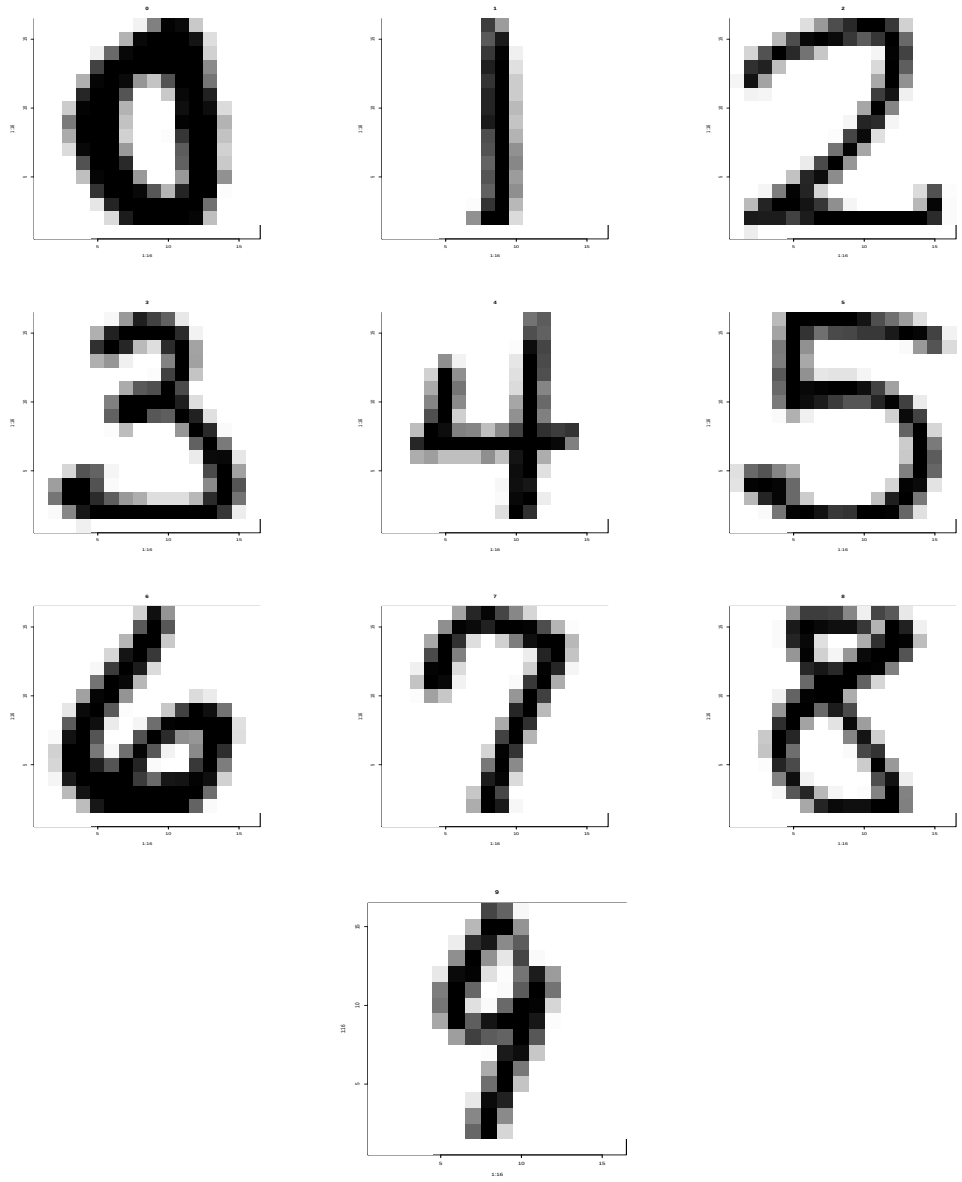


Figure 3.6: Examples of normalised, centred and scaled digits from the training set USPS.

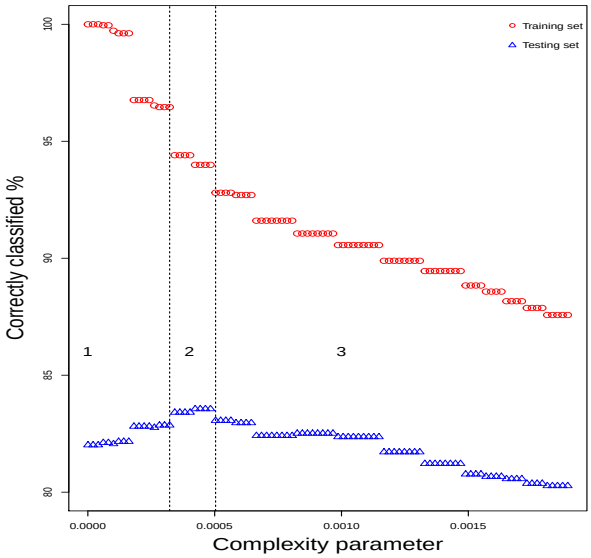


Figure 3.7: Comparing between the training and testing datasets in handwritten digits data with different complexity parameters. It is clear that when the percentage of correctly classified observations decreased in the training dataset the percentage of correctly classified observations increased in the testing dataset in the first and second regions. First, second and third areas show an overfitting, desired and underfitting situations as explained in Section 2.6

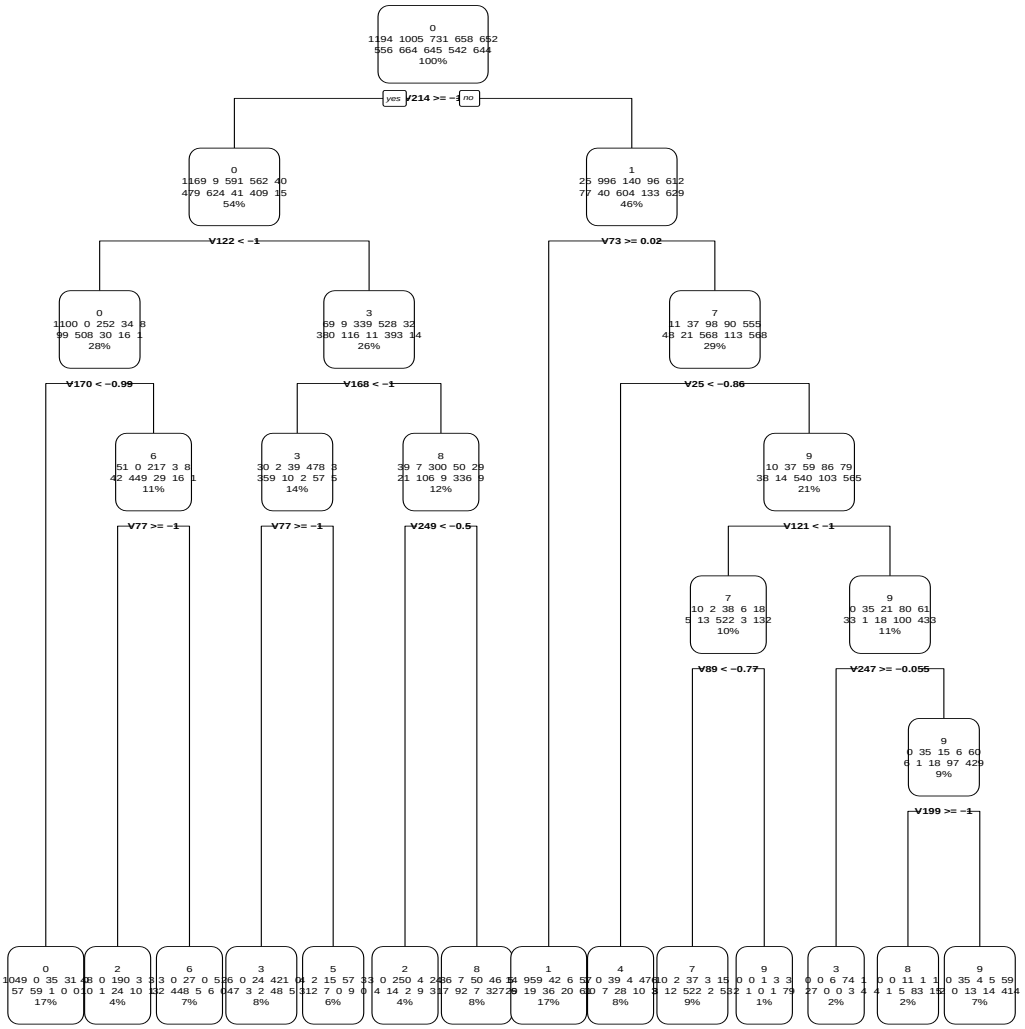


Figure 3.8: A classification tree fitted on the training data from handwritten digits data USPS. Each rectangle box represents the leaf and contains a dominating class. It is clear that every leaf is a mixture of most of the classes' observations.

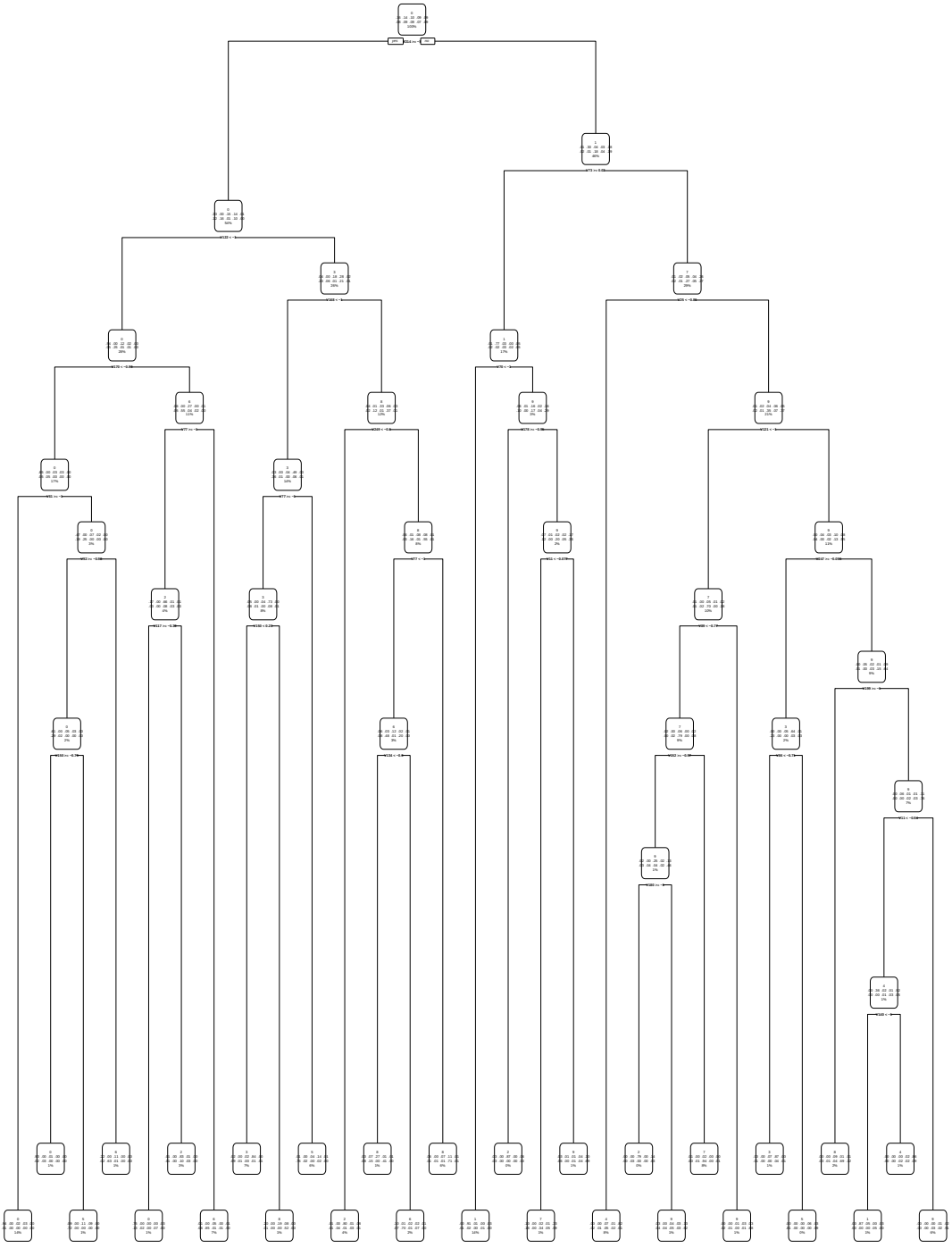


Figure 3.9: The maximum size tree of handwritten digit data when complexity parameter 1×10^{-11}.

However, pruning is not relevant to the method of this thesis as we consider random forests, in which every tree is grown to the maximum possible size without pruning. This method will be explained further in Section 3.7.

Researchers need sometimes to change the data variable measurements for some reason in some researches or in some data collection circumstances and using decision trees in a monotonically transformed data does not affect structure of the tree and hence the accuracy, we will discuss this in more detail in the next section.

3.5 Decision Trees with Monotonically Transformed Data

Definition 7. *$M : \mathbb{R} \to \mathbb{R}$ is a strictly monotonic function if $M(a) \leqslant M(b), \forall a \leqslant b$ or $M(a) \geqslant M(b), \forall a \geqslant b$.*

We apply monotonic functions to just the explanatory variables in data frames (monotonic functions are not applied to the response variable), i.e. $\tilde{\mathbb{X}} = M(\mathbb{X})$ and $\tilde{Y}_i = Y_i$, to tackle for example changing in the measurements from imperial to metric or vice versa.

If we have observations $x_1, x_2, \ldots, x_n$ and $t \in \mathbb{R}$ is the split point, such that $x_i \leqslant t$ or $x_i > t$, then $M(x_i) \leqslant M(t)$ or $M(x_i) > M(t)$. A monotonic transformation does not change the order of observations. Also, as we can see from Gini index

$$\Gamma(c) = 1 - \sum_{j=1}^{J} \left(\frac{n_{cj}}{n_c}\right)^2,$$

as in Section 3.2, for a given split point $t \in \mathbb{R}$, $n_c(t) > 0$ denotes the number of observation at node c and the split point t and $n_{cj}(t) \geq 0$ denote the number of observations in class j, $j \in \{1, 2, \ldots, J\}$, such that $n_c(t) = \sum_{j=1}^{J} n_{cj}(t)$.

Rewriting Gini index

$$\Gamma_c(t) = 1 - \sum_{j=1}^{J} \left(\frac{n_{cj}(t)}{n_c(t)} \right)^2, \tag{3.26}$$

we can see that Gini depends on number of observations to the left and the right of the split point t and it does not change by the monotonic function which preserves points order, i.e. $\Gamma_c(t) = \tilde{\Gamma}_c(M(t)), \forall t$.

There are many strictly monotonic functions, for instance, a linear function (which takes the form $a' + b'x$), quadratic if $x \geqslant 0$ and logarithmic if $x > 0$. The adding (or subtracting) and multiplying (or dividing) operations are called shifting and scaling the distribution of the data, respectively, and they change the location and the shape of the data points whilst preserving their order.

The monotonic transformations give nearly equivalent tree to the data without transformation. We will use the monotonic transformation of the data in Chapter 6.

Example 4

A logarithmic function log is applied to the explanatory variables in the `iris` data (32) and it gives us a tree as in Figure 3.10. This tree does not show any different in terms of decisions to the tree that used `iris` without a transformation as in Figure 3.11. The only difference between the two trees (with and without transforming the data) is in the split point values and they are approximately equal to the logarithm of the split point values for the tree without transforming the data.

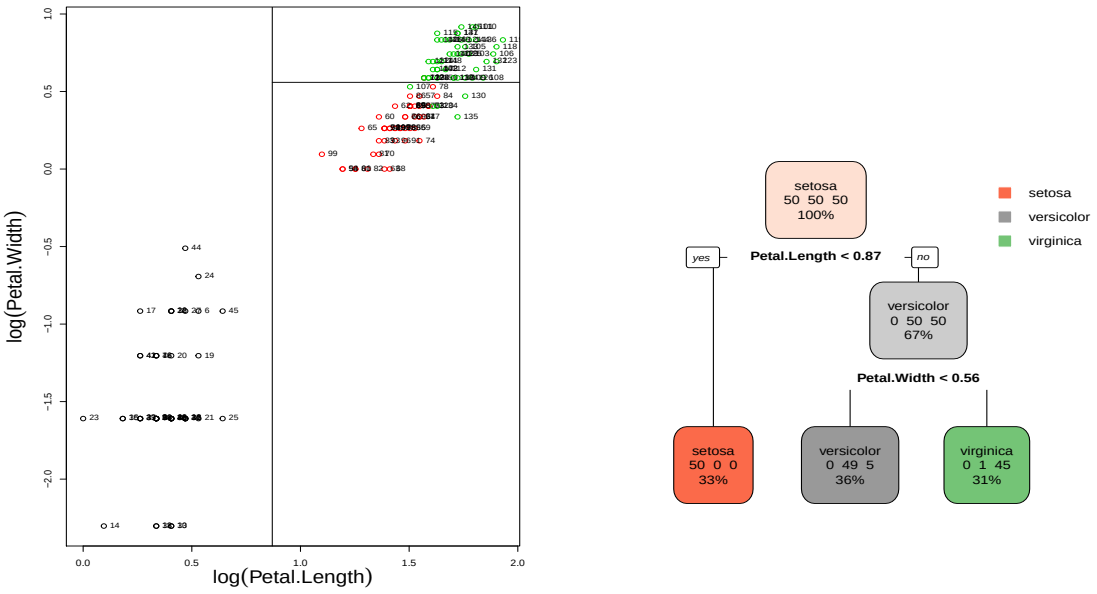


Figure 3.10: The plot to the left shows the split points and split variables in a decision tree, to the right, which used `iris` data that is logarithmically transformed.

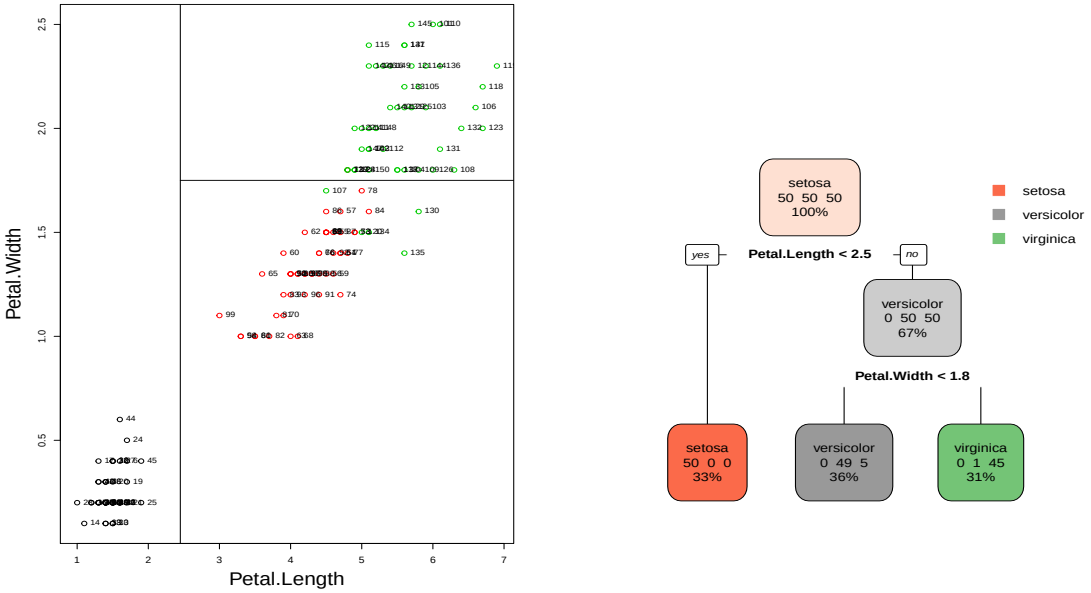


Figure 3.11: The plot to the left shows the split points and split variables in a decision tree, to the right, that used the untransformed `iris`.

3.6 Decision Aggregation

The aggregated decision is made by an aggregation function. The choice of aggregation function plays an important role in many disciplines such as statistics, computer science and finance. It gives a single decision based on combining several aggregated decisions in which the decision represents all the individual combined decisions well and for further information see (33). There are many forms of aggregation functions like means (arithmetic, geometric, etc), mode, sum, minimum and maximum functions. In this section, we will talk about two aggregation methods:

3.6.1 Majority Vote

Majority vote (mode) is used in bagging, as was introduced in Section 2.8, as an aggregation decision function for classification. A majority vote method of

aggregating decisions is counting the votes for each classifier and considering the most highly voted class as the final decision. If $d(x)$ predicts a class j, then the method of aggregating the $d_b(x)$ is by voting and the formula is as in Equation (2.18).

3.6.2 Weighted Vote

A weighted vote is another way of combining classifiers. This method can be explained by the following illustrative example:

Consider a dataset that contains n observations, two classes $\mathcal{A}$ and $\mathcal{B}$, and a single explanatory feature variable $\mathbf{x}$. A forest of three stumps $\{(1), (2), (3)\}$ as in Figure 3.12 is grown on this dataset and the classification decision is made by using one of two methods: majority vote (MV) or weighted vote (WV). Assume an observation x_i, where $i = \{1, 2, \ldots, n\}$, is equal to 30 and from the information in Figure 3.12, the weighted vote works in a way which, x_i is in the left leaf in the first decision stump (1), so it has two proportions, $\hat{\pi}_{\mathcal{A}1} = 0.52$ and $\hat{\pi}_{\mathcal{B}1} = 0.48$, where $\hat{\pi}_{\mathcal{A}1}$ denotes the proportion of class $\mathcal{A}$ in the first tree (1). The same rule is applied to the other two stumps, the position of x_i and proportions are as in Table 3.2. By adding up $\sum_{b=1}^{B} \hat{\pi}_{jb}$, where B is the number of stumps, for each class j separately, then we choose the class which maximises this summation $\sum_{b=1}^{B} \hat{\pi}_{jb}$.

$$d(x) = \underset{1 \leqslant j \leqslant J}{\arg\max} \sum_{b=1}^{B} \hat{\pi}_{jb} \; I(d_b(x) = j). \tag{3.27}$$

Here, this observation will be allocated to class $\mathcal{B}$ due to its added-up proportion 1.53 which is greater than class $\mathcal{A}$ with added-up proportion 1.47. It is clearly shown that this observation is allocated to class $\mathcal{B}$ by using the weighted vote while it is allocated to class $\mathcal{A}$ by using the majority vote.

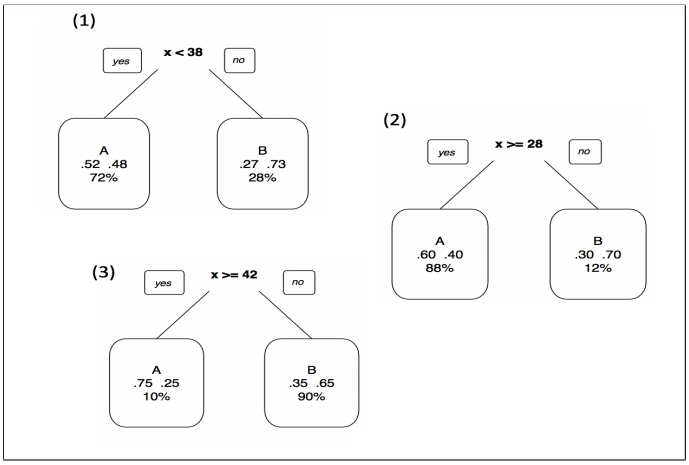


Figure 3.12: An example of aggregation methods applied on a forest of three stumps. The percentages indicate the percentages of the total number of observations in each leaf regardless of classes.

Stump	Leaf	Majority vote	Weighted vote	
			$\hat{\pi}_{\mathcal{A}}$	$\hat{\pi}_{\mathcal{B}}$
(1)	Left	$\mathcal{A}$	0.52	0.48
(2)	Left	$\mathcal{A}$	0.60	0.40
(3)	Right	$\mathcal{B}$	0.35	0.65
			1.47	1.53
Decision		$\mathcal{A}$	$\mathcal{B}$	

Table 3.2: The positions of x_i in the dataset and the accordance decisions.

3.7 Random Forests

Random forests, or random decision forests (34), are an ensemble learning method for classification and regression. Random forests proceed by constructing a group of decision trees on training data and outputting the class which is the mode (majority vote) of the classes (classification) or the mean prediction (regression) of the individual trees. Random forests are an alteration of bagging that grow a large number of de-correlated trees. Moreover, the expectation of the average of B trees is the same as the expectation of any one of them because every fitted tree in bagging is identically distributed. This means that the bias of bagged trees is the same as the bias of one tree. This implies that the only chance to improve random forest performance is through variance reduction (15). To explain that, the variance of an average of B identical independent distributed random variables, each having variance σ^2, is $\frac{1}{B}\sigma^2$. If the variables are identical but not independent, having positive correlation ρ, the variance of the average is:

$$
\begin{aligned}
Var\left(\frac{1}{B}\sum_{i=1}^{B} T_i(x)\right) &= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B}\mathrm{Cov}\Big(T_i(x), T_j(x)\Big) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\left(\sum_{i\neq j}\mathrm{Cov}\big(T_i(x), T_j(x)\big) + Var\big(T_i(x)\big)\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\Big((B-1)\sigma^2\cdot\rho + \sigma^2\Big) \\
&= \frac{B(B-1)\rho\sigma^2 + B\sigma^2}{B^2} \\
&= \frac{(B-1)\rho\sigma^2}{B} + \frac{\sigma^2}{B} \\
&= \rho\sigma^2 - \frac{\rho\sigma^2}{B} + \frac{\sigma^2}{B} \\
&= \rho\sigma^2 + \sigma^2\frac{1-\rho}{B}.
\end{aligned}
\tag{3.28}
$$

The first term in Equation (3.28) decreases as ρ decreases, and the second term disappears as B increases.

As we have seen previously in bagging, that constructing of each tree is using a different bootstrap sample X_b of the data. Whilst in standard trees, each node is split using the best split among all variables, Breiman (6) in random forests added a new level of randomness which is split at each node using the best split among a subset p^* from the p feature variables randomly chosen at that node. The growing procedure for random forests is described in Algorithm 1.

Random forest tree selects, at each node, a subset p^* at random of the p feature variables to be candidates for splitting data at that node and the variable that results in the maximum decrease in Gini index is chosen for that split. Figure 3.13 shows an example of a bootstrap X_b of the training data (red) that is used to build a tree inside a random forest and the chosen observations and a subset p^* feature variables change at each node within each tree randomly (green).

A typical value for p^* is $\sqrt{p}$, rounded down to the nearest integer, with a minimum node size of 1 as the default in classification in `RandomForest` command in (35) in R.

Algorithm 1 Random Forests for Classification

For$(b \leftarrow 1 \quad to \quad B)$ **do**

1. Draw a bootstrap sample of size n from the training data
2. Fit a random-forest tree T_b to the bootstrapped sample by repeating recursively the following steps for each node of the tree, until the minimum node size is reached
   (a) Select p^* variables randomly from the p variables
   (b) Choose the best feature among the p^*
   (c) Split this node into two child nodes

Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point z from the testing data:

Let $d_b(z)$ be the classifier of the b^{th} random-forest tree.

Then $d(z) = \underset{1 \leqslant j \leqslant J}{\arg\max} \sum_{b=1}^{B} I(d_b(z) = j)$.

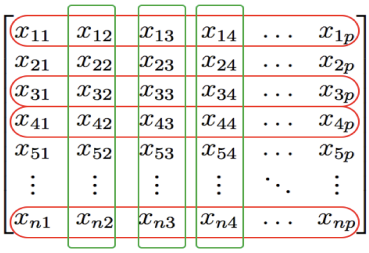


Figure 3.13: A bootstrap sample (red) X_b that is used to fit a tree within a random forest with a subset p^* (green) of feature variable that is changed at each node within each tree.

Chapter 4

Weighted Samples in Random Forests

The goal of this chapter is to investigate the effect of changing sampling method from bootstrapping to reweighting the observations according to a distribution, in an attempt to investigate whether this change will cause any difference to the distribution of the first split of each tree within the random forest. As we will see in Subsection 4.1.2 that changing observation weight has an effect on enhancing the accuracy of the tree performance. The reason why the first split of each tree is analysed is due to the fact that the first splitting variable and split point has the higher ranking in Gini gain (or information gain), which means this variable attempts to split the data better from the beginning of the tree. The better splitting from the beginning leads to a more accurate prediction. Before starting investigating the effect of changing the weighting schemes, we will study how to change the weights of observations (or the prior of classes) in decision trees as in Section 4.1. Section 4.2 shows investigating schemes of weighted samples in random forests, we will explore whether changing the sampling mechanism from bagging to giving observations weights based on a distribution has an impact on the distribution of the first split of each tree within the random forest or not. Then, we present the results of this exploring in Section 4.3 using simulations and real world datasets.

4.1 Trees with Priors and Weights

It is possible that the distribution of the J classes in the training data is not the same as the perceived distribution of these classes in the whole population. A common example in which training data does not represent the population is credit scoring. The training data of a credit scoring shows the customers who default or pay back their loans (good and bad loans) but it does not show the part of the population which is customers who were not given loans in the first place, so, they are a part of the population of loan applicants but not represented in the training data.

In the instance where it is believed that the training data is not distributed in the same manner as the population, it is often desirable to impact the way in which the tree is trained. There are two ways to overcome problems which occur because of unbalanced data which are: changing the priors (from the classes' proportions to assumed priors) or changing observation weights. Changing priors is used when a class proportion in the training data does not represent the class proportion in the population. Changing weights is applied on observations which are more important. We can see that priors are associated with classes and weights with observations. Subsections 4.1.1 and 4.1.2 show trees with priors and weights, and Subsection 4.1.3 shows how to make weights act like priors.

4.1.1 Trees with Priors

The first way to deal with a difference in distribution between the population and training data is to consider prior probabilities for each class j. If the training dataset was assumed to follow the same distribution as the population then these prior probabilities $(\pi_1, \pi_2, \ldots, \pi_J)$ would simply be estimated by the proportion of observations in each class. We now consider introducing alternate prior probabilities to the J classes. By considering the probability of a given observation in class j going to a node c, it is possible to see how the priors impact the tree fitting.

As we saw previously in Section 3.2 $\sum_{j=1}^{J} \frac{n_{cj}}{n_c} = 1$, where n_c is the number of observation at node c, and n_{cj} is the number of observations in class j and node c.

By taking the sum over all J classes, it can be seen that, by the law of total probability, the probability of a randomly chosen observation in the training set going to node c is

$$Pr(c) = \sum_{j=1}^{J} Pr(c, j) = \sum_{j=1}^{J} Pr(c|j)Pr(j) = \sum_{j=1}^{J} \frac{n_{cj}}{n_j}\pi_j. \tag{4.1}$$

This shows how changing the prior of the distribution of the classes will result in a change in the way that the tree is produced. From this, the posterior probability is

$$Pr(j|c) = \frac{Pr(c|j)Pr(j)}{Pr(c)} = \frac{\frac{n_{cj}}{n_j}\pi_j}{\sum_{i=1}^{I} \frac{n_{ci}}{n_i}\pi_i}.$$

Trees with priors simply is done by changing the proportions of classes in the training data, then constructing the tree as it is described in Section 3.1.

Example 5

To see the impact of changing `prior` parameter on `rpart` trees, we will consider fitting a `rpart` tree on a real dataset, the `iris` dataset (32) with different prior values, where the `iris` data has 150 observations, 4 explanatory variables `Sepal.Length`, `Sepal.Width`, `Petal.Length` and `Petal.Width`, and a response variable that has 3 classes `setosa`, `versicolor` and `virginica`. Observations are equally divided between the three classes, i.e. the proportions of the data classes are $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$. Gini is the default impurity function in `rpart` and Gini with weights is explained in Subsection 3.2.2, and we will see in Subsection 4.1.3 that weights can be made to resemble priors.

Figure 4.1 shows the fitted tree without changing prior in `rpart.control` paramter in `rpart` command as in Figure 4.1(a). By selecting a correctly classified subset of the given `iris` data which are drawn as in

Figure 4.1(b), the boundaries of the regions coincide with the split cut points in Figure 4.1(a) and these observations are classified according to the class that has the largest posterior probability.

Posterior probabilities of these observations are given in Table 4.1 and they are computed as follows from Equation (2.4) for the three leaves:

$$Pr(\text{setosa}|C_1) = \frac{1 \times \frac{1}{3}}{1 \times \frac{1}{3} + 0 \times \frac{1}{3} + 0 \times \frac{1}{3}} = 1, \ldots$$

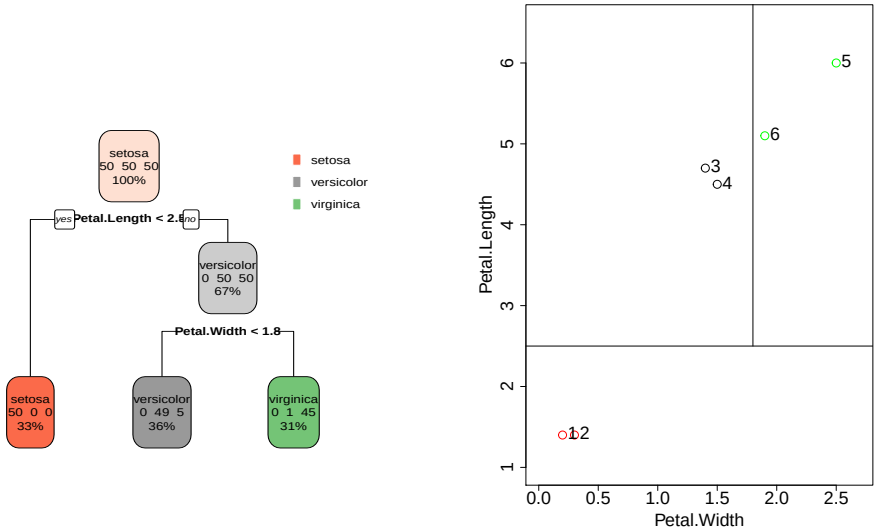
$$Pr(\text{versicolor}|C_2) = \frac{\frac{49}{54} \times \frac{1}{3}}{0 \times \frac{1}{3} + \frac{49}{54} \times \frac{1}{3} + \frac{5}{54} \times \frac{1}{3}} = \frac{49}{54}, \ldots$$

$$Pr(\text{virginica}|C_3) = \frac{\frac{45}{46} \times \frac{1}{3}}{0 \times \frac{1}{3} + \frac{45}{46} \times \frac{1}{3} + \frac{1}{46} \times \frac{1}{3}} = \frac{45}{46},$$

where C_j is the class j and $j \in \{\text{setosa, versicolor, virginica}\}$.

	setosa	versicolor	virginica
1	1	0	0
2	1	0	0
3	0	0.90740741	0.09259259
4	0	0.90740741	0.09259259
5	0	0.02173913	0.97826087
6	0	0.02173913	0.97826087

Table 4.1: The posterior probability for a subset of correctly classified observations from `iris` data as they appear in Figure 4.1(b). Each observation is allocated based on the class which has the largest probability and the dominating class has the largest probability in case of a not pure node.



(a) A fitted tree on `iris` data with priors $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$.

(b) Boundaries of the regions.

Figure 4.1: First plot shows a tree fitted on `iris` dataset and the second plot shows a subset of correctly classified observations from `iris` data. It is clear that classes' region limits in Figure 4.1(b) satisfy with split cut points in Figure 4.1(a). All observations which have `Petal.Length` < 2.5 go to `setosa` terminal node (red colour) and the rest of them if they have `Petal.Width` < 1.8 go to `versicolor` terminal node (black colour) and if not go to `virginica` (green colour).

Now, we change the priors from $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ to $(0.1, 0.1, 0.8)$, which means we give the classes `setosa`, `versicolor` and `virginica` prior probabilities 0.1, 0.1 and 0.8, respectively. Giving one class a very large prior should be indicating the high importance and dominance of that class.

We can notice there is a difference between the tree in Figure 4.1(a) and the tree in Figure 4.2 in terms of how each leaf is allocated to a class. From Figure 4.2, we can see that we have 4 terminal nodes. A node's colour indicates the purity of that node, the darker the colour, the purer the node. For example, the node that has light red shade, the second level, this node has (50 `setosa` observations, 44 `versicolor` observations and 1 `virginica` observation). The node is allocated to a class by multiplying the number of observations by the prior values and from this the winning class is `setosa`, the same way is applied to the rest of the nodes.

As we expected that the class with the largest prior probability, i.e. `virginica`, is dominating. The light green node in Figure 4.2 has 7 observations, 6 of them are `versicolor` and 1 is `virginica` and yet the class label is `virginica`.

Finally, the percentage in each node represents the percentage of the total observations at that node. Table 4.2 shows the probabilities of observations are mentioned in Figure 4.1(b). As we can see from this table that changing the prior caused in providing more accurate probabilities. Observations 3 and 4 probabilities changed from 0.90740741 to 1 and observations 5 and 6 changed from 0.97826087 to 0.9849246.

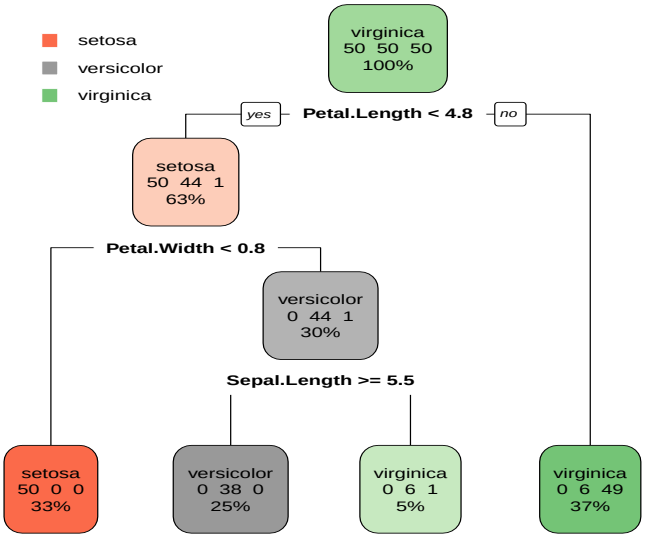


Figure 4.2: A classification tree on `iris` data with $(0.1, 0.1, 0.8)$ prior probabilities. Changing the classes frequencies (priors) can result to a different tree from the tree without changing these frequencies as in Figure 4.1(a).

	setosa	versicolor	virginica
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	0	0.01507538	0.9849246
6	0	0.01507538	0.9849246

Table 4.2: The posterior probability for the same subset, as in Figure 4.1(b), of correctly classified observations from `iris` data. Each observation is allocated based on the class which has the larger probability and the dominating class has the larger probability in case of a not pure node.

4.1.2 Trees with Weights

The second method of overcoming unbalanced data relies on changing the contribution of the observations. This method changes the weights for certain "more important" observations. If the weights added to each observation are equal then this is of little interest because observations have an equal impact on the final decision. If however, there are observations, which the experimenter believes, are of more importance "according to the precision of the data measurements" (Page 159, (25)), then a larger weight can increase the importance of these observations. Due to increasing the weight of these observations, they have higher impact on the the tree growing process than the observations with less weight, i.e. the tree will be biased towards observations similar to those whose weights are increased. Another example for increasing the weights of observation is to increase the weight of misclassified observations which will cause the importance of these observations to be increased, thus reducing the likelihood of incorrect classification.

Here, when a tree is being constructed, instead of considering the proportion of each class in each node, as in Subsection 4.1.1, a weighted sum is calculated. Consider that there are n_c observations at node c, with J different classes, each observation (x_i, y_i) has weight w_i, then the weighted sum of observations at node c in class j is

$$\gamma_{cj}(x_i) = \frac{\sum_{i=1}^{n_c} w_i \ I(y_i = j)}{\sum_{i=1}^{n_c} w_i}. \tag{4.2}$$

We can see from Equation (4.2) that the case where there is equal weight is equivalent to simply giving each observation a weight $w_i = 1$,

$$\gamma_{cj}(x_i) = \frac{1}{n_c} \sum_{i=1}^{n_c} I(y_i = j) = \frac{n_{cj}}{n_c}. \tag{4.3}$$

Finally, replicating the data has the same effect of giving observations integer weights.

Example 6

Changing the weight of some observations from `iris` data and applying `rpart` trees can result in a different decision. For example, we take one of the five misclassified observations in Figure 4.1(a) in the leaf that is allocated to `versicolor` class, which were allocated to the `versicolor` class instead of the `virginica` class. Then, by changing the weight allocated to a chosen observation (number 107) from 1 to 2, its contribution is also changed as well as the split point. Furthermore, this causes a change in the split point (changed from 1.8 to 1.6) and the observation becomes correctly classified as in Figure 4.3.

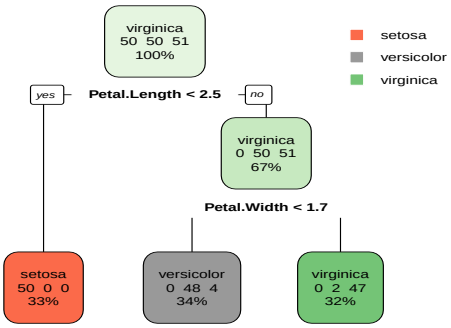


Figure 4.3: A weighted classification tree on `iris` data after changing the weights of one observation in the `virginica` class.

4.1.3 How to Make Weights Act Like Priors

We can have the same effect of priors on a tree by changing the weight of observations. In `rpart` function, we use `weight` parameter and apply w_{ij}, where w_{ij} is the weight of the i^{th} observation in the j^{th} class and n_j is the number of observations in that class. The weight of an observation in class j is equal to the prior of that class, i.e. $w_{ij} = \hat{\pi}_j$, where $\hat{\pi}_j$ is the actual or wanted prior of class j.

Example 7

We have `iris` data that has three classes and by using `weights` parameter in `rpart` and applying weights to observations as follows: $w_{i_{\texttt{setosa}}} = 0.1$, $w_{i_{\texttt{versicolor}}} = 0.1$ and $w_{i_{\texttt{virginica}}} = 0.8$. We can compare the decision tree accuracy to a decision tree that uses `prior` parameter as follows: $\hat{\pi}_{\texttt{setosa}} = 0.1$, $\hat{\pi}_{\texttt{versicolor}} = 0.1$ and $\hat{\pi}_{\texttt{virginica}} = 0.8$.

The number and percentage of observations in both trees can be seen from Figure 4.4. Figure 4.4(a) and Figure 4.4(b) are similar but the only difference is the number of observations in the two trees leaves. The number of observations in the tree in Figure 4.4(b) is multiplied by the weight of observations in each class. For example, if we take the `virginica` leaf, in the far right of the plot, in Figure 4.4(a), and multiply the number of observations by $0.1, 0.1, 0.8$, then we will have $0\times0.1 = 0$, $6\times0.1 = 0.6$ and $49\times0.8 = 39$ which are the same numbers in `virginica` leaf, in the far right of the plot, in Figure 4.4(b).

The subset of observations in Figure 4.1(b) has the same probability values by using this tree settings. The next equations show the same results as in Table 4.2 and they are computed as follows from Equation (2.4):

Observations 1 and 2:

$$Pr(\text{setosa}|C_1) = \frac{1 \times \frac{1}{10}}{1 \times \frac{1}{10} + 0 \times \frac{1}{10} + 0 \times \frac{8}{10}} = 1.$$

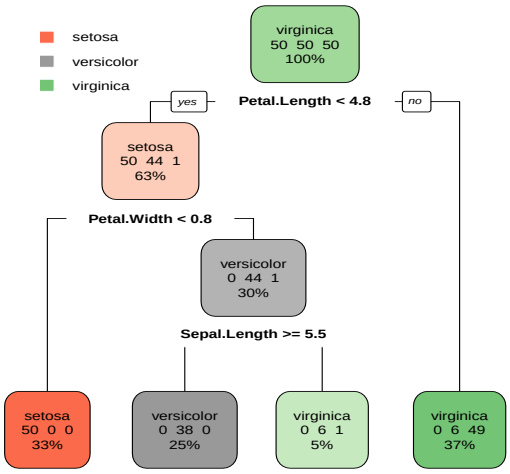
Observations 3 and 4:

$$Pr(\text{versicolor}|C_2) = \frac{1 \times \frac{1}{10}}{0 \times \frac{1}{10} + 1 \times \frac{1}{10} + 0 \times \frac{8}{10}} = 1.$$

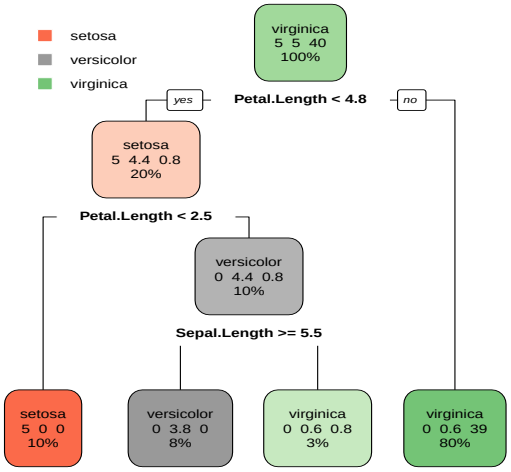
Observations 5 and 6:

$$Pr(\text{versicolor}|C_3) = \frac{\frac{6}{55} \times \frac{1}{10}}{0 \times \frac{1}{10} + \frac{6}{55} \times \frac{1}{10} + \frac{49}{55} \times \frac{8}{10}} = 0.01507538,$$

$$Pr(\text{virginica}|C_3) = \frac{\frac{49}{55} \times \frac{8}{10}}{0 \times \frac{1}{10} + \frac{6}{55} \times \frac{1}{10} + \frac{49}{55} \times \frac{8}{10}} = 0.9849246$$



(a) Tree with priors $(0.1, 0.1, 0.8)$.



(b) Tree with weights equivalent to priors $(0.1, 0.1, 0.8)$.

Figure 4.4: Decision trees used `iris` data once with priors and the other with weights equivalent to the same priors. The two trees are similar and the only difference is number of observations in 4.4(b) is multiplied by $0.1, 0.1, 0.8$. The number of observations in the root node in the first tree is $50, 50, 50$, while the number of observations in the root node in the second tree is $0.1 \times 50 = 5$, $0.1 \times 50 = 5$ and $0.8 \times 50 = 40$.

4.2 Investigating Schemes of Weighted Samples in Random Forests

In this section, we will be exploring whether changing the sampling mechanism from bagging, as was explained in Section 2.8, to giving observations weights based on a distribution has an impact on the distribution of the first split of each tree within the random forest or not.

We are interested in this question to test that changing observations' weights according to a distribution has any effect on changing the distribution of the first splitting variable and split point. The importance of first split variable and point lies on the fact that decision trees choose the variable and split point "cut-off point" which provide the most homogenous child nodes, i.e. the variable and split point that minimise the Gini index. Gini index, as explained in Subsection 3.2.2, is an impurity function to measure how impure a node is.

We will be comparing between bagging and weighted sampling and their effect on a tree but before this comparison, we will explain in the next subsection the resemblance between bagging and multinomial distribution in terms of the way they choose observations.

4.2.1 The Resemblance Between Bagging and Multinomial Distribution

In order to explain the resemblance between how bagging and multinomial distribution select samples from a data, we will suppose we have a sequence of learning sets $\{X_b\}, b = 1, 2, \ldots, B$, and each of these learning sets has n observations, drawn with replacement uniformly at random from the dataset. This is the bootstrap technique and it has been used in many ensemble methods like: random forests (6) and bagging predictors (5).

To explain the way bagging works, suppose one does an experiment by selecting n^* observations with replacement (notice that sometimes $n^* = n$) from a data set

that has n observations.

1. In sampling process, let

$$I_{iq} = \begin{cases} 1 & \text{if observation } x_i \text{ is selected from the } q^{th} \text{ draw,} \\ 0 & \text{otherwise} \end{cases},$$

   where $i = 1, \ldots, n$ and $q = 1, \ldots, n^*$ and this drawing procedure is by nature with replacement, so once an observation has been selected, it is still allowable for further selection.

2. $I_{iq} \in \{0, 1\}, i = 1, \ldots, n$ is a binary random variable distributed as a Bernoulli distribution i.e. $I_{iq} \sim \text{Bernoulli}(\frac{1}{n})$ and $\frac{1}{n}$ is the probability of choosing any observation from a data set which contains n observations.

3. We take n^* draws and the number of times the i^{th} observation is drawn is the number of successes in n^* independent draws with the same probability $\frac{1}{n}$ of success in each draw and the sum of these n^* draws L_i (sum of times observation i is drawn) has a binomial distribution i.e. $L_i \sim \text{Bin}(n^*, \frac{1}{n})$, where $L_i = \sum_{q=1}^{n^*} I_{iq}$,

   Note: L_i represents the number of draws of observation x_i such that $L_1 + \ldots + L_n = n^*$.

   There are $\binom{n^*}{L_1}$ ways to draw x_1 observation, $\binom{n^*-L_1}{L_2}$ ways to draw x_2 observation among the remaining $n^* - L_1$ draws, etc. Finally, there are only L_n spaces left for observation x_n. The total number of ways is

$$\frac{n^*!}{L_1! \ldots L_n!},$$

4. The draws are independent, so the chance of $L_1, L_2, \ldots, L_n$ is

$$\left(\frac{1}{n}\right)^{L_1} \ldots \left(\frac{1}{n}\right)^{L_n}.$$

5. Therefore, the chance that the n^* draws result in L_1 outcomes of observation

x_1, L_2 outcomes of observation $x_2, \ldots$ and L_n of observation x_n is

$$Pr(L_1 = l_1, L_2 = l_2, \ldots, L_n = l_n) = \frac{n^*!}{l_1! \ldots l_n!} \left(\frac{1}{n}\right)^{l_1} \ldots \left(\frac{1}{n}\right)^{l_n}$$

$$= \frac{n^*!}{l_1! \ldots l_n!} n^{-\sum_{i=1}^{n^*} l_i} = \frac{n^*!}{l_1! \ldots l_n!} n^{-n^*},$$

and this is a multinomial distribution.

4.2.2 Investigating the Distribution of the First Split

For a given "resampling scheme" we save the split variable and point when we split the root node, over B samples, this gives B pairs of (variable, point). We then can compare the distribution of these pairs across different weighting or sampling schemes.

To investigate this, we will carry out several experiments that are done by using the `rpart` function. We will generate n independent weights $w_i, i \in \{1, 2, \ldots, n\}$ based on a distribution that has a probability function $\Psi_W(w; \lambda) = Pr(W = w)$, where W is a random variable, and λ is the parameter of this distribution.

The parameter λ is calculated in a way to make the expected value of the occurrence of the training data observations around a specific value and we will explain how to calculate it later in Sub-subsection 4.2.2.1. Then, we use this set of discrete numbers $w \in \mathbb{Z}^+$ to replicate the training data, where for example, if $w_i = 0$, then observation x_i will be ignored and if the weight of the observation is $w_i = 2$, then observation x_i is duplicated twice. There is no upper limits on the weights are applied to this experiment but the expected sum of the weighted bootstrap is fixed. Replicating data is equivalent to associating integer weights to the data. For a given dataset $(x_i, y_i), i = 1, \ldots, n$ and by selecting the i^{th} observation w_i times with replacement, we will have a new dataset "weighted bootstrap sample" from the original training data. Following this, we grow a tree on this weighted bootstrap sample and collect the wanted information about the first split. The tree here considers all p variables of the training data unlike random

forests by Breiman (6) that selects a set of the variables.

The investigation experiment can be summarised in the following steps:
For$(b \leftarrow 1 \quad to \quad B)$ **do**

1. Generate n weights w_i
2. Select the i^{th} observation w_i times with replacement, and we will have a weighted bootstrap sample
3. Fit a tree on the weighted bootstrap sample
4. Save the first split variable index $l \in \{1, 2, \ldots, p\}$ and its corresponding split point

The distribution $\Psi_W(w; \lambda)$ is taken in this work as a Poisson distribution and geometric distribution in the weighting scheme, and a multinomial distribution corresponds to bagging. Poisson and geometric distributions are used here because their random variable W takes possible values belonging to $\mathbb{Z}^+$.

After applying this experiment by using one of the weighting scheme distributions (Poisson or geometric) and the bagging scheme (multinomial distribution), we will have a set of variables and split points from each scheme. We then take the mutual most common variables on both schemes and test the two hypotheses as in Sub-subsection 4.2.2.2.

4.2.2.1 How to calculate the parameter λ?

We have that $\Psi_W(w; \lambda)$ is the geometric distribution, λ is the success probability and w is the number of failures before the first success. To find λ value, we need to decide roughly the size of the weighted bootstrap sample, i.e. an expected value of the observation occurrence in the weighted bootstrap. For example, if we have $n = 1000$ and we need the weighted bootstrap sample size to be 50, then we solve this equation $n^* = n\,\frac{1-\lambda}{\lambda}$, where n^* is the size of the weighted bootstrap sample, and that is $50 = 1000\,\frac{1-\lambda}{\lambda}$, which gives $\lambda = \frac{n}{n+n^*} = \frac{1000}{1050}$.

In case of the Poisson distribution, λ can be found from this equation $n^* = n\lambda$, which gives $\lambda = \frac{n^*}{n} = 0.05$ for $n^* = 50$.

Finally, in the case of multinomial distribution, we can find λ by solving this equation $n^* = n\lambda$, where λ here is the event probabilities.

4.2.2.2 Hypotheses

1. The null hypothesis is "the choice of $\Psi_W(w;\lambda)$ does not affect the distribution of the first split variable".
   This hypothesis is tested using Fisher's exact test (36) on the contingency table of the variables from the two schemes to test the first split variable's marginal homogeneity.

2. The null hypothesis is "the choice of $\Psi_W(w;\lambda)$ does not affect the distribution of the first split point".
   Anderson-Darling test (37) is applied to test this hypothesis. Here, we compare between the distribution of the split points in the mutual most common variable in both scheme then the mutual second most common variable. If the variable V_l is the mutual most common variable in both schemes, we test the corresponding split points. Then, repeat the test on the second common variable, let say it is V_r.

4.3 The Results of Investigating Weighted Schemes

We will test the previous hypotheses on simulation datasets as in the following subsections and we consider two types of simulated data:

4.3.1 Multiple Explanatory Variables

We have here a dataset which has $n = 3000$, a response variable and 10 explanatory variables. The response variable has three classes C_1, C_2 and C_3, and they

are equally likely. The mean μ_1 has a difference of a little more than 4 between its values, the mean μ_2 has a difference of less than 2 between its values, and the difference between μ_3 values is 5.

Each population is taken from $N(\mu, \sigma^2 I)$, where:
in class 1,
$\mu_1^T = [1 \quad 5.30 \quad 9.70 \quad 14 \quad 18.30 \quad 22.70 \quad 27 \quad 31.30 \quad 35.70 \quad 40]$,$\sigma_1 = 1$,
class 2,
$\mu_2^T = [30 \quad 31.70 \quad 33.30 \quad 35 \quad 36.70 \quad 38.30 \quad 40 \quad 41.70 \quad 43 \quad 45]$,$\sigma_2 = 2$
and class 3,
$\mu_3^T = [35 \quad 40 \quad 45 \quad 50 \quad 55 \quad 60 \quad 65 \quad 70 \quad 75 \quad 80]$, $\sigma_3 = 3$.

We run the investigation experiment as in Page 71 $B = 1000$ times on the multivariable dataset 1. Then, collect the joint most common variables from the weighted and the bagging schemes for different n^*. We then test the contingency table for a pair of first variables from Poisson distribution using weight and bagging and another pair of geometric distribution using weight and bagging. We aim here to explore the difference which is made by changing the way we sample the bootstrap samples.

Table 4.3 shows P-values from testing the first hypothesis on these two schemes with different weighted bootstrap sample sizes. It clearly can be seen from the table that Poisson P-values decrease when n^* values increase until $n^* = \frac{n}{10}$ which gives very large value of P-value 0.82. P-values of geometric distribution using weighting also decrease by increasing n^* values until they reach P-value =0.0009 at $n^* = n$.

From these results, the first null hypothesis is accepted in case of Poisson distribution and the choice of $\Psi_W(w; \lambda)$ does not affect the distribution of the first split variable, however, in case of geometric distribution this null hypothesis is rejected.

We will test the points of the first two most common split variables. The two most common variables are $V2$ and $V3$ in both weighting schemes. Table 4.4 shows P-values of testing the second hypothesis. This table has some P-values less than 0.05 only for the geometric distribution.

n^*	Poisson distribution		Geometric distribution	
	λ	P-value	λ	P-value
$\frac{n}{50}$	$\frac{\frac{n}{50}}{n} = \frac{1}{50}$	0.12	$\frac{n}{n+\frac{n}{50}}$	0.67
$\frac{n}{40}$	$\frac{\frac{n}{40}}{n} = \frac{1}{40}$	0.09	$\frac{n}{n+\frac{n}{40}}$	0.20
$\frac{n}{30}$	$\frac{\frac{n}{30}}{n} = \frac{1}{30}$	0.07	$\frac{n}{n+\frac{n}{30}}$	0.18
$\frac{n}{20}$	$\frac{\frac{n}{20}}{n} = \frac{1}{20}$	0.06	$\frac{n}{n+\frac{n}{20}}$	0.16
$\frac{n}{10}$	$\frac{\frac{n}{10}}{n} = \frac{1}{10}$	0.82	$\frac{n}{n+\frac{n}{10}}$	0.09
$\frac{n}{5}$	$\frac{\frac{n}{5}}{n} = \frac{1}{5}$	0.45	$\frac{n}{n+\frac{n}{5}}$	0.009
$\frac{n}{2}$	$\frac{\frac{n}{2}}{n} = \frac{1}{2}$	0.73	$\frac{n}{n+\frac{n}{2}}$	0.004
n	$\frac{n}{n} = 1$	0.80	$\frac{n}{n+n} = \frac{1}{2}$	0.0009

Table 4.3: P-values of testing the first split variable by using Fisher test from different weighted bootstrap sample size n^* for multivariable dataset 1.

We generate a second dataset that has $n = 1000$, a response variable and 9 explanatory variables. The response variable has two classes C_1, C_2, and they are equally likely. Each population is taken from $N(\mu, \sigma^2 I)$, where:
in class 1,
$\mu_1^T = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$, $\sigma_1 = 1$,
class 2,
$\mu_2^T = [1 \quad 1 \quad 1 \quad 1.5 \quad 1.5 \quad 1.5 \quad 2 \quad 2 \quad 2]$, $\sigma_2 = 1$.

We run the investigation experiment $B = 1000$ times on the multivariable dataset 2. Table 4.5 shows P-values from testing the first hypothesis and it can be seen from the table that the largest Poisson P-value is at $n^* = \frac{n}{5}$ but there is no significant values. P-values of geometric distribution using weights are all less than 0.05. This results match the previous results of multivariable dataset 1.

We will test the points of the first two most common split variables. Table 4.6 shows P-values of testing the second hypothesis. This table has some P-values less than 0.05 only for the geometric distribution. Because of this, we will test the second hypothesis on one-explanatory-variable datasets as in the next subsection to focus more on this hypothesis.

n^*	Poisson distribution			Geometric distribution		
	λ	variable	P-value	λ	variable	P-value
$\frac{n}{50}$	$\frac{1}{50}$	$V2$	0.21	$\frac{n}{n+\frac{n}{50}}$	$V2$	0.04
		$V3$	0.85		$V3$	0.95
$\frac{n}{40}$	$\frac{1}{40}$	$V2$	0.45	$\frac{n}{n+\frac{n}{40}}$	$V2$	0.12
		$V3$	0.55		$V3$	0.85
$\frac{n}{30}$	$\frac{1}{30}$	$V2$	0.20	$\frac{n}{n+\frac{n}{30}}$	$V2$	0.03
		$V3$	0.80		$V3$	0.04
$\frac{n}{20}$	$\frac{1}{20}$	$V2$	0.80	$\frac{n}{n+\frac{n}{20}}$	$V2$	0.60
		$V3$	0.20		$V3$	0.50
$\frac{n}{10}$	$\frac{1}{10}$	$V2$	0.30	$\frac{n}{n+\frac{n}{10}}$	$V2$	0.10
		$V3$	0.90		$V3$	0.30
$\frac{n}{5}$	$\frac{1}{5}$	$V2$	0.70	$\frac{n}{n+\frac{n}{5}}$	$V2$	0.035
		$V3$	0.35		$V3$	0.02
$\frac{n}{2}$	$\frac{1}{2}$	$V2$	0.90	$\frac{n}{n+\frac{n}{2}}$	$V2$	0.03
		$V3$	0.55		$V3$	0.10
n	$\frac{n}{n}=1$	$V2$	0.20	$\frac{n}{n+n}=\frac{1}{2}$	$V2$	0.023
		$V3$	0.60		$V3$	0.80

Table 4.4: P-values of testing the points of the first and second split variables by using Anderson-Darling test from different weighted bootstrap sample size n^* for multivariable dataset 1.

n^*	Poisson distribution		Geometric distribution	
	λ	P-value	λ	P-value
$\frac{n}{50}$	$\frac{\frac{n}{50}}{n} = \frac{1}{50}$	0.14	$\frac{n}{n+\frac{n}{50}}$	0.02
$\frac{n}{40}$	$\frac{\frac{n}{40}}{n} = \frac{1}{40}$	0.09	$\frac{n}{n+\frac{n}{40}}$	0.001
$\frac{n}{30}$	$\frac{\frac{n}{30}}{n} = \frac{1}{30}$	0.11	$\frac{n}{n+\frac{n}{30}}$	0.017
$\frac{n}{20}$	$\frac{\frac{n}{20}}{n} = \frac{1}{20}$	0.17	$\frac{n}{n+\frac{n}{20}}$	0.01
$\frac{n}{10}$	$\frac{\frac{n}{10}}{n} = \frac{1}{10}$	0.12	$\frac{n}{n+\frac{n}{10}}$	0.005
$\frac{n}{5}$	$\frac{\frac{n}{5}}{n} = \frac{1}{5}$	0.62	$\frac{n}{n+\frac{n}{5}}$	0.01
$\frac{n}{2}$	$\frac{\frac{n}{2}}{n} = \frac{1}{2}$	0.16	$\frac{n}{n+\frac{n}{2}}$	0.004
n	$\frac{n}{n} = 1$	0.06	$\frac{n}{n+n} = \frac{1}{2}$	0.0009

Table 4.5: P-values of testing the first split variable by using Fisher test from different weighted bootstrap sample size n^* for multivariable dataset 2.

4.3.2 One Explanatory Variable

We have in Table 4.7 five simulated models, and each of these models have one explanatory variable and a response variable. The response variable has two classes C_1, C_2 and observations are equally likely divided between the two classes. The number of observations is $n = 3000$ and the densities of the explanatory variable of these models are shown in Figure 4.5. All the densities are normally distributed apart from the first class of model 3 which has a beta distribution. Some of the normal distributions of the classes are bimodal and the others are unimodal. These simulations are chosen to have a variety of overlapping between the two classes, and this variety differs from a simple overlapping as model 1 to complicated overlapping as model 4. The main reason of choosing simulation settings this way is to have a more reliable results from the simulation studies. The maximal model accuracies, i.e. Bayes classification errors, of the five models are shown in Table 4.8. We will explain in the next example how to find the maximal accuracy of a model.

n^*	Poisson distribution			Geometric distribution		
	λ	variable	P-value	λ	variable	P-value
$\frac{n}{50}$	$\frac{1}{50}$	$V7$	0.60	$\frac{n}{n+\frac{n}{50}}$	$V7$	0.70
		$V8$	0.65		$V8$	0.35
$\frac{n}{40}$	$\frac{1}{40}$	$V7$	0.36	$\frac{n}{n+\frac{n}{40}}$	$V7$	0.09
		$V9$	0.53		$V9$	0.10
$\frac{n}{30}$	$\frac{1}{30}$	$V7$	0.95	$\frac{n}{n+\frac{n}{30}}$	$V7$	0.70
		$V9$	0.35		$V9$	0.65
$\frac{n}{20}$	$\frac{1}{20}$	$V7$	0.85	$\frac{n}{n+\frac{n}{20}}$	$V7$	0.02
		$V9$	0.50		$V9$	0.09
$\frac{n}{10}$	$\frac{1}{10}$	$V7$	0.90	$\frac{n}{n+\frac{n}{10}}$	$V7$	0.15
		$V9$	0.60		$V9$	0.85
$\frac{n}{5}$	$\frac{1}{5}$	$V7$	0.90	$\frac{n}{n+\frac{n}{5}}$	$V7$	0.028
		$V9$	0.50		$V9$	0.65
$\frac{n}{2}$	$\frac{1}{2}$	$V7$	0.70	$\frac{n}{n+\frac{n}{2}}$	$V7$	0.01
		$V9$	0.56		$V9$	0.15
n	$\frac{n}{n} = 1$	$V7$	0.80	$\frac{n}{n+n} = \frac{1}{2}$	$V7$	0.023
		$V9$	0.65		$V9$	0.007

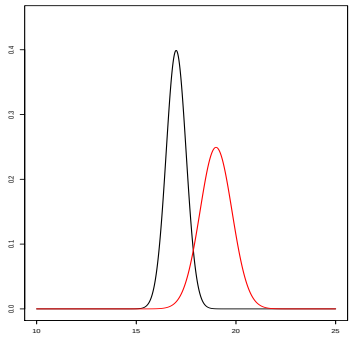
Table 4.6: P-values of testing the points of the first and second split variables by using Anderson-Darling test from different weighted bootstrap sample size n^* for multivariable dataset 2.

Model	C_1	C_2
1	$N(17, 0.5^2)$	$N(19, 0.8^2)$
2	$\begin{cases} N(5, 1.8^2), Pr = 0.45 \\ N(20, 0.6^2) \end{cases}$	$N(13, 4^2)$
3	$B(0.2, 0.2^2)$	$N(0.5, 0.2^2)$
4	$\begin{cases} N(4, 0.4^2), Pr = 0.30 \\ N(11, 1^2) \end{cases}$	$\begin{cases} N(8, 2^2), Pr = 0.65 \\ N(15, 1^2) \end{cases}$
5	$\begin{cases} N(5, 1.3^2), Pr = 0.35 \\ N(11, 1.4^2) \end{cases}$	$N(8, 1.6^2)$

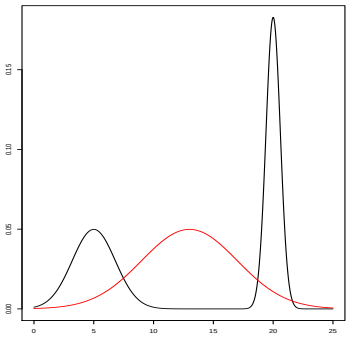
Table 4.7: Densities of the explanatory variable for five models.

Model	1	2	3	4	5
Accuracy	94.21%	92.87%	83.51%	87.37%	78.51%

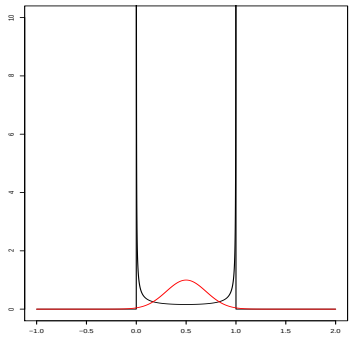
Table 4.8: Percentages of the maximal model accuracies for the five models.



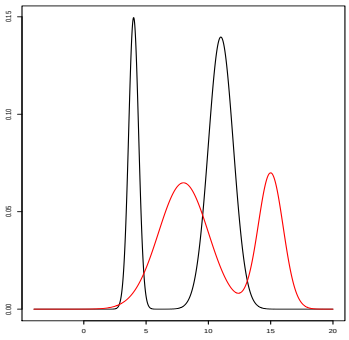
(a) Model 1



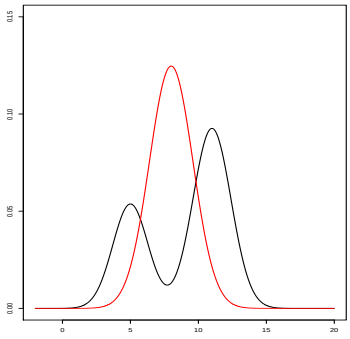
(b) Model 2



(c) Model 3



(d) Model 4



(e) Model 5

Figure 4.5: Densities of the explanatory variable of the five models according to their classes. Red and black colours indicate class 1 and class 2 of the response variable.

Example 8

Here, to find the maximal model accuracy, we use Equation (2.14).

$$R^*(d^*) = 1 - \sum_{j=1}^{J} \int_{\Omega_j} \max_{j'} f_{j'}(x)\pi_{j'} dx.$$

If we take model 1 in which its response variable has two classes and their densities are $f_1(x) \sim N(17, 0.5^2)$ and $f_2(x) \sim N(19, 0.8^2)$. From Figure 4.6, we have two areas greater and less than the interaction point 17.86. These areas are: on the left of the interaction point $f_2(x)$ and on the right $f_1(x)$, however, the yellow shaded area is part of $f_2(x)$ but the maximum value in the left region is $f_1(x)$ and the purple shaded area is part of $f_1(x)$ but the maximum value in the right area is $f_2(x)$ and by applying the above equation, we have

$$1 - \pi\left(\int_{-\infty}^{17.86} f_1(x) + \int_{17.86}^{\infty} f_2(x)\right) dx = 0.9421,$$

where $\pi = 0.5$ because observations are divided equally between the two classes.

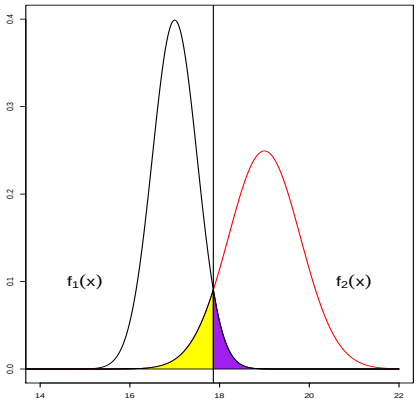


Figure 4.6: Model 1 density with the overlapping areas. The yellow shaded area is part of $f_2(x)$ but the maximum value in the left region is $f_1(x)$ and the purple shaded area is part of $f_1(x)$ but the maximum value in the right area is $f_2(x)$.

This experiment is run $B = 1000$ times on each weighting scheme, then the split points from the alternative weighting scheme and bagging is tested. These two sets of split points are tested by using Anderson-Darling test to see if they have the same underlying distribution. Table 4.9 shows the P-values of testing the split points of the five models above. We can see that there are some values under the threshold 0.05 but results in general do not show significant differences between bagging weight and the alternative distribution of weights. The significant values appear at small values of n^* and more frequent with geometric distribution than Poisson distribution of weights.

Model	n^*	Poisson distribution	Geometric distribution
1	$\frac{n}{50}$	0.22	0.35
	$\frac{n}{40}$	0.90	0.65
	$\frac{n}{30}$	0.05	0.005
	$\frac{n}{20}$	0.10	0.11
	$\frac{n}{10}$	0.90	0.08
2	$\frac{n}{50}$	0.04	0.08
	$\frac{n}{40}$	0.56	0.86
	$\frac{n}{30}$	0.58	0.33
	$\frac{n}{20}$	0.43	0.04
	$\frac{n}{10}$	0.54	0.07
3	$\frac{n}{50}$	0.45	0.04
	$\frac{n}{40}$	0.85	0.45
	$\frac{n}{30}$	0.94	0.81
	$\frac{n}{20}$	0.74	0.45
	$\frac{n}{10}$	0.10	0.80
4	$\frac{n}{50}$	0.60	0.01
	$\frac{n}{40}$	0.80	0.36
	$\frac{n}{30}$	0.37	0.60
	$\frac{n}{20}$	0.45	0.41
	$\frac{n}{10}$	0.49	0.34
5	$\frac{n}{50}$	0.60	0.02
	$\frac{n}{40}$	0.70	0.38
	$\frac{n}{30}$	0.50	0.37
	$\frac{n}{20}$	0.40	0.43
	$\frac{n}{10}$	0.80	0.58

Table 4.9: P-values of testing the split points by using Anderson-Darling test from different weighted bootstrap sample size n^* for the five models. Values in red are less than the threshold 0.05.

4.3.3 Real Datasets

We will apply the alternative weighting scheme and bagging to three real datasets to verify the previous results we found in simulations. The datasets are USPS dataset, as mentioned in Example 3 in Chapter 3, and the other two datasets are from the UCI Machine Learning Repository[1]. Table 4.10 presents a description of the datasets and the numbers of their observations and variables.

Data	p	n	Description
USPS	257	7290	Handwritten digits obtained from envelopes by the U.S. Postal Service
Banknote [2]	4	1372	Data was obtained from images that were taken to evaluate the authentication procedure of banknotes
Survival[3]	3	306	Data set has cases from a study conducted on the survival of patients who had undergone surgery for breast cancer

Table 4.10: Real datasets, number of variables and observations, and a short description of the dataset.

We apply bagging and the alternative weight schemes on the real data $B = 1000$ times with three different n^* sizes, which are a (third, quarter and fifth) of the data. Then, collect and test the first split variable and points as was explained previously.

Table 4.11 shows the P-values of the variables from the two weighting methods. USPS data shows P-values that are less than 0.05 with geometric distribution, while the P-values are not significant with Poisson distribution of weights. The weighting methods with Banknote dataset select one variable and we cannot test the first hypothesis here. Finally, testing the first hypothesis with Survival dataset does not show P-values smaller than the threshold 0.05 and that means changing

[1] http://archive.ics.uci.edu/ml
[2] https://archive.ics.uci.edu/ml/datasets/banknote+authentication
[3] https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival

$\Psi_W(w;\lambda)$ from multinomial to an alternative weighting scheme does not affect the first split variable distribution in this dataset.

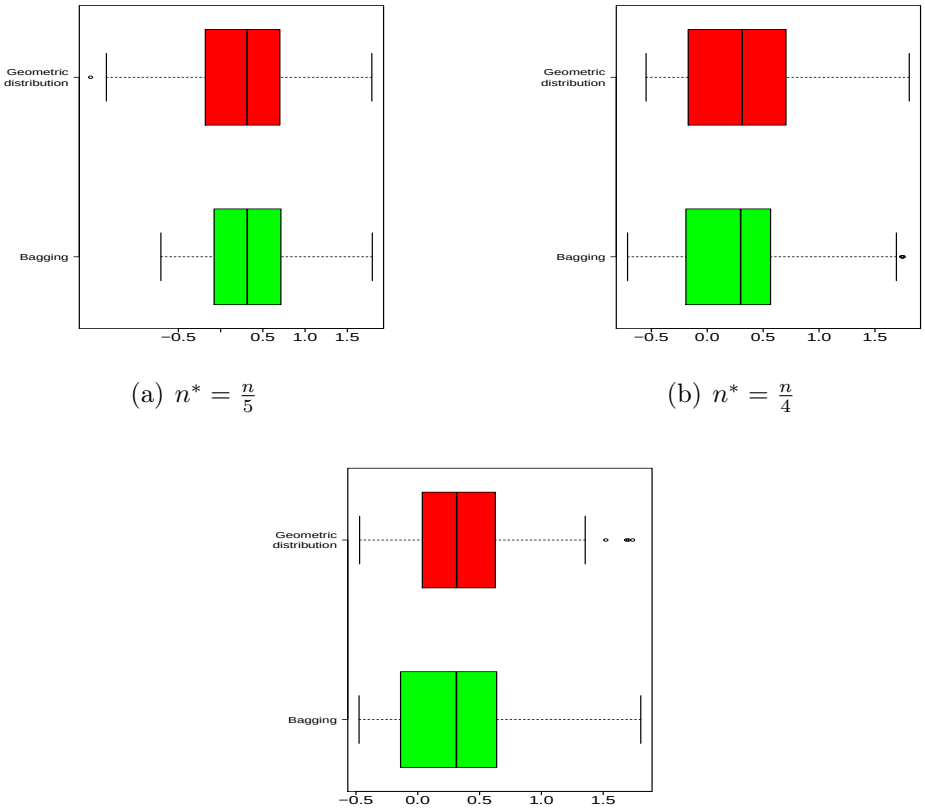
Table 4.12 shows the P-values of testing the split points of the common variables from both weighting methods. USPS dataset indicates significant values for the split points of one variable $V214$ for different n^* values with geometric distribution of weights. Banknote dataset shows significant values with all n^* values and geometric distribution of weights. This difference match the box plots in Figure 4.7. We can see in this figure that the two groups of split points differ in the minimum, maximum and the spread values. Finally, Survival dataset does not show any significant values with either distributions.

Dataset	n^*	Poisson dist.	Geometric dist.
USPS	$\frac{n}{5}$	0.48	0.012
	$\frac{n}{4}$	0.46	0.027
	$\frac{n}{3}$	0.37	0.0009
Banknote	$\frac{n}{5}$	The two weighting schemes select just one variable $V2$.	
	$\frac{n}{4}$		
	$\frac{n}{3}$		
Survival	$\frac{n}{5}$	0.19	0.31
	$\frac{n}{4}$	0.33	0.21
	$\frac{n}{3}$	0.87	0.07

Table 4.11: P-values of testing the first split variable for real datasets.

Dataset	n^*	Variable	Poisson dist.	Geometric dist.
USPS	$\frac{n}{5}$	$V154$	0.85	0.99
		$V170$	0.90	0.70
		$V197$	0.99	0.99
		$V205$	0.99	0.70
		$V214$	0.35	0.01
		$V221$	0.99	0.99
	$\frac{n}{4}$	$V154$	0.99	0.99
		$V170$	0.05	0.25
		$V197$	0.99	0.99
		$V205$	0.99	0.99
		$V214$	0.50	0.025
		$V221$	0.99	0.99
	$\frac{n}{3}$	$V154$	0.99	0.45
		$V170$	0.90	0.50
		$V214$	0.80	0.02
Banknote	$\frac{n}{5}$	$V2$	0.70	0.04
	$\frac{n}{4}$		0.15	0.03
	$\frac{n}{3}$		0.45	0.025
Survival	$\frac{n}{5}$	$V2$	0.45	0.90
		$V3$	0.85	0.90
		$V4$	0.90	0.25
	$\frac{n}{4}$	$V2$	0.40	0.85
		$V3$	0.20	0.35
		$V4$	0.60	0.15
	$\frac{n}{3}$	$V2$	0.90	0.90
		$V3$	0.15	0.50
		$V4$	0.35	0.20

Table 4.12: P-values of testing the first split points for real datasets.Values in red are less than the threshold 0.05.



(a) $n^* = \frac{n}{5}$

(b) $n^* = \frac{n}{4}$

(c) $n^* = \frac{n}{3}$

Figure 4.7: The first split points for Banknote Authentication dataset by using bagging and weighted sampling with different n^*. Box-plots in red represent the weight of geometric distribution and the green ones are bagging. The red and green box-plots are different in the minimum, maximum and the spread values.

4.4 Summary of the Chapter

From the experiments we ran, we find that changing the mechanism of sampling does not have an impact on the first split variables in the simulation with Poisson distribution but it does with the geometric distribution, also one of the real datasets, USPS, shows significant results for the first split variable with geometric distribution while the other real datasets did not show any statistically significant P-values with either Poisson or geometric distributions. Also, testing the first split points shows some significant results with some simulated models and real datasets with only geometric distribution of weights. This variation in the split points distribution may lead to enhance the overall performance of the random forest, which is a topic for further investigation.

Chapter 5

Forests of Stumps

Decision stumps are one of the simplest classification methods and this term was first used by Iba and Langley (38).

In this chapter, we will investigate the aggregation of decision stumps which is called a one-level decision tree. A decision stump is a tree has a root that is directly connected to leaves and it is also called a weak learner, which is a learner performs poorly and the accuracy of this learner is barely above 50% (38).

A stump uses a feature variable to split the training data and make the prediction on the testing data later on. For continuous feature variables, the most common approach is that a feature variable and a corresponding threshold feature value, i.e. split point, are selected to create a stump with two leaves for values below and above that split point.

Decision stumps perform poorly due to their simplicity. However, many numerical studies ((39), (7), (40), (41)) indicate that decision stumps perform more accurately within ensemble methods like bagging, which we will discuss in this chapter, or boosting, as in Chapter 7. We will investigate two approaches to create a forest of stumps for classification.

The first method is bagging with stumps where changing the bootstrap sample size is investigated. This method works in such a way that a stump fitted on

bootstraps of different sample sizes and the decisions of these stumps are combined by using one of the aggregating methods: majority vote (MV) or weighted vote (WV) as mentioned in Section 3.6.

The second method is Gini stumps and it has two sub-methods: Gini-sampled stumps and Gini-midpoints stumps. The way Gini stumps is constructed is instead of the ordinary way of constructing trees, as described in Section 3.1, the data is divided by sampling split points with probability proportional to Gini gain. The technique of this method is to sample split points according to the Gini index values, split the training dataset into two subsets greater and smaller than a split point by using this set of split points, and then combine the results by using the aggregation methods mentioned previously.

These two schemes will be described in more detail in Section 5.1. Then, we will apply the two methods on simulations and measure the accuracy and compare it to the maximal accuracy as in Section 5.2 and Section 5.3. A comparison will be carried out in Section 5.4 to compare between the two methods from two perspectives, the consumed time and the distribution of the split points from the two methods. Finally, we will summarise some of the results and test some developments of these two methods, for example bagging with two-level trees and Gini-sampled two-level trees as in Section 5.5.

5.1 Investigating Schemes of Forests of Stumps

We will investigate two main different methods to build an ensemble of decision stumps. The first scheme is bagging with stumps that is basically combining stumps with bagging technique and it will be described in more detail in Subsection 5.1.1. The second scheme is Gini stumps that is a new method to build a forest of stumps. This method will be explained in Subsection 5.1.2.

5.1.1 Bagging with Stumps

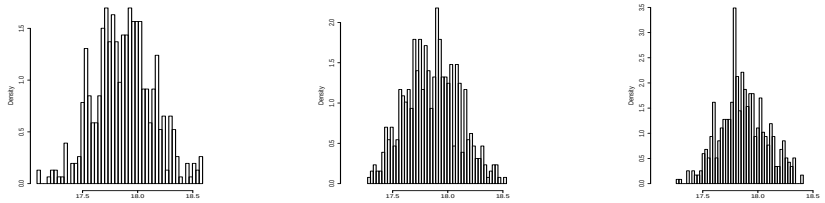
The main idea of this method is fitting a decision stump on a bootstrap of size n^* drawn from the training data by using `rpart` command, making a prediction on the testing data by using this model and finally applying aggregating methods to the stumps. We consider different n^* here to test whether the bootstrap size has any effect on the performance of the method. Split points in this method are determined, such that the split point maximises the Gini gain in each bootstrap sample. Also, the way the distribution of each class in each leaf is determined by checking bootstrap sample from training data. Then, stumps are aggregated by one of the aggregation methods that are mentioned in Section 3.6. The methodology of this ensemble, for one explanatory variable, can be summarised in Algorithm 2.

Algorithm 2 Bagging with Stumps

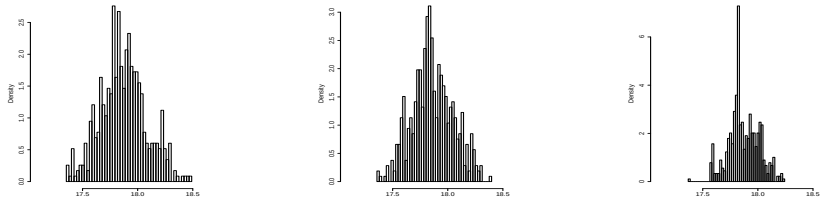
1. **For** ($b \leftarrow 1 \quad to \quad B$)
   (a) Draw n^* observations with replacement from the training data
   (b) Fit a decision stump on the sampled data
2. Apply one of the aggregation methods that are mentioned in Section 3.6
3. Predict on the testing data

Note that: using the following settings with `rpart` command: `data= a bootstrap of size` n^*, `method = ''class''` and `rpart.control` parameters used here are: `rpart.control(maxdepth =1, minsplit=1, cp = 0)`. This setting is needed in this way because the maximum depth is `maxdepth =1` to provide a stump, minimum split is `minsplit=1` to split the data even in case when one of the leaves has just one observation, complexity parameter is `cp = 0` to overcome the problem of having just a root and the method used is `method = ''class''` to fit a classification tree.

Running this algorithm with the two aggregation methods on simulations as described in Subsection 4.3.2 and bootstrap samples of different sizes n^* shows that the variation of split points decreases as n^* increases as in Figures from 5.1 to 5.5.

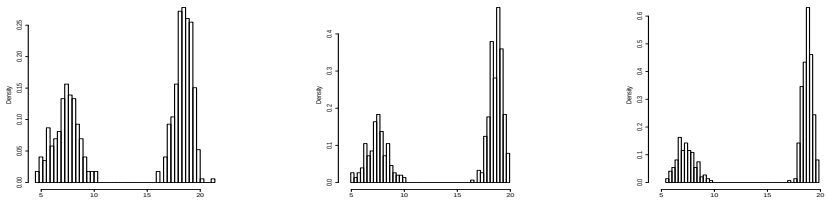


(a) $n^* = 20$ (b) $n^* = 40$ (c) $n^* = 60$

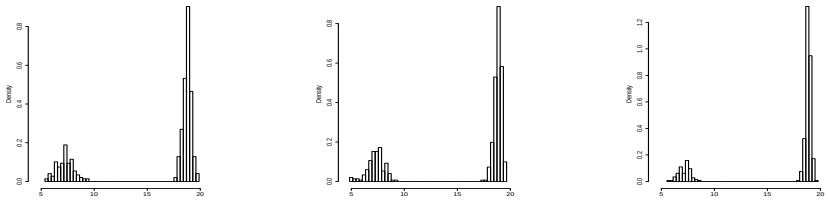


(d) $n^* = 80$ (e) $n^* = 100$ (f) $n^* = 200$

Figure 5.1: Split points of 500 stumps for bootstrap sample size n^* from model 1 where the size of training data is $n = 1000$.

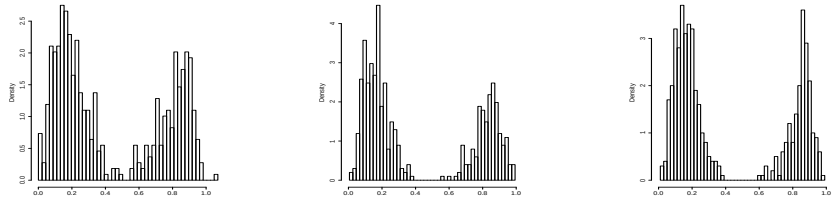


(a) $n^* = 20$ (b) $n^* = 40$ (c) $n^* = 60$

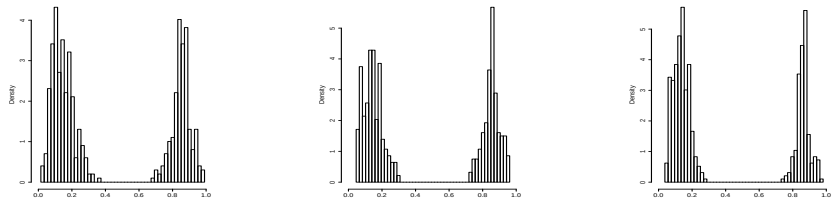


(d) $n^* = 80$ (e) $n^* = 100$ (f) $n^* = 200$

Figure 5.2: Split points of 500 stumps for bootstrap sample size n^* from model 2 where the size of training data is $n = 1000$.

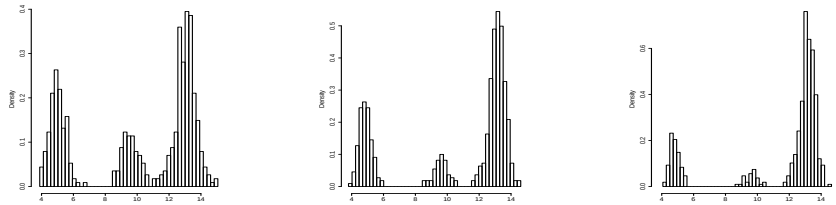


(a) $n^* = 20$ (b) $n^* = 40$ (c) $n^* = 60$

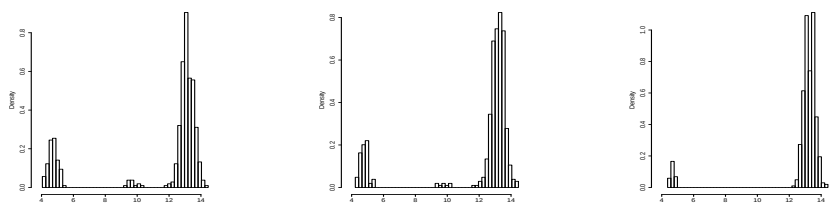


(d) $n^* = 80$ (e) $n^* = 100$ (f) $n^* = 200$

Figure 5.3: Split points of 500 stumps for bootstrap sample size n^* from model 3 where the size of training data is $n = 1000$.

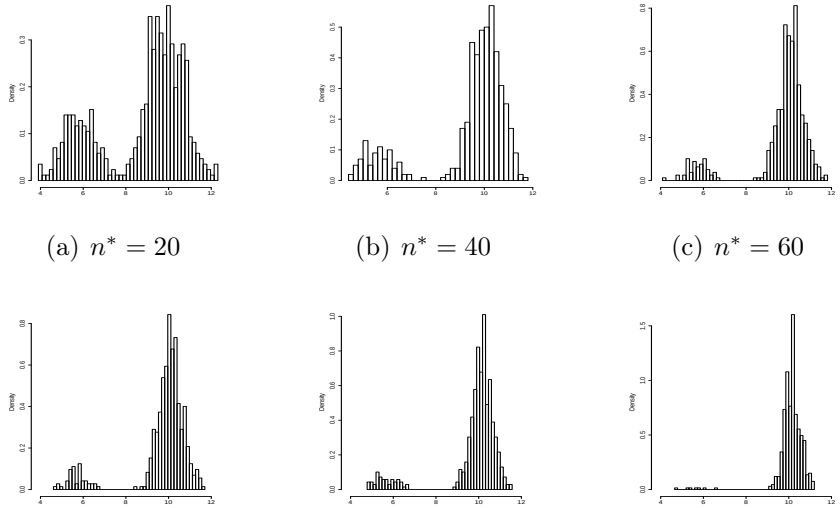


(a) $n^* = 20$ (b) $n^* = 40$ (c) $n^* = 60$



(d) $n^* = 80$ (e) $n^* = 100$ (f) $n^* = 200$

Figure 5.4: Split points of 500 stumps for bootstrap sample size n^* from model 4 where the size of training data is $n = 1000$.



(a) $n^* = 20$ (b) $n^* = 40$ (c) $n^* = 60$

(d) $n^* = 80$ (e) $n^* = 100$ (f) $n^* = 200$

Figure 5.5: Split points of 500 stumps for bootstrap sample size n^* from model 5 where the size of training data is $n = 1000$.

5.1.2 Gini Stump Methods

We will explain here two methods of Gini stumps: Gini-sampled stumps as in Sub-subsection 5.1.2.1 and Gini-midpoints stumps as in Sub-subsection 5.1.2.2. Then, we will compare between the two ways of generating the split points as in Sub-subsection 5.1.2.3.

5.1.2.1 Gini-sampled Stumps Method

Here we take a random sample of split points derived from Gini indices to build an ensemble of stumps. It is known that, fitting a tree, in general, to split the data into two groups needs to calculate the Gini indices for all possible variables and corresponding split points, chooses the highest Gini gain and then splits the data by using that variable and corresponding split point.

In this method, we find the mid-points of an explanatory variable as follows: let $x_1, \ldots, x_n$ be the data-points of variable $p = 1$ and $x_{(1)}, \ldots, x_{(q)}$ be the unique

values in increasing order, here q depends on the number of the unique elements of the variable. Then, calculate the change in Gini g of the ordered midpoints by using the formula as in Equation (3.5). After this, we normalise Gini vector by using Riemann integral (42) as in Step 2 in Algorithm 3. We use the mid-points here as the split points because Gini index is constant on the interval between two split points.

We can sample from this *pdf* directly or generate random split points by applying inverse transform method (Page 18, (43)). Then, we use the split points to divide the training data into two subsets and finally predict on the testing data.

We also consider raising Gini index values g to a power κ such that $\kappa \in [0, \infty)$ before Step 2 in Algorithm 3 to have more flexibility and test whether decreasing the variation of the randomly generated split points will have any effect on making this method perform differently. Note, $\kappa = 0$ gives uniformly distributed splits and $\kappa \to \infty$ will always return the optimal split point for that training data.

Plots in Figure 5.6 indicate Gini indices values as a function of mid-point split points from the five models that are mentioned in Subsection 4.3.2.

Algorithm 3 Gini-sampled Stumps Method $p = 1$

1. Gini gain g is constant on the intervals $(x_{(i)}, x_{(i+1)})$ and we have $g(x) = 0$ for $x < x_{(1)}$ and $x > x_{(q)}$
2. Generate random samples with density $\frac{1}{S}\ g$, where

$$S = \sum_{i=1}^{q-1} g\left(\frac{x_{(i)} + x_{(i+1)}}{2}\right)(x_{(i+1)} - x_{(i)})$$

3. For the inverse transform method, find the *cdf*

$$G(x) = \frac{1}{S}\left[\sum_{i=1}^{k} g\left(\frac{x_{(i)} + x_{(i+1)}}{2}\right)(x_{(i+1)} - x_{(i)}) + g(x)(x - x_{(k+1)})\right],$$

   where $k = \max\{i | x_{(i+1)} \leq x\}$, and let $G_k = G(x_{(k)})$
4. Sample from G by using the inverse transform method as follows:
   For $(b \leftarrow 1 \quad to \quad B)$
   (a) Generate a random value uniformly distributed $u \sim U(0, 1)$
   (b) Find $v = \max\{i | G(x_i) \leq u\}$, so that $G_v \leq u \leq G_{v+1}$
   (c) Find the slope of the v^{th} line segment $a = \frac{u - G_v}{G_{v+1} - G_v}$
   (d) Finally, compute $x_{(v)} + a(x_{(v+1)} - x_{(v)})$
5. Make a stump of every split point from the previous step

Algorithm 4 Gini-sampled Stumps Method $p > 1$

1. For $l = \{1, 2, \ldots, p\}$, find the Gini values g_l of the increasingly ordered midpoints
2. Sum the values of $\{g_1, g_2, \ldots, g_p\}$ for each variable separately and the result is a vector of length p, refer to this vector as $\mathcal{S}_{Gini}$, then sample from this vector with probability equals to $\mathcal{S}_{Gini}$
3. Take the sampled variable l and apply Algorithm 3

5.1.2.2 Gini-midpoints Stumps Method

The way split points are chosen in Gini-sampled stumps method is by using inverse transform method to generate a split point randomly from the range of midpoints

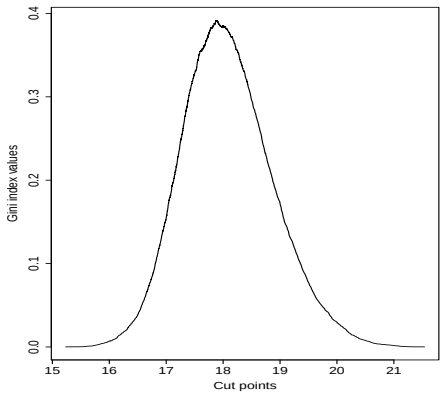
and repeat this B times. We are going to modify this method at the generating step and call this method Gini-midpoints stumps. Gini-midpoints stumps can be explained as in Algorithm 5.

Algorithm 5 Gini-midpoints Stumps Method $p = 1$

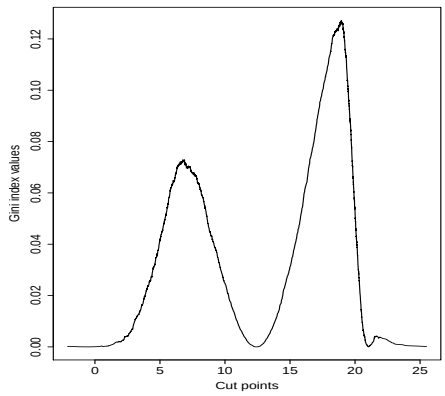
1. Gini gain g is constant on the intervals $(x_{(i)}, x_{(i+1)})$ and we have $g(x) = 0$ for $x < x_{(1)}$ and $x > x_{(q)}$
2. Sample from the mid-points vector with respect to the corresponding Gini indices g
3. Make a stump of every split point from the previous step

Algorithm 6 Gini-midpoints Stumps Method $p > 1$

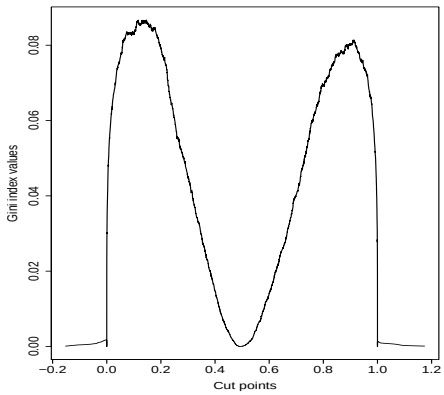
1. For $l = \{1, 2, \ldots, p\}$, find the Gini values g_l of the increasingly ordered midpoints
2. Sum the values of $\{g_1, g_2, \ldots, g_p\}$ for each variable separately and the result is a vector of length p, refer to this vector as $\mathcal{S}_{Gini}$, then sample from this vector with probability equals to $\mathcal{S}_{Gini}$
3. Take the sampled variable l and apply Algorithm 5



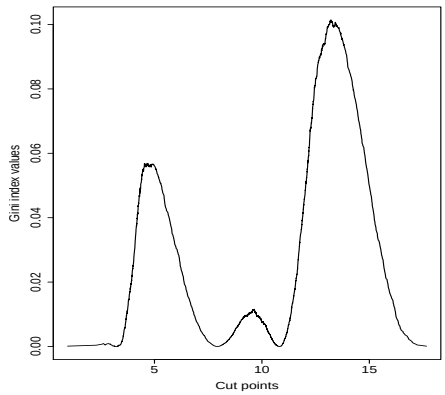
(a) Model 1



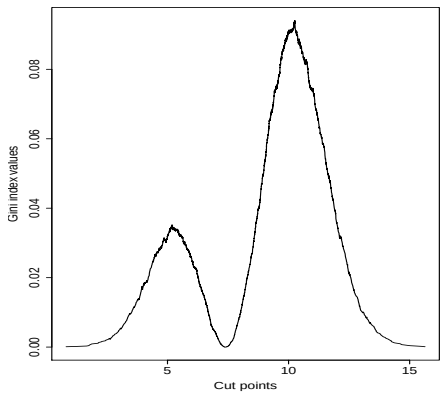
(b) Model 2



(c) Model 3



(d) Model 4



(e) Model 5

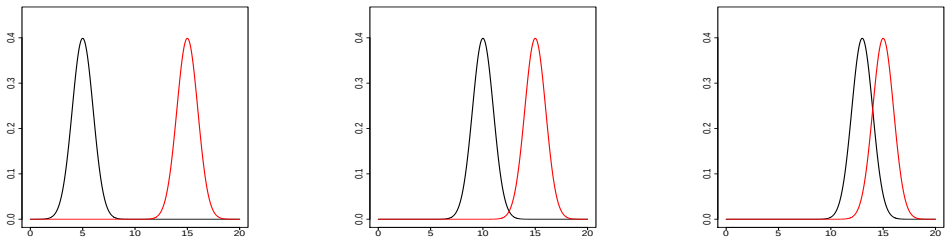
Figure 5.6: Gini gain as a function of split points from the five models. The highest values in these curves occur at the overlapping areas between the densities in the corresponding model as in Figure 4.5.

5.1.2.3 Comparing Between Gini-sampled Stumps and Gini-midpoints Stumps Methods

These two methods differ in the way they generate the split points from randomly sampling by using the inverse transform method in Gini-sampled stumps to sampling from the midpoints with probability equals to Gini values in Gini-midpoints stumps. The accuracy performance of these two methods is estimated by using three models, where each of these models has one explanatory variable, a binary response variable, where classes are equally likely, and number of observations is $n = 2000$. The distribution of the explanatory variables of each model are shown in Table 5.1. The *pdf*s of the explanatory variables of these models according to classes can be seen in Figure 5.7. Table 5.2 indicates the averages of 50 simulations of the three models.

Model	Class 1	Class 2
1	$N(5, 1)$	$N(15, 1)$
2	$N(10, 1)$	$N(15, 1)$
3	$N(13, 1)$	$N(15, 1)$

Table 5.1: The *pdf*s of the three models which are used to test the performance of Gini methods.



(a) Model 1 (b) Model 2 (c) Model 3

Figure 5.7: Three models to test the two Gini methods.

From the results of the three models in Table 5.2, we find that Gini methods (sampling and midpoints) performs similarly and very well compared to the maximal accuracy.

Model	Gini-sampled stumps		Gini-midpoints stumps		Maximal accuracy
	MV	WV	MV	WV	
1	100	100	99.25	99.54	100
2	99.36	99.36	99.99	99.18	99.99
3	84.28	84.31	84.27	84.30	84.35

Table 5.2: Average of 50 percentages of correctly classified observations from the three models mentioned in Table 5.1.

5.2 Applying The Two Forests of Stumps Methods to Simulated Models with One Explanatory Variable

In this section, we will apply the methods mentioned in Section 5.1 combined with decision aggregation methods as explained in Section 3.6 to the five models as in Subsection 4.3.2 to measure the performance. There are six methods that will be investigated as in Table 5.3.

	Aggregation methods	
Method	Majority vote (MV)	Weighted vote (WV)
Bagging with stumps	Bagging with stumps with MV	Bagging with stumps with WV
Gini-sampled stumps	Gini-sampled stumps with MV	Gini-sampled stumps with WV
Gini-midpoints stumps	Gini-midpoints stumps with MV	Gini-midpoints stumps with WV

Table 5.3: Forests of stumps with aggregation methods.

5.2.1 Result of Bagging with Stumps Method $p = 1$

Table 5.4 represents the prediction performance of different tree-based methods for the five models. These methods are: bagging with stumps with MV, bagging with stumps with WV, a forest of trees with MV and a forest of trees with WV. The term bagging with stumps is a group of a specific number B of stumps, that are bagged by using one of the aggregation methods whereas forests of trees is generally understood to mean a group of B large trees.

These methods are fitted on a bootstrap of size n^* that is drawn from a training dataset that has n observations, and finally predicted on a testing data which has m observations. Here, $n = 1000, m = 1000, B = 500$ and $n^* \in \{20, 40, 60, 80, 100, 200, 1000\}$.

Table 5.4 shows that forests of stumps perform more accurately than forests of trees in model 1. The best performance appears at the sample size $n^* = 20$ and with WV. One can see that bagging with stumps has similar results with either MV or WV that is a little higher than 94%. However, forests of trees accuracy improves with the increasing in the sample size.

Model 2 has different performance pattern where bagging with stumps with MV accuracy is noticeably less than bagging with stumps with WV. Bagging with stumps with MV performance is in the interval $(72.51\%, 73.01\%)$ while bagging with stumps with WV performance drops from 84.37% at sample size $n^* = 20$ to 73.07% when the bootstrap size is equal to the training data. Forests of trees performance decreases from nearly 89.88% at $n^* = 80$ with WV to 85.09% at $n^* = n$.

Model 3 performance decreases when the bootstrap sample size increase in forests of stumps and trees. Bagging with stumps with WV has much better accuracy than the same method with MV. The accuracy of bagging with stumps especially with sample size 100 and smaller is almost as good as forests of trees.

Model 4 and 5 have similar performance pattern. Bagging with stumps with both aggregation methods perform poorly whereas forests of trees give more accu-

rate results. In the first three models, bagging with stumps method has a higher performance with WV and smaller sample size $\{20, 40\}$.

In terms of the aggregation methods, WV aggregation method wins regardless whether it combines with bagging with stumps or forest of trees with higher performance percentages with the latter method, generally. The numbers in red in Table 5.4 occur with WV and they are close to the maximal accuracies in Table 5.5.

5.2.2 Result of Gini Stumps Methods $p = 1$

We will show the results of Gini stump methods in the next two sub-subsections.

5.2.2.1 Gini-sampled Stumps Result

Table 5.5 shows the percentages of correctly predicted classes by using Gini-sampled stumps method on the five models that are mentioned in Subsection 4.3.2 and $B = 500$. The performance of model 1 with Gini-sampled stumps method gets better when κ increases. While model 1 does not show a large difference between the results of the two aggregation methods, Model 2 does have a significant difference between percentages from MV and WV in favour of the latter especially with smaller values of κ. Model 3 also shows a large difference between the results from the two aggregation methods with better performance with WV. Model 4 does not show large differences between the two method results apart from when $\kappa = 0$. Finally, model 5 shows similar performance to model 2, which gives better results with WV and smaller values of κ.

Comparing general results of the two aggregation methods, WV gives better accuracy than MV especially with the smaller values of $\kappa = (0, 1/3, 1/2, 1)$. The (red) best results appear in WV column with smaller values of κ and they are similar to the maximal accuracies, especially for models 1 and 3.

To conclude, comparing between bagging with stumps' results in Table 5.4 and Table 5.5 shows that Gini-sampled stumps method is doing better than bagging

Model	Sample size n^*	Bagging with stumps		Forest of trees		Maximal accuracy
		MV	WV	MV	WV	
1	20	94.08	94.10	92.67	92.91	94.21
	40	94.06	94.09	92.94	93.02	
	60	94.06	94.06	93.05	93.10	
	80	94.05	94.08	93.03	93.03	
	100	94.01	94.03	93.10	93.13	
	200	94.03	94.04	93.16	93.18	
	1000	94.02	94.02	93.20	93.20	
2	20	72.63	84.37	89.46	89.45	92.87
	40	72.51	83.12	89.52	89.50	
	60	72.67	78.82	89.67	89.70	
	80	72.75	76.38	89.87	89.88	
	100	72.83	75.31	89.86	89.87	
	200	72.97	74.79	89.66	89.66	
	1000	73.01	73.07	85.09	85.09	
3	20	72.47	82.94	83.11	83.12	83.51
	40	69.03	82.86	82.97	82.98	
	60	66.54	82.79	82.82	82.83	
	80	66.30	82.71	82.72	82.69	
	100	65.79	82.43	82.61	82.63	
	200	65.54	80.95	81.86	81.88	
	1000	66.09	72.27	75.80	75.80	
4	20	66.47	66.52	84.66	83.96	87.37
	40	66.49	66.52	84.58	84.29	
	60	66.49	66.51	84.71	84.54	
	80	66.48	66.53	84.77	84.60	
	100	66.50	66.51	84.89	84.77	
	200	66.50	66.54	84.96	84.92	
	1000	66.52	66.54	85.02	85.03	
5	20	69.35	69.72	73.14	73.29	78.51
	40	69.42	69.71	73.49	73.63	
	60	69.44	69.71	73.58	73.62	
	80	69.34	69.66	73.74	73.80	
	100	69.35	69.69	73.68	73.72	
	200	69.39	69.71	73.96	73.98	
	1000	69.35	69.70	74.12	74.13	

Table 5.4: The average of percentages of correctly predicted classes for 50 simulations from five models by using bagging with stumps and forests of trees with the two aggregation methods and the results in red are the largest results out of all results.

with stumps in four models out of five.

5.2.2.2 Gini-midpoints Stumps Result

Table 5.5 shows the percentages of correctly predicted classes by using Gini-modpoints stumps method with the five models. The performance of this method with model 1 is similar for all κ values and the two aggregation methods, which is around 93.50%. This method shows a better performance at $\kappa = 0$ than Gini-sampled stumps with MV. Model 2 shows a poor performance at smaller κ and with MV aggregation method, however, the largest percentage appears when $\kappa = 0$ with WV. The performance of this method increases by increasing κ values with WV and decreases with MV. In model 3, Gini-midpoints stumps with MV accuracy increases from 53% to 63% by increasing κ and decreases with WV. Model 4 shows similar accuracy with this method with the two aggregation methods and for different values of κ with a better performance at $\kappa = 0$ and with WV. This method accuracy with model 5 acts in a way the performance increases by increasing κ values and vice versa with WV.

By comparing the general results of the two aggregation methods with Gini-midpoints stumps, WV has the largest values with all five models. Finally, Gini-midpoints stumps method gives better performance than Gini-sampled stumps method in just one model.

Model	κ	Gini-sampled stumps		Gini-midpoints stumps		Maximal accuracy
		MV	WV	MV	WV	
1	0	91.85	93.76	93.40	93.70	94.21
	1/3	92.34	93.99	93.65	93.84	
	1/2	93.00	94.03	93.69	93.85	
	1	93.67	94.09	93.80	93.87	
	2	94.01	94.14	93.86	93.89	
	3	94.11	94.16	93.91	93.94	
	4	94.14	94.14	93.94	93.95	
2	0	54.11	87.82	54.91	85.97	92.87
	1/3	58.89	87.26	57.62	85.79	
	1/2	60.14	86.99	60.08	85.51	
	1	67.13	85.26	64.28	83.71	
	2	71.99	79.40	69.45	79.25	
	3	72.58	74.68	71.27	75.88	
	4	72.73	74.01	72.09	74.51	
3	0	78.94	82.91	53.36	82.41	83.51
	1/3	68.60	83.05	54.50	82.44	
	1/2	65.88	83.03	55.95	82.30	
	1	63.22	82.99	57.68	82.28	
	2	64.09	82.74	61.10	81.96	
	3	64.46	82.04	62.29	81.62	
	4	64.96	80.29	63.61	80.90	
4	0	66.86	67.60	67.19	68.05	87.37
	1/3	66.11	64.85	66.02	66.30	
	1/2	66.15	64.55	66.71	65.99	
	1	66.22	65.14	66.18	65.26	
	2	66.31	65.70	66.47	65.18	
	3	66.26	65.99	66.58	65.84	
	4	66.24	66.15	66.55	66.16	
5	0	50.87	75.84	63.71	72.78	78.51
	1/3	57.17	73.78	66.64	70.69	
	1/2	60.91	72.83	67.45	70.03	
	1	66.81	70.11	68.54	69.55	
	2	68.78	69.84	69.10	69.62	
	3	69.18	69.85	69.23	69.61	
	4	69.27	69.88	69.20	69.67	

Table 5.5: The average of percentages of correctly predicted classes of 50 simulations from the five models by using Gini stumps methods with raising Gini indices to different powers κ. Red results represent the largest results from Gini-sampled stumps and green from Gini-midpoints stumps. The underlined results represent the largest results between the two Gini methods.

5.3 Applying The Two Forests of Stumps Methods to Multivariate Models

We investigate the performance of these two methods with models that have one explanatory variable in Section 5.2 and now we will investigate them with models that have more than one explanatory variable. Table 5.6 shows three models each of them has a response variable that has two classes $\{C_1, C_2\}$, each of these models has p variables, where p takes one of the values $\{10, 30, 50, 100, 150\}$. The number of observations is $n = 1000$ in the simulations that have p is equal to one of the values $\{10, 30, 50, 100\}$, and the number of observations is $n = 100$ in the simulations that have $p = 150$. Observations are equally likely in each class and the explanatory variables are normally distributed $C_1 \sim N(\mu_1, \sigma_1^2 I)$, $C_2 \sim N(\mu_2, \sigma_2^2 I)$, where $\underline{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$, and $\mu_2 = \mu_1 + \delta \cdot \mathbb{1}$.

Table 5.6 shows the models are used, and Table 5.7 and Table 5.8 indicate the prediction accuracy on these models with different number of variables by using bagging with stumps and Gini stumps methods, respectively. Last row of each group of results of a model shows a model that has $p > n$.

Model	μ_1	δ	σ_1	σ_2
1	$\mathbf{0}$	1	1	1
2	$\mathbf{0}$	1	0.5	2
3	$\mathbf{0}$	1	0.5	0.5

Table 5.6: Multivariable models to test the performance of the two forests methods.

5.3.1 Result of Bagging with Stumps Method $p > 1$

Table 5.7 shows the results of applying bagging with stumps method with the multivariate models as in Table 5.6. The results in the table are similar within each model even when the number of variables differs. The results of bagging with

stumps with WV are larger than with MV in most cases and that is compatible with the previous results when the model has one explanatory variable.

Model	No. of variables	MV	WV
1	10	68.34	68.17
	30	67.88	68.47
	50	67.50	68.34
	100	68.17	68.88
	150	68.47	68.50
2	10	73.48	73.96
	30	74.24	74.84
	50	72.60	73.48
	100	73.96	74.24
	150	74.84	74.60
3	10	83.82	83.96
	30	83.35	83.14
	50	83.10	83.82
	100	83.96	83.35
	150	83.14	83.20

Table 5.7: The average of 10 percentages of correctly predicted classes from 3 multivariate models by using bagging with stumps method.

5.3.2 Result of Gini Stumps Methods $p > 1$

Table 5.8 shows the results of applying Gini stumps methods to multivariate models with different number of variables. It is clearly seen from the table that the average of percentages increases when the number of variable increases. Also, the results of Gini-midpoints stumps method is better than Gini-sampled stumps method with all three models. The performance accuracy of WV aggregation method is larger than MV regardless of the Gini stumps method. Therefore, out of the six methods here Gini-midpoints stumps with WV has the best performance.

By comparing between bagging with stumps and the two methods of Gini stumps, we find that Gini stumps method gives much more accurate results than bagging with stumps with all the models.

Model	No. of variables	Gini-sampled stumps		Gini-midpoints stumps	
		MV	WV	MV	WV
1	10	92.92	93.26	92.94	93.28
	30	99.24	99.28	99.90	99.99
	50	99.86	99.89	99.75	99.82
	100	99.01	99.10	99.30	99.94
	150	100.00	100.00	100.00	100.00
2	10	80.26	87.96	82.14	88.41
	30	81.79	92.86	84.47	93.94
	50	83.60	99.98	95.90	96.63
	100	87.14	89.10	99.76	99.99
	150	89.60	100.00	100.00	100.00
3	10	99.85	99.87	99.83	100.00
	30	100.00	100.00	100.00	100.00
	50	100.00	100.00	100.00	100.00
	100	100.00	100.00	100.00	100.00
	150	100.00	100.00	100.00	100.00

Table 5.8: The average of 10 percentages of correctly predicted classes from the three models by using Gini-sampled and Gini-midpoints stumps methods.

5.4 Comparing Between Bagging with Stumps & Gini Stumps

There are two main differences between bagging with stumps and Gini stumps methods. The first difference is the way split points are determined. That is, picking the split point to maximise the Gini gain in each bootstrap sample (in bagging with stumps), generating random split points according to Gini indices distribution (in Gini-sampled stumps) and sampling from mid-points split points with respect to Gini indices (in Gini-midpoints stumps). The second difference is the way to determine the class distribution in each leaf. For bagging with stumps method, this is defined by checking bootstrap sample, while for both Gini stumps methods the whole training dataset is checked.

Two comparisons are carried out here to test these two methods performance. The first comparison is to compare the consumed time. The second one is to test whether the split points from the different methods are distributed differently or not. We will focus here on comparing between bagging with stumps and Gini-sampled stumps method. We choose Gini-sampled stumps method to run the comparisons because it generates the split points slower than the Gini-midpoints stumps method due to the more complicated way of the split points generation process for the former method. The next two subsections discuss these two comparisons in more detail.

5.4.1 The Consumed Time for Generating Split Points

A comparison is carried out to compare between the two subject of study methods in terms of their consumed time to generate 500 split points. Gini-sampled stumps are fitted on a training data which has $n = 1000$, and bagging with stumps are fitted on different bootstrap sample sizes n^* that drawn from a training data that has $n = 1000$ and the bootstrap sample size $n^* \in \{40, 60, 80, 100, 200, 500, 1000\}$.

Figure 5.8 indicates the needed time to generate 500 split points from Gini-sample stumps and bagging with stumps methods. Bagging with stumps method

consumes more time when the bootstrap size increases. The range of the consumed time by bagging with stumps is between one second to two seconds for bootstrap size from 40 to 1000. In general, Gini-sample stumps method is faster than bagging with stumps even for smaller bootstrap sample sizes.

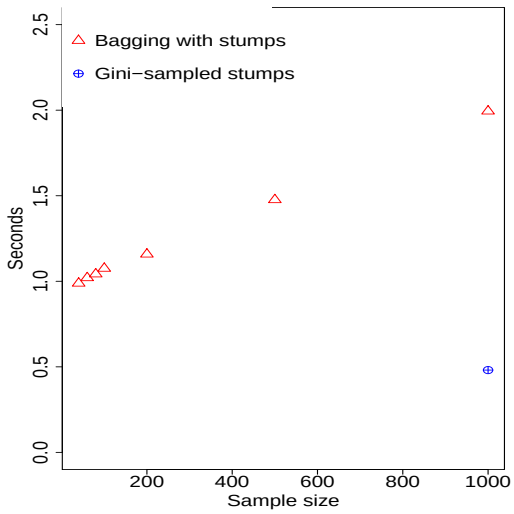


Figure 5.8: Consumed time to generate 500 split points by the two stumps approaches. Gini-sampled stumps are fitted on a training data that has n observations and bagging with stumps are trained on different bootstrap sample sizes n^*. These bootstrap sample size are drawn from a training dataset which has $n = 1000$.

5.4.2 Comparing The Distribution of Split Points From Bagging with Stumps & Gini-sampled Stumps

We now will compare between the distribution of the split points from bootstrap samples and the split points that are generated according to distribution of the Gini index values.

As we can see from Figures 5.1 to 5.5 that the distribution of split points from bootstrap samples depends on n^*, which is the size of the sample of bootstrap which is drawn from the training data. We here compare between the split points generated from bagging with stumps with different bootstrap sample sizes and

the split points generated from Gini-sampled stumps after raising Gini values to different powers κ to have more flexibility in Gini curves.

The null hypothesis is that the split points from bootstrap samples and split points that are generated according to distribution of the optimised (transformed) Gini indices are from the same distribution. Two-sample Anderson-Darling test is used to test this hypothesis. The null distribution of this statistic is calculated under the null hypothesis that the samples are drawn from the same distribution.

Simulation

In order to test this comparison, we run the following steps:
Input
$n = 1000$,
$A = 50$ simulations,
$Q = (1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6)$,
$\psi, = (20, 40, 60, 80, 100)$

For $(n^* \in \psi,)$

1. **For** $(a \leftarrow 1 \quad to \quad \text{length}(A))$

   (a) Generate 100 split points from bagging with stumps method in Algorithm 2 with bootstrap sample size n^*

   (b) Generate 100 split points from Gini-sampled stumps method in Algorithm 3

   (c) In order to find a power κ which should give split points generated by using Gini-sampled stumps method that are similarly distributed to split points which are generated by using bagging with stumps with a specific sample size n^* (we will refer to this power as an optimal power), we transform Gini index values according to values of this vector, Q

(d) **For** ($\kappa \leftarrow 1 \quad to \quad \text{length}(Q)$)

i. Use two-sample Anderson-Darling test to find the $D_{a\kappa}$ statistic between the empirical *cdf* of the split points in Step 1a and the empirical *cdf* of the transformed Gini in Step 1c

ii. Save $D_{a\kappa}$ in a matrix column, at the end we will have a matrix $A \times Q$

$$\begin{bmatrix} D_{1\kappa_1} & D_{1\kappa_2} & \dots & D_{1\kappa_Q} \\ D_{2\kappa_1} & D_{2\kappa_2} & \dots & D_{2\kappa_Q} \\ \vdots & \vdots & \ddots & \vdots \\ D_{A\kappa_1} & D_{A\kappa_2} & \dots & D_{A\kappa_Q} \end{bmatrix},$$

here, rows represent how much these values varies from simulation to another and columns represent the used power κ

2. Take the mean of each column of the matrix in Step 1(d)ii, then determine the location of the smallest value and choose its column number to be the optimal κ, i.e. if, for example, the smallest value's location is in column number three and that means $\kappa = 2$ of Gini indices gives the smallest distance between the two *cdf*s and this value is considered the most optimal κ

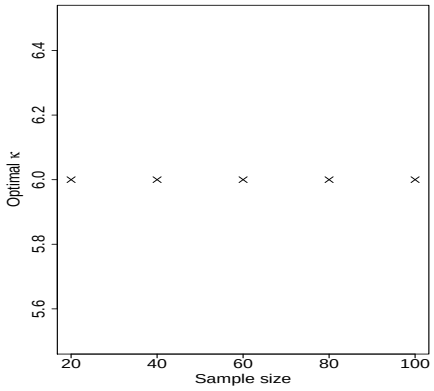
Figure 5.9 indicates the optimal κ for the five models and each n^*. Figure 5.10, Figure 5.12, Figure 5.14, Figure 5.16 and Figure 5.18 show histograms of the pooled split points generated from bootstrap of size n^* from all simulations $a = \{1, 2, \dots, A\}$ and the red curves are transformed Gini indices according to κ values that are corresponding to bootstrap size n^* as in Figure 5.9. The red curves are obtained from the first simulation of each model (i.e. $a^* = 1$) and because of the use of the red curve from the first simulation Figure 5.10, Figure 5.12, Figure 5.14, Figure 5.16 and Figure 5.18 might be a little misleading.

Figure 5.11, Figure 5.13, Figure 5.15, Figure 5.17 and Figure 5.19 show the distance between the two empirical *cdf*s of the two methods. The green curve indi-

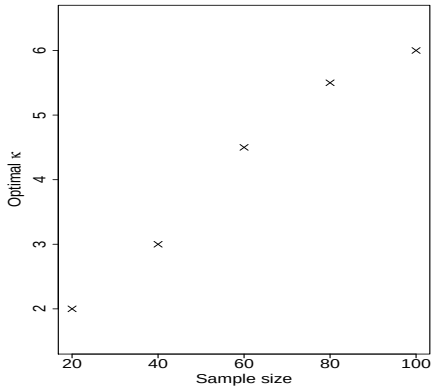
cates the empirical cumulative distribution functions of split points from bagging with stumps with different n^* and purple curve shows the empirical cumulative distribution function of split points from Gini-sampled stumps transformed according to the optimal κ as in Figure 5.9. It can be seen that the values of κ in model 1 are equal to 6 for all sample sizes, while these values for the other four models fluctuate between 2 and 6 for model 2, 1 and 3.5 for model 3, 1 and 2.5 for model 4, and 1 and 5.5 for model 5.

The optimal κ values match the histogram in model 1 and the distance between the two empirical *cdf*s are very small. However, this distance gets a little larger when the sample size n^* increases. Model 2 shows good matching between the histograms and the transformed Gini curves. The gap between the histograms and red curves enlarges when the n^* increases. This performance matches the large distances between the two empirical *cdf*s. Histograms from model 3 and the red curves show a good matching especially at $n^* = 80$ and $n^* = 100$. Model 4 indicates a large difference between the two *cdf*s which is consistent with histograms and red curves for model 4. Finally, model 5 histograms and Gini curves match slightly better comparing to the previous model especially at $n^* = 80, n^* = 100$ but still there is a difference between the two *cdf*s.

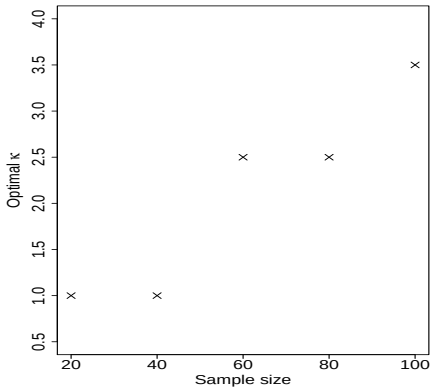
To conclude from that, the results from the models 2 to 5 show a difference between the distribution of the split points from bagging with stumps and Gini-sampled stumps and we rejected the null hypothesis.



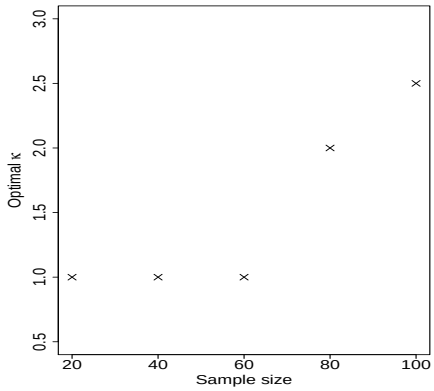
(a) Model 1



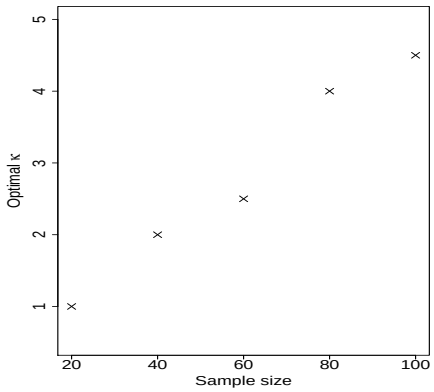
(b) Model 2



(c) Model 3

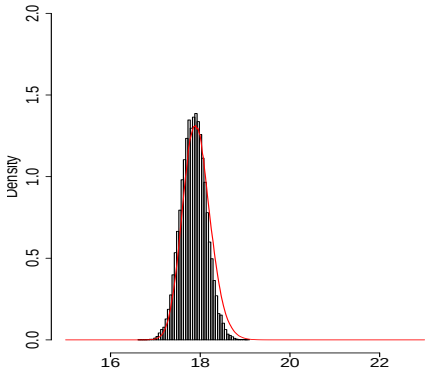


(d) Model 4

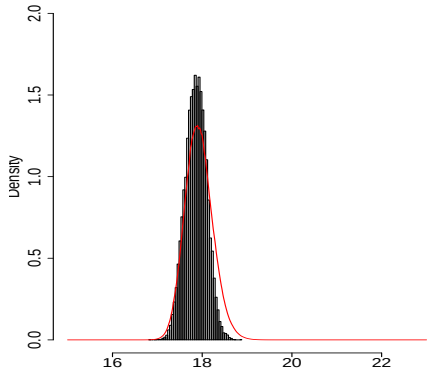


(e) Model 5

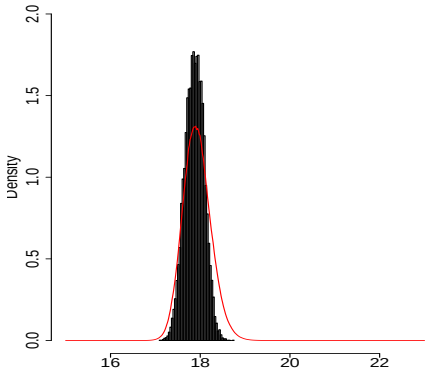
Figure 5.9: Optimal κ for the five models with different bootstrap sample size n^*.



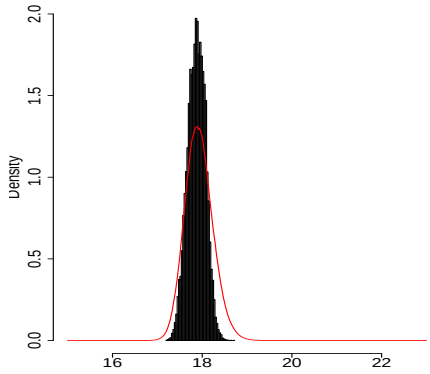
(a) $n^* = 20,\ \kappa = 6$



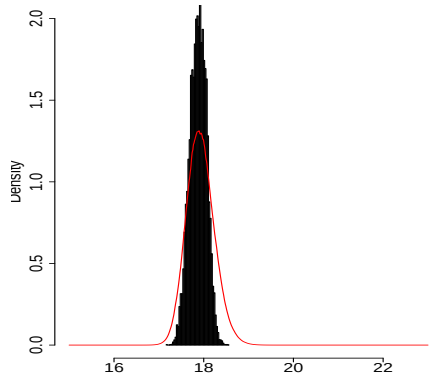
(b) $n^* = 40,\ \kappa = 6$



(c) $n^* = 60,\ \kappa = 6$

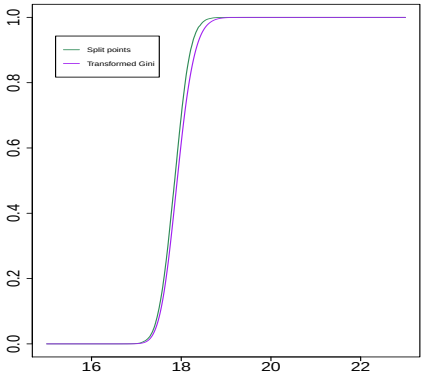


(d) $n^* = 80,\ \kappa = 6$

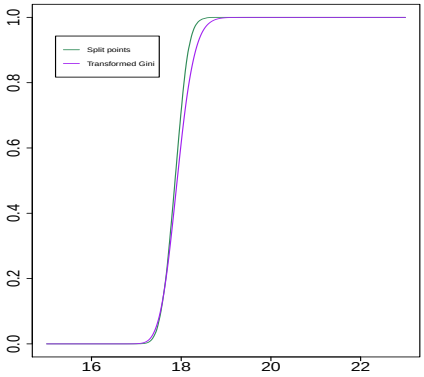


(e) $n^* = 100,\ \kappa = 6$

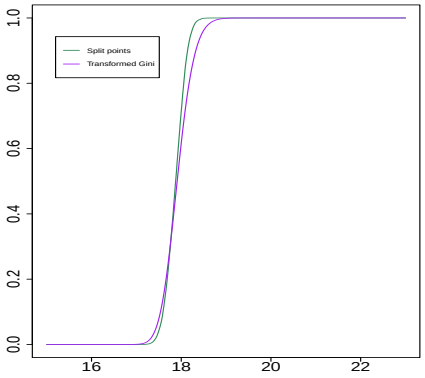
Figure 5.10: Histograms show the pooled split points from 50 simulations from model 1 with different bootstrap sample size n^* and power κ. The red curve shows Gini curve raised to the corresponding power κ as in Figure 5.9(a).



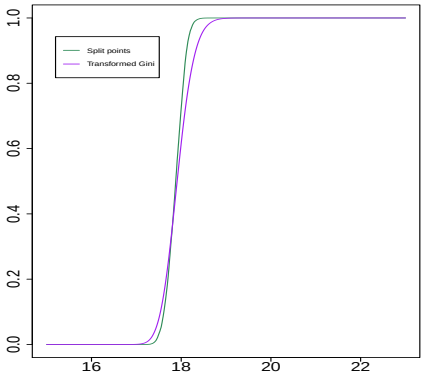
(a) $n^* = 20,\ \kappa = 6$



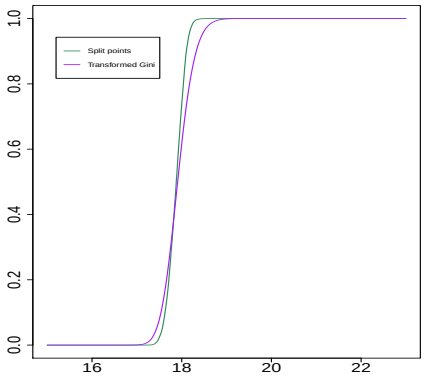
(b) $n^* = 40,\ \kappa = 6$



(c) $n^* = 60,\ \kappa = 6$

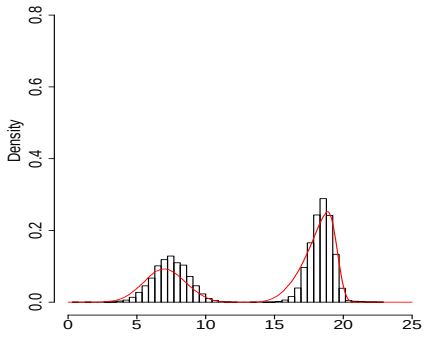


(d) $n^* = 80,\ \kappa = 6$

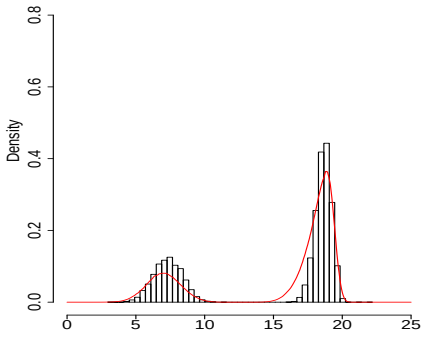


(e) $n^* = 100,\ \kappa = 6$

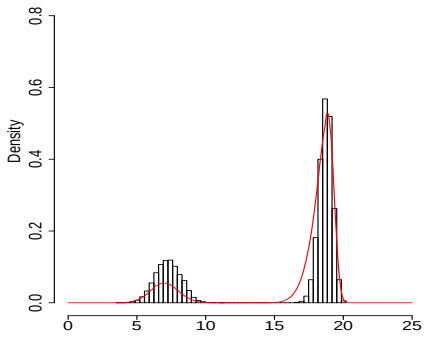
Figure 5.11: Two *cdf*s of the split points and transformed Gini indices according to κ from model 1. The green curve indicates the empirical cdf of split points from bagging with stumps and purple curve shows the empirical cdf of split points from Gini-sampled stumps.



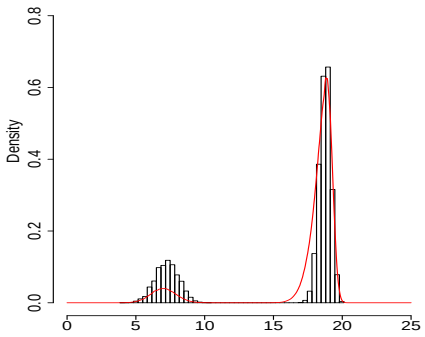
(a) $n^* = 20,\ \kappa = 2$



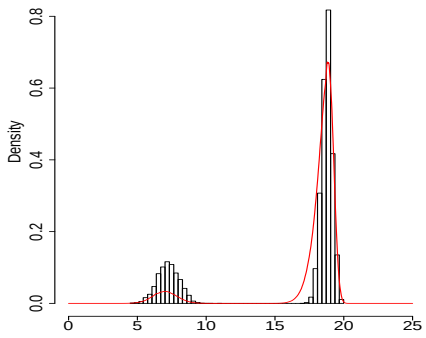
(b) $n^* = 40,\ \kappa = 3$



(c) $n^* = 60,\ \kappa = 4.5$

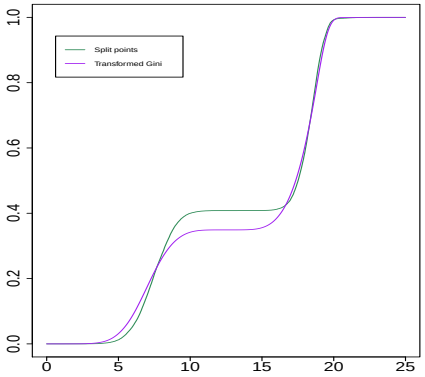


(d) $n^* = 80,\ \kappa = 5.5$

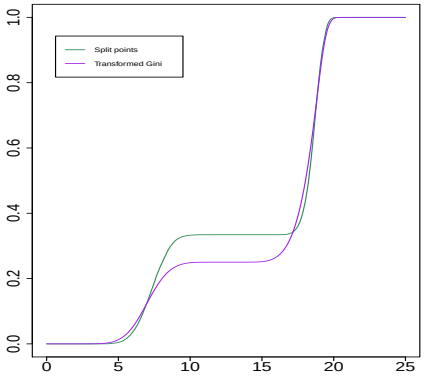


(e) $n^* = 100,\ \kappa = 6$

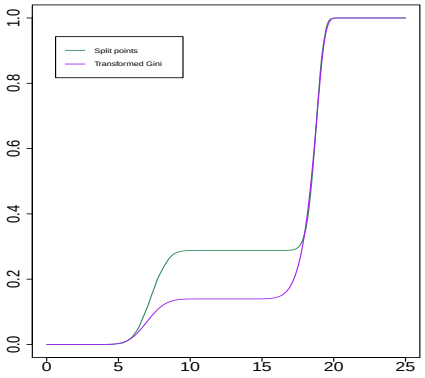
Figure 5.12: Histograms show the pooled split points from 50 simulations from model 2 with different bootstrap sample size n^* and power κ. The red curve shows Gini curve raised to the corresponding power κ as in Figure 5.9(b)



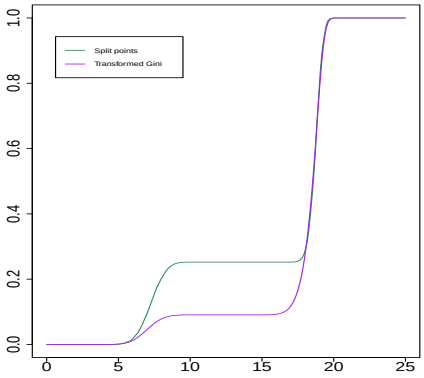
(a) $n^* = 20,\ \kappa = 2$



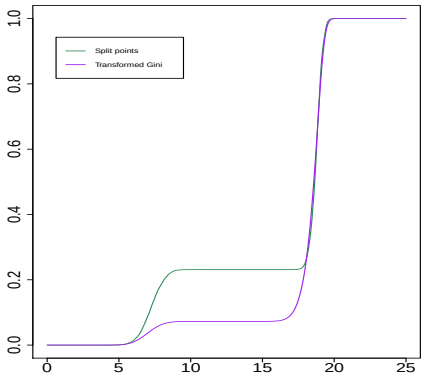
(b) $n^* = 40,\ \kappa = 3$



(c) $n^* = 60,\ \kappa = 4.5$

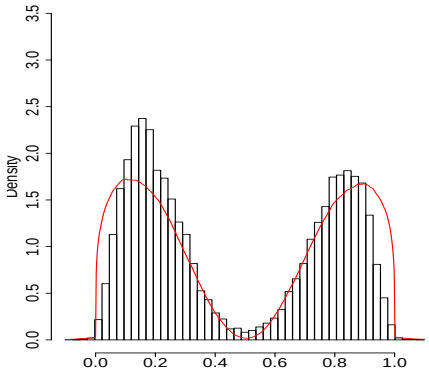


(d) $n^* = 80,\ \kappa = 5.5$

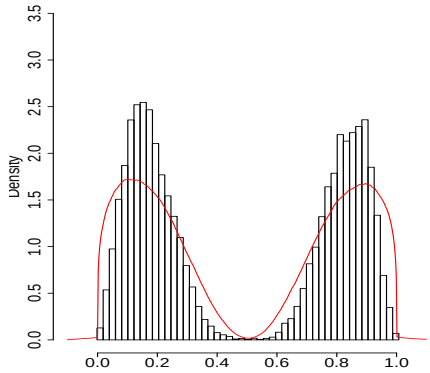


(e) $n^* = 100,\ \kappa = 6$

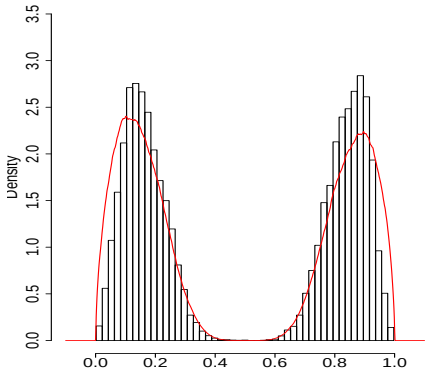
Figure 5.13: Two *cdf*s of the split points and transformed Gini indices according to κ from model 2. The green curve indicates the empirical cdf of split points from bagging with stumps and purple curve shows the empirical cdf of split points from Gini-sampled stumps.



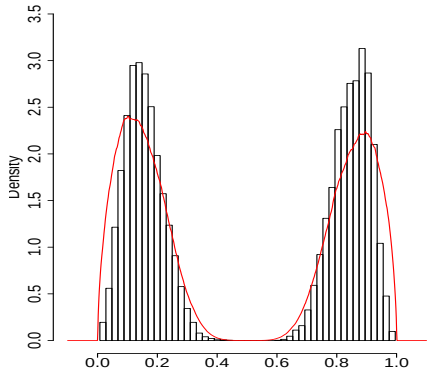
(a) $n^* = 20,\ \kappa = 1$



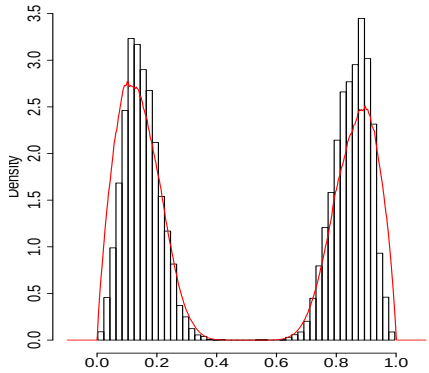
(b) $n^* = 40,\ \kappa = 1$



(c) $n^* = 60,\ \kappa = 2.5$

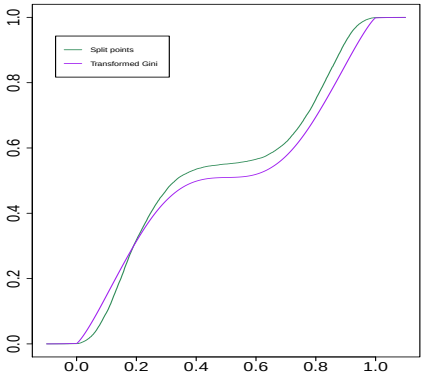


(d) $n^* = 80,\ \kappa = 2.5$

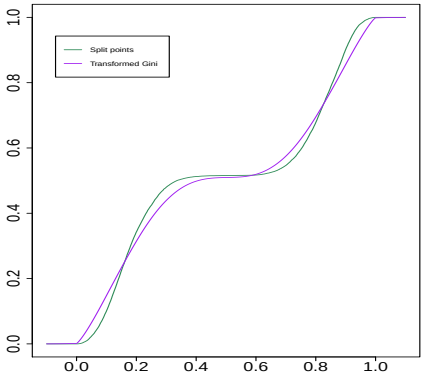


(e) $n^* = 100,\ \kappa = 3.5$

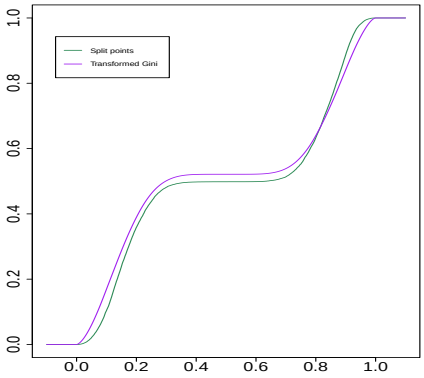
Figure 5.14: Histograms show the pooled split points from 50 simulations from model 3 with different bootstrap sample size n^* and power κ. The red curve shows Gini curve raised to the corresponding power κ as in Figure 5.9(c)



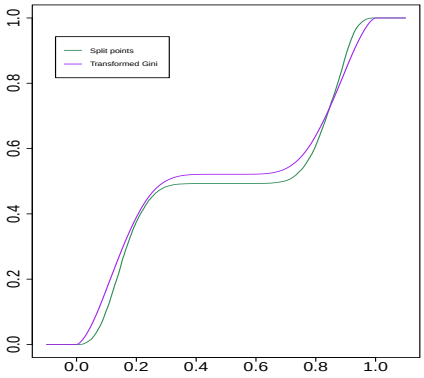
(a) $n^* = 20,\ \kappa = 1$



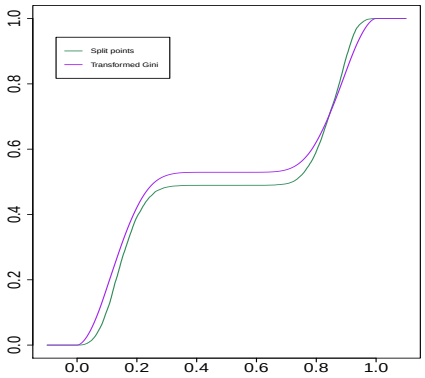
(b) $n^* = 40,\ \kappa = 1$



(c) $n^* = 60,\ \kappa = 2.5$

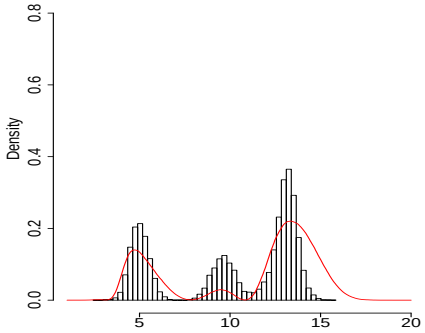


(d) $n^* = 80,\ \kappa = 2.5$

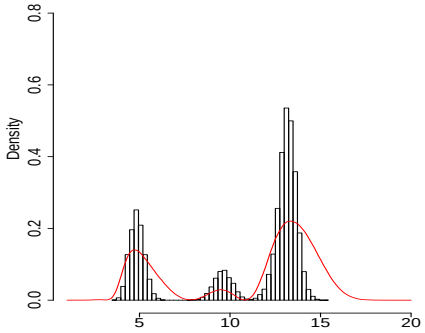


(e) $n^* = 100,\ \kappa = 3.5$

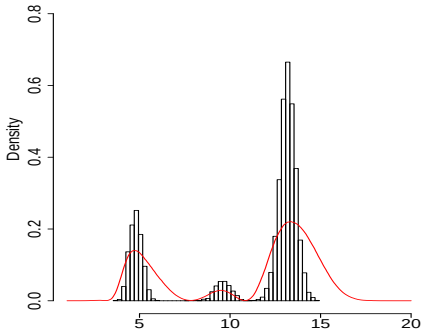
Figure 5.15: Two *cdf*s of the split points and transformed Gini indices according to κ from model 3. The green curve indicates the empirical cdf of split points from bagging with stumps and purple curve shows the empirical cdf of split points from Gini-sampled stumps.



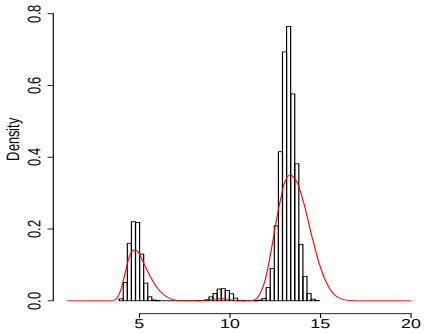
(a) $n^* = 20,\ \kappa = 1$



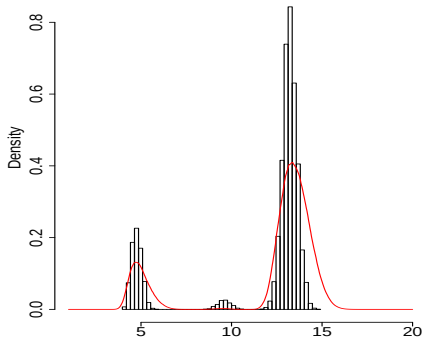
(b) $n^* = 40,\ \kappa = 1$



(c) $n^* = 60,\ \kappa = 1$

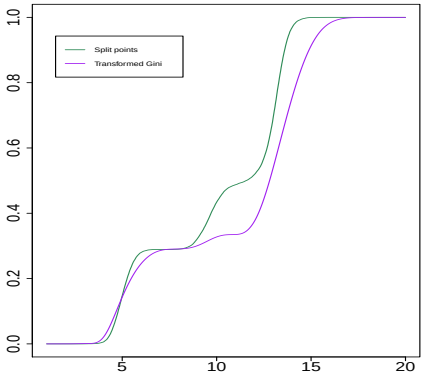


(d) $n^* = 80,\ \kappa = 2$

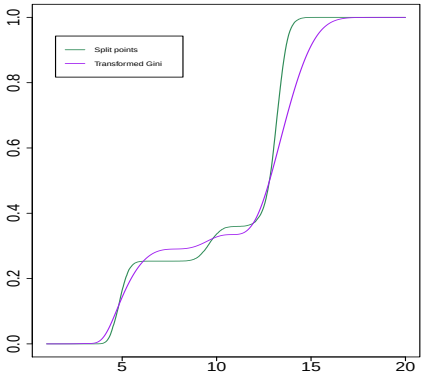


(e) $n^* = 100,\ \kappa = 2.5$

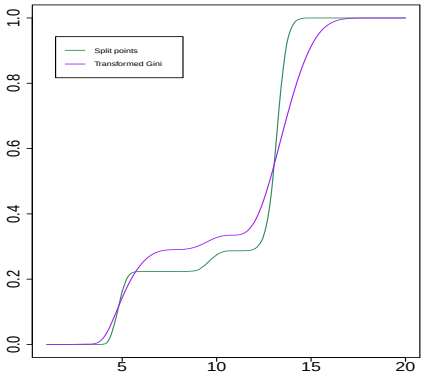
Figure 5.16: Histograms show the pooled split points from 50 simulations from model 4 with different bootstrap sample size n^* and power κ. The red curve shows Gini curve raised to the corresponding power κ as in Figure 5.9(d)



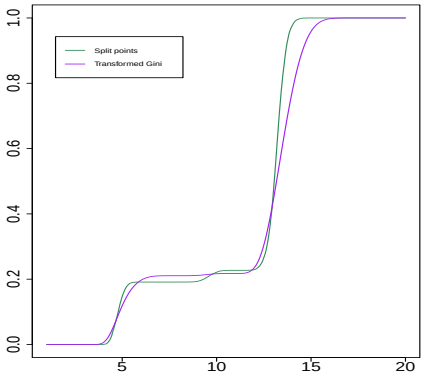
(a) $n^* = 20$, $\kappa = 1$



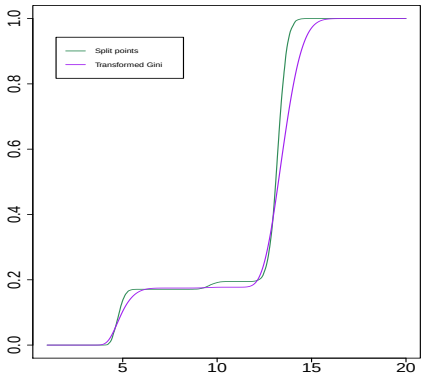
(b) $n^* = 40$, $\kappa = 1$



(c) $n^* = 60$, $\kappa = 1$

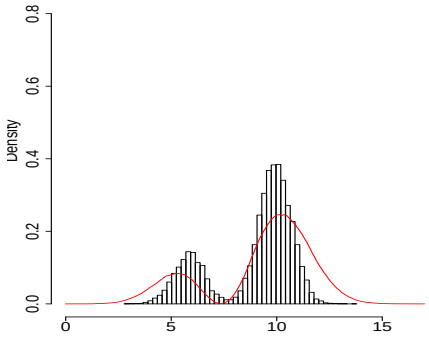


(d) $n^* = 80$, $\kappa = 2$

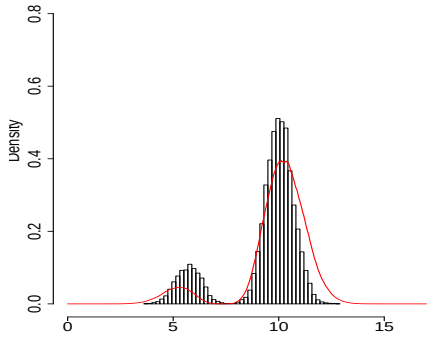


(e) $n^* = 100$, $\kappa = 2.5$

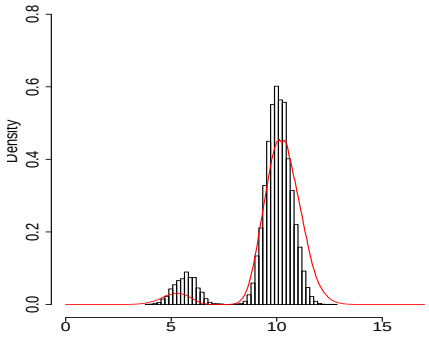
Figure 5.17: Two *cdf*s of the split points and transformed Gini indices according to κ from model 4. The green curve indicates the empirical cdf of split points from bagging with stumps and purple curve shows the empirical cdf of split points from Gini-sampled stumps.



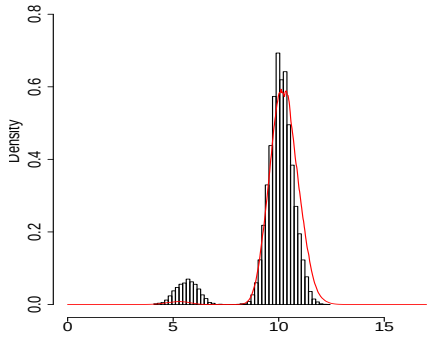
(a) $n^* = 20,\ \kappa = 1$



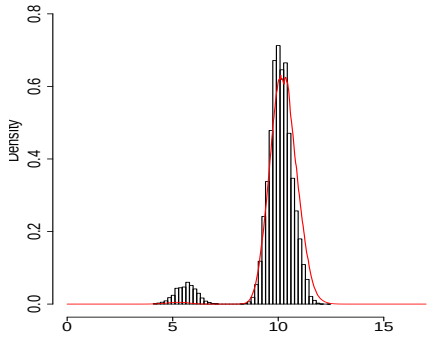
(b) $n^* = 40,\ \kappa = 2$



(c) $n^* = 60,\ \kappa = 2.5$

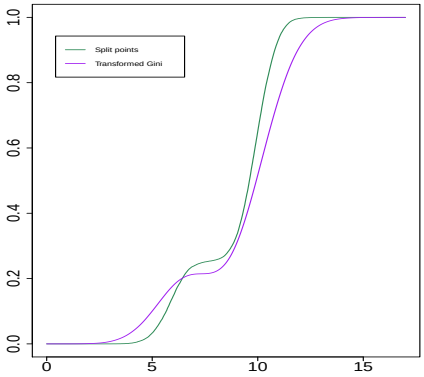


(d) $n^* = 80,\ \kappa = 4$

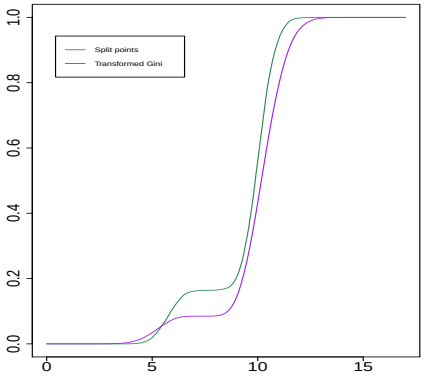


(e) $n^* = 100,\ \kappa = 4.5$

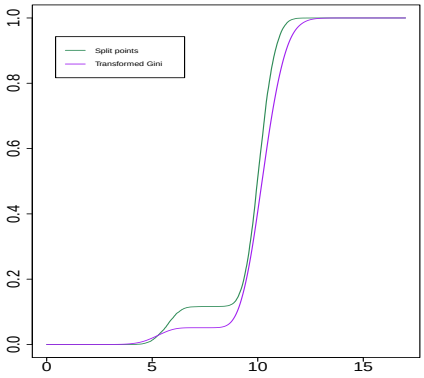
Figure 5.18: Histograms show the pooled split points from 50 simulations from model 5 with different bootstrap sample size n^* and power κ. The red curve shows Gini curve raised to the corresponding power κ as in Figure 5.9(e)



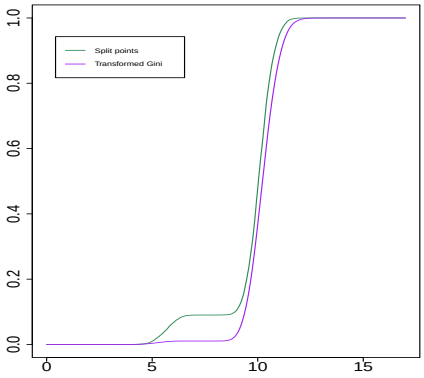
(a) $n^* = 20$, $\kappa = 1$



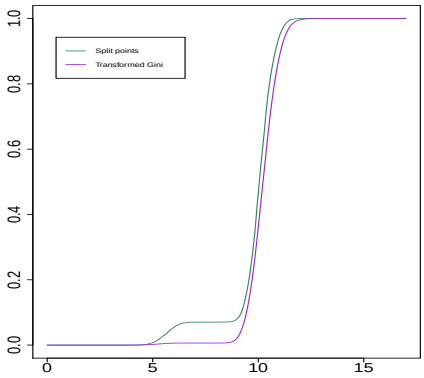
(b) $n^* = 40$, $\kappa = 2$



(c) $n^* = 60$, $\kappa = 2.5$



(d) $n^* = 80$, $\kappa = 4$



(e) $n^* = 100$, $\kappa = 4.5$

Figure 5.19: Two *cdf*s of the split points and transformed Gini indices according to κ from model 5. The green curve indicates the empirical cdf of split points from bagging with stumps and purple curve shows the empirical cdf of split points from Gini-sampled stumps.

5.5 Summary of the Chapter

We can summarise this chapter in a few points:

1. There are two main differences between the methods of stumps that are investigated in this chapter:
   (a) While bagging with stumps method uses bootstrap sample size n^* that are drawn from a training dataset to determine the split points, Gini stumps methods use the whole training data
   (b) The way the class distribution is determined at each leaf for each method, for bagging with stumps is by checking the bootstrap samples and for Gini stumps by checking the whole training data
2. Gini-sampled stumps method shows a reduction in computing time to generate a set of split point. The time needed to generate 500 split points by Gini-sampled stump method is around 0.5 second while it is almost 2 seconds by using the Bagging with stumps method as it is shown in Figure 5.8
3. By comparing between bagging with stumps and Gini stumps methods in terms of their prediction accuracies, we find that Gini stumps perform more accurately than bagging with stumps in four out of five models. Also, WV with either Gini stumps or bagging with stumps is doing better than MV in all models.

We expand the methods that are mentioned in this chapter to deal with two-level trees as the following two subsections:

5.5.1 Bagging with Two-level Trees

This method is similar to bagging with stumps which is explained previously in Algorithm 2 but with two levels of splitting. `rpart` command is used here to grow a tree of two levels on different bootstrap sample size n^* and the settings of `rpart.control` parameters used here are: `rpart.control(maxdepth =2,`

`minsplit=1, cp = 0)`. The training data size $n = 1000$, the testing data size $m = 1000$ and the number of trees $B = 500$.

Table 5.9 shows the performance accuracy of bagging with two-level trees with the five models. This method with all the models gives very accurate results that are close to the maximal accuracy of each model. We can see that the largest number of averages of percentages happen at the $n^* = 1000$, i.e. the whole training data with replacement, apart from model 4. WV aggregation method performs better than MV with all cases. Bagging with two-level trees gives more accurate results than bagging with stumps method as in Table 5.4.

Model	Sample size n^*	Bagging with two-level trees		Maximal accuracy
		MV	WV	
1	20	94.10	94.09	94.21
	40	94.11	94.13	
	60	94.12	94.14	
	80	94.13	94.15	
	100	94.13	94.14	
	200	94.14	94.10	
	1000	94.15	94.20	
2	20	89.63	89.68	92.87
	40	90.66	90.65	
	60	90.61	90.68	
	80	90.66	90.65	
	100	90.66	90.81	
	200	91.23	91.28	
	1000	91.45	91.65	
3	20	83.17	83.06	83.51
	40	83.18	83.17	
	60	83.16	83.17	
	80	83.12	83.15	
	100	83.06	83.21	
	200	83.00	83.14	
	200	83.10	83.23	
	1000	83.04	83.40	
4	20	86.89	87.10	87.37
	40	82.49	87.02	
	60	75.01	86.83	
	80	73.91	86.58	
	100	73.92	84.89	
	200	73.77	82.90	
	1000	73.47	82.89	
5	20	78.38	78.40	78.51
	40	78.31	78.26	
	60	78.21	78.39	
	80	78.13	78.37	
	100	77.58	78.38	
	200	77.98	78.17	
	1000	77.80	78.40	

Table 5.9: The average of percentages of correctly predicted classes of 50 simulations from five models by using bagging with two-level trees method and the results in red are the largest results out of all results.

5.5.2 Gini-sampled Two-level Trees

Gini stumps has 2 terminal nodes since it divides the data at the root node just one time. We add another level of split at each node to have a two-level tree. This tree has 4 terminal nodes. The simple explanation of this generalisation is apply Algorithm 3 at the root node then at the left and right nodes again, see Algorithm 7 for more details:

Algorithm 7 Gini-sampled Two-level Trees

1: Apply Algorithm 3
2: Repeat Algorithm 3 on $\mathcal{L}$ and $\mathcal{R}$ nodes, then we will have four terminal nodes $\mathcal{LL}$, $\mathcal{LR}$, $\mathcal{RL}$ and $\mathcal{RR}$ nodes

Table 5.10 shows the accuracy of Two-levels Gini-sampled trees on the five models. By looking at the table, we can see that the method with WV is doing better than MV. Comparing Gini-sampled two-level tree with WV's results to the maximal accuracy, we find that this method with the five models gives results that are close to the maximal accuracy.

Model	1	2	3	4	5
MV	94.01	89.35	83.15	85.66	76.86
WV	94.11	92.23	83.21	87.27	78.20

Table 5.10: The average of percentages of correctly predicted classes of 50 simulations from five models by using Gini-sampled Two-level trees method.

To compare between Gini-sampled two-level trees results as in Table 5.10 and Gini-sampled stumps results at $\kappa = 1$ as in Table 5.5, we find that Gini-sampled two-level trees method performs better than Gini-sampled stumps method with all the five models regardless the aggregation method. Also, to compare between Gini-sampled two-level trees results as in Table 5.10 and bagging with two-level trees results as in Table 5.9 at sample size 1000, we find that Gini-sampled two-level trees is doing better than bagging with two-level trees method in model 2 and 4, and both methods have very similar performance in the rest of the models.

Chapter 6

Forests of Stumps with Real Datasets

The two methods of Gini stumps (Gini-sampled stumps and Gini-midpoints stumps) and bagging with stumps method have been applied to real datasets to measure their performance accuracy and the results are presented in this chapter. Section 6.1 indicates a summary of some real world datasets and the results of applying the two forests of stumps on these datasets to measure the accuracy of the methods. One of the real datasets results gives a different result when applying Gini stumps methods, which is Vertebral dataset. In Section 6.2, we will investigate the reasons behind this result. There are two potential reasons why this dataset performs differently with Gini stumps methods which are: the way Gini stumps method selects the split variables and the second reason is related to the nature of the dataset itself. We run an experiment to study many simulations and apply the two methods to measure their performance in Subsection 6.2.2.

6.1 Investigation of the Performance Accuracy of the Two Forests of Stumps with Real Datasets

In this section, we measure the performance accuracy of the forests of stump methods that are mentioned in Table 5.3. We will introduce the real datasets and the results of the investigation in the next two subsections.

6.1.1 Real Datasets

The datasets can be found on the UC Irvine Machine Learning Repository website (44) and the links below provide direct access to the dataset webpages. These datasets have multiple explanatory real-valued variables and a binary response variable. Table 6.1 shows the number of variables p, number of observations n, and a description of each dataset.

In order to use cross validation, the datasets are divided into K-folds, where $K = 10$, to estimate the average of the percentages of the correctly predicted classes by using the previous mentioned methods. Table 6.2 shows roughly numbers of observations in the training data after applying the cross validation method.

6.1.2 Results of the Real Datasets

Tables 6.3 and 6.4 indicate the percentages of the performance accuracy of the methods (two Gini stumps methods and bagging with stumps method) with the real datasets. We can see from these two tables that: Gini stumps methods perform better than bagging with stumps in Banknotes dataset, by a difference is around 1%. Bagging with stumps method performs better in Vertebral dataset,

[1] https://archive.ics.uci.edu/ml/datasets/banknote+authentication
[2] http://archive.ics.uci.edu/ml/datasets/vertebral+column
[3] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[4] https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival
[5] https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center

Data	p	n	Description
Bank[1]	4	1372	data was obtained from images that were taken to evaluate the authentication procedure of banknotes
Vertebral[2]	10	310	dataset to classify orthopaedic patients into normal and abnormal
Cancer[3]	31	569	Breast Cancer Wisconsin (Diagnostic) data set
Survival[4]	3	306	data set has cases from a study conducted on the survival of patients who had undergone surgery for breast cancer
Blood[5]	4	748	data taken from the blood transfusion service center in Hsin-Chu city in Taiwan

Table 6.1: Real datasets description.

Dataset	Bank	Vertebral	Cancer	Survival	Blood
n	1234	279	512	244	673

Table 6.2: Approximate numbers of observations used in the training stage with $K = 10$-fold cross validation.

by around 9%, Cancer dataset, by around 3% and Survival dataset, by 1%. All three methods give the same accuracy for blood data.

By comparing between the two aggregation methods, WV aggregation method gives higher results in most cases. However, when each row of the results in Table 6.3 are tested by using McNemar's test, there is no significant differences between the two Gini stumps method accuracies. McNemar's test is a non-parametric test used on paired nominal data and is applied to a 2×2 contingency table to determine whether the row and column marginal frequencies are equal or not. In Table 6.3, each row makes a 2×2 contingency table where the columns show the Gini stumps methods and rows show the aggregation methods.

Data	Gini-sampled Stumps		Gini-midpoints Stumps	
	MV	WV	MV	WV
Banknotes	86.45	85.94	83.68	85.06
Vertebral	67.74	67.74	68.06	70.32
Cancer	86.11	88.40	89.63	89.98
Survival	73.59	73.59	73.59	73.59
Blood	76.20	76.20	76.20	76.20

Table 6.3: The average of 10 accuracy percentages of correctly predicted classes from the real datasets, as in Table 6.1, by using Gini stumps methods. The results in the red colour represent the largest results for each data and the last two rows are tied.

Model	Sample size	Bagging with stumps	
	n^*	MV	WV
Banknotes	40	84.84	85.20
	60	84.77	85.06
	80	84.84	84.84
	100	84.76	84.98
Vertebral	40	78.06	79.03
	60	78.06	78.39
	80	78.39	77.10
	100	77.74	77.10
Cancer	40	92.80	92.62
	60	92.97	93.15
	80	92.97	93.15
	100	92.45	92.62
Survival	40	74.19	72.27
	60	74.19	72.58
	80	74.19	73.24
	100	74.19	71.91
Blood	40	76.20	76.20
	60	76.20	76.20
	80	76.20	76.20
	100	76.20	76.20

Table 6.4: The average of 10 accuracy percentages of correctly predicted classes from the real datasets, as in Table 6.1, by using bagging with stumps method. The results in the red colour represent the largest results for each dataset.

Due to the weak performance of Gini stumps methods in especially Vertebral dataset, we will investigate this data in the next section.

6.2 Investigating Vertebral Dataset Performance with Gini-midpoints Stumps Method

As we can see from the previous section Vertebral dataset has an inaccurate performance when it is used with Gini methods generally. There are two potential reasons for this performance, either the way Gini stumps methods select the variables, which is randomly from all variables based on sampling variables with respect to Gini indices, or there are some datasets perform poorly with Gini methods.

Here, we will focus on Gini-midpoints stumps method because it shows a better performance than Gini-sampled stumps method in case of Vertebral dataset. We will compare between the selected variables by using Gini-midpoints stumps and bagging with stumps methods as in Subsection 6.2.1. The other reason is Gini stumps methods perform inaccurately with some datasets and we will investigate this in Subsection 6.2.2.

6.2.1 The Selected Variables by Using Bagging with Stumps and Gini-midpoints Stumps

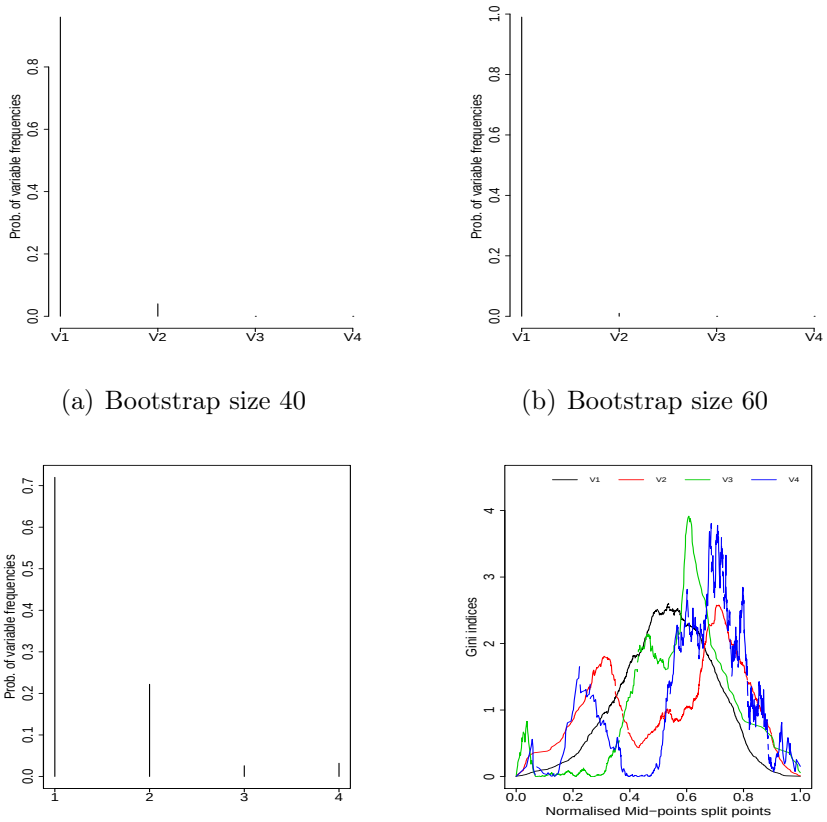
In order to investigate the reason these two methods perform differently, we will study how they choose variables because some variables are more informative than others, i.e. the variable that decreases most the node impurity, (Page 319, (45)). We will run an experiment to study the way the two methods choose variables on all datasets.

Bagging with stumps and Gini-midpoints stumps methods are run $B = 500$ times on the training datasets, then the split variables are taken from each run and method separately. These runs are independent in both methods.

Each figure from Figures 6.1 to 6.5 shows in the first two subfigures the selected variables from bagging with stumps method with two sample sizes 40 and 60, the third subfigure shows the selected variable from Gini-midpoints method and the last subfigure shows the normalised Gini curves of the variables. Due to the differences in these curves' values, where some of the curves have very small values and others have large values and the small values are not shown clearly on the plots, these curves are normalised in a way that the area under each curve is equal to 1.

As can be seen from Figures 6.1 to 6.5 bagging with stumps method sometimes chooses the variable that has the largest Gini gain as in Figure 6.2 and 6.5 but other times chooses variables do not have the largest Gini value as in Figure 6.1, 6.3 and 6.4. Figures 6.4(d) and 6.5(d) show discrete curves because they have discrete values.

According to this experiment, bagging with stumps way of choosing the variable to split at in these datasets, is not always at the variable which has the highest Gini gain. Because of this, we cannot say that the reason of the poor performance by Gini-midpoints stumps method is because of its way of selecting the variables. Regardless, how bagging with stumps and Gini-midpoints stumps methods choose variable to split at, we only have a problem with Vertebral dataset and due to these differences in choosing the split variables, this experiment does not explain the inaccurate performance by Gini stumps methods for this specific dataset.



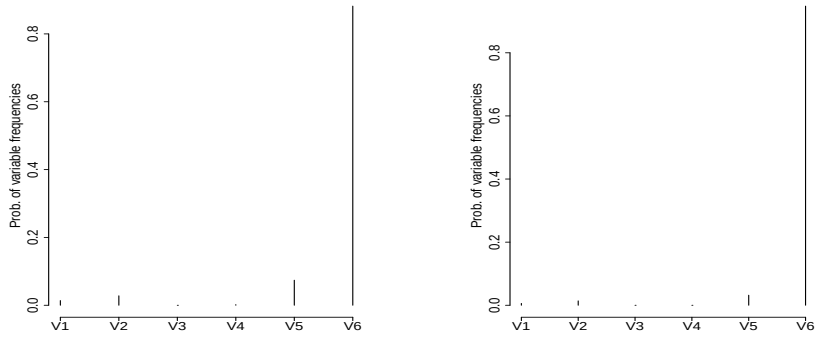
(a) Bootstrap size 40

(b) Bootstrap size 60

(c) Gini-midpoints stumps

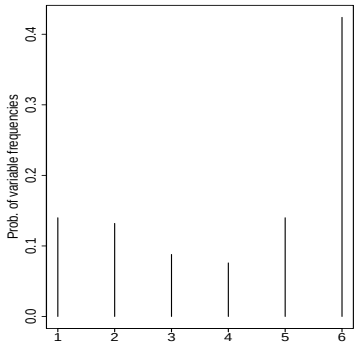
(d) Gini curves

Figure 6.1: Selected variables by bagging with stumps with two bootstrap size and Gini-midpoints stumps from the Banknote dataset. The last plot shows normalised Gini curves. The variable that has the largest Gini value is variable 3 and yet this variable is not the most selected variable by both methods.

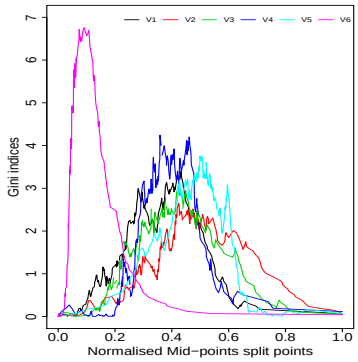


(a) Bootstrap size 40

(b) Bootstrap size 60

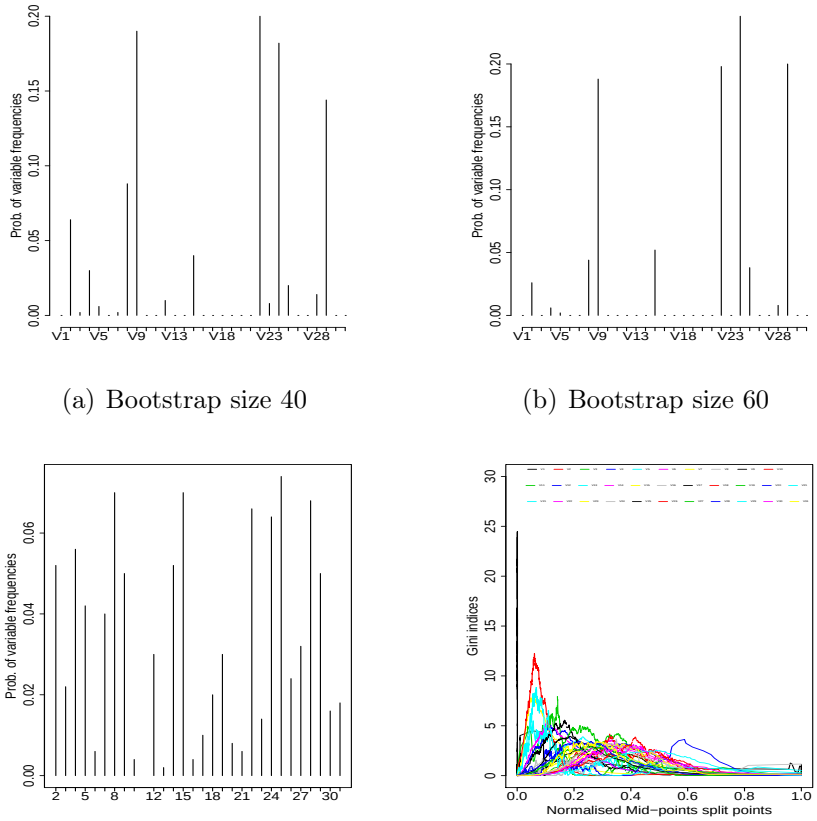


(c) Gini-midpoints stumps



(d) Gini curves

Figure 6.2: Selected variables by bagging with stumps with two bootstrap size and Gini-midpoints stumps from the Vertebral dataset. The last plot shows normalised Gini curves. Variable 6 has the largest Gini value and it is also the most selected variable by both methods.



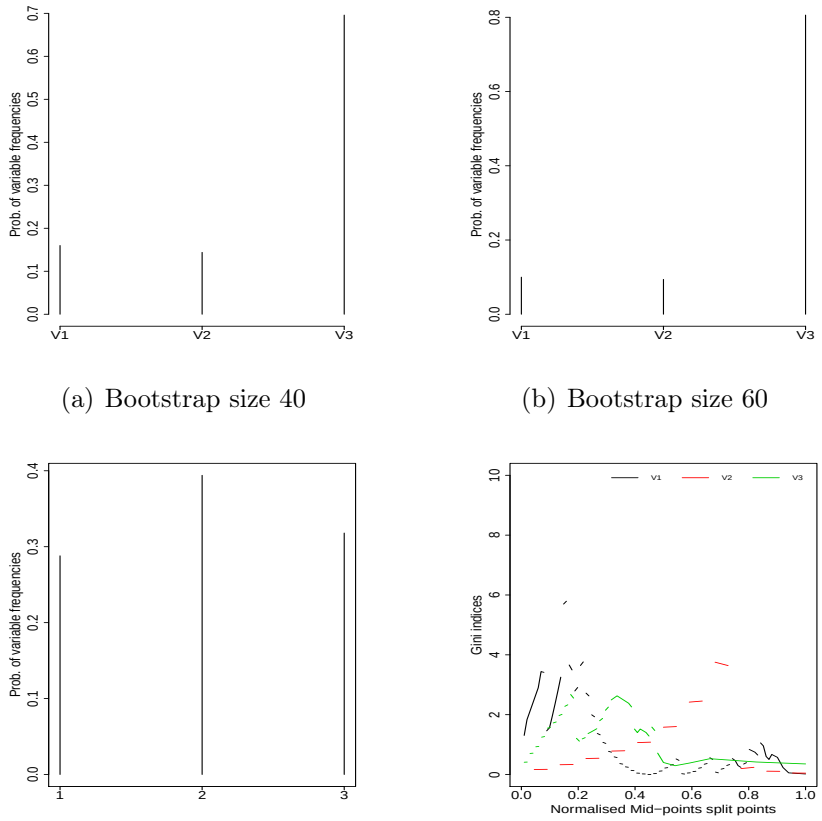
(a) Bootstrap size 40

(b) Bootstrap size 60

(c) Gini-midpoints stumps

(d) Gini curves

Figure 6.3: Selected variables by bagging with stumps with two bootstrap size and Gini-midpoints stumps from the Cancer dataset. The last plot shows normalised Gini curves.



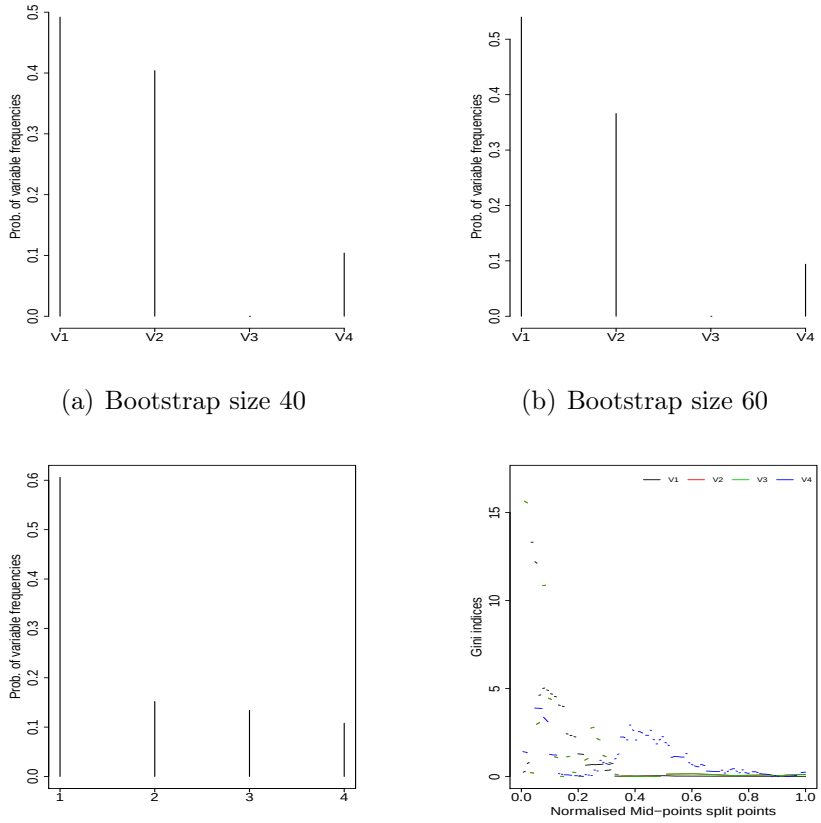
(a) Bootstrap size 40

(b) Bootstrap size 60

(c) Gini-midpoints stumps

(d) Gini curves (explanatory variables are discrete)

Figure 6.4: Selected variables by bagging with stumps with two bootstrap size and Gini-midpoints stumps from the Survival dataset. The last plot shows normalised Gini curves.



(a) Bootstrap size 40

(b) Bootstrap size 60

(c) Gini-midpoints stumps

(d) Gini curves (explanatory variables are discrete)

Figure 6.5: Selected variables by bagging with stumps with two bootstrap size and Gini-midpoints stumps from the Blood dataset. The last plot shows normalised Gini curves.

6.2.2 Two Variables Experiment

Another potential reason of the poor performance is some datasets do not perform as accurately with Gini methods as they do with bagging with stumps method. To investigate this hypothesis further, we will generate different simulations where they have two explanatory variables with different parameter settings. Figure 6.2 shows that the most selected variables are variable 5 and 6 and here we will focus more on simulations with two explanatory variables.

The aim of this experiment is trying to find characteristics of datasets in which bagging with stumps method does better than Gini-midpoints stumps method.

This experiment has two parts:

1. First part is to generate a group of datasets where the performance of the two methods is determined
2. Second part is to obtain the datasets which achieve the condition: bagging with stumps method performs better than Gini-midpoints stump method. Then, we will have a subgroup that contains all the datasets that bagging with stump method performs better than Gini-midpoints stump method. We generate datasets from this subgroup with applying small changes to the subgroup underlying distribution.

We will start this experiment by creating a simulation with two explanatory variables and a binary response variable. The setting of this simulation is two variables that are a bivariate normal distributed as follows:
$N(\mu_j, \Sigma_j)$ where $j = \{1, 2\}$

- $\mu_j = \begin{bmatrix} \mu_{j1} \\ \mu_{j2} \end{bmatrix}$, where $\mu_{1j} = 0$ and $\mu_{2j} \sim U(0, 3)$
- $\Sigma_j = \begin{bmatrix} \sigma_{j1}^2 & \rho_j \sigma_{j1} \sigma_{j2} \\ \rho_j \sigma_{j1} \sigma_{j2} & \sigma_{j2}^2 \end{bmatrix}$, where $\sigma_{1j} = 1$, $\sigma_{2j} \sim U(0, 2)$ and $\rho_j \in U(-1, 1)$

Note: we choose $\mu_{1j} = 0$ and $\sigma_{1j} = 1$ as fixed values, and change μ_{2j} and σ_{2j} values because decision trees are not affected by monotonic transformation and choosing variables in this way gives us more understanding of the variables that are used

to specify a region of the space where its datasets perform more accurately with bagging with stumps.

Experiment Steps

We will do this experiment in two parts as follows:

- **First Part:**

1. Generate a single realisation of $\mu_j, \Sigma_j, \rho_j, j = \{1, 2\}$ according to the settings above
2. Use these values as parameters to generate a dataset that has $n = 1000$ observations
3. Apply the following two methods $B = 500$ times on the generated data from Step 2
   - Bagging with stumps and sample size $n^* = 20$
   - Gini-midpoints stumps
4. Find the proportion of the observations that have got their classes correctly predicted from both methods listed in Step 3
5. Save the parameters that are generated in Step 1 and the performance accuracies from Step 4
6. Repeat from Step 1 to Step 5, 4500 times and save the results into a data frame has the following columns:
   Diff, ρ_1, ρ_2, μ_{21}, μ_{22}, σ_{21}, σ_{22}, $\frac{\mu_{21}}{\sigma_{21}}$, $\frac{\mu_{22}}{\sigma_{22}}$, $\rho_1\rho_2$, $\frac{\sigma_{21}^2+\sigma_{22}^2}{2}$

Note that:

- Diff is subtracting the proportion of bagging with stumps accuracy from the proportion of Gini-midpoints stumps accuracy. If the result is greater than 0, then Gini is winning and it is coded as 1, it is coded as 2 if the difference is less than 0 and finally the difference is coded as 3 if it is equal to 0. We will refer to this column as "method" after recoding
- the last four columns are made-up variables from existing variables; and they are showing the scale distances $\frac{\mu_{21}}{\sigma_{21}}$, $\frac{\mu_{22}}{\sigma_{22}}$, correlation multiplication $\rho_1\rho_2$ and average variances $\frac{\sigma_{21}^2+\sigma_{22}^2}{2}$

Now, we have a dataset D_1 that has 4500 observations, and 11 variables: method, ρ_1, ρ_2, μ_{21}, μ_{22}, σ_{21}, σ_{22}, $\frac{\mu_{21}}{\sigma_{21}}$, $\frac{\mu_{22}}{\sigma_{22}}$, $\rho_1\rho_2$, $\frac{\sigma_{21}^2+\sigma_{22}^2}{2}$. Each of the observations is derived from a dataset, the parameters of this dataset and a comparison of the performance of the two methods.

We will apply a classification `rpart`-tree on this dataset D_1 and treat "method" variable as the response variable. Figure 6.6 shows the classification decision tree by using dataset D_1. As we can see, this tree shows the Gini-midpoints stumps method is doing better in a large percentage of the observations (datasets), however, this is not clear enough because we have a lot of observations with equal or very close performance. Due to this, Diff variable values that belong to the interval $(-0.02, 0.02)$ are excluded in Step 6 to focus more on datasets that have greater differences in the performance and again this variable is recoded and used to fit a classification tree which is shown in Figure 6.7.

Figure 6.7 shows tree with blue nodes that are allocated to Gini-midpoints stumps method and green nodes represent bagging with stumps method. It is clearly shown here that Gini-midpoints stump method performs better in most cases; which emphasise the reason why we need to focus more on finding a specific setting that can explain the reason of a large difference in the performance between the two methods in some datasets in favour of bagging with stumps method.

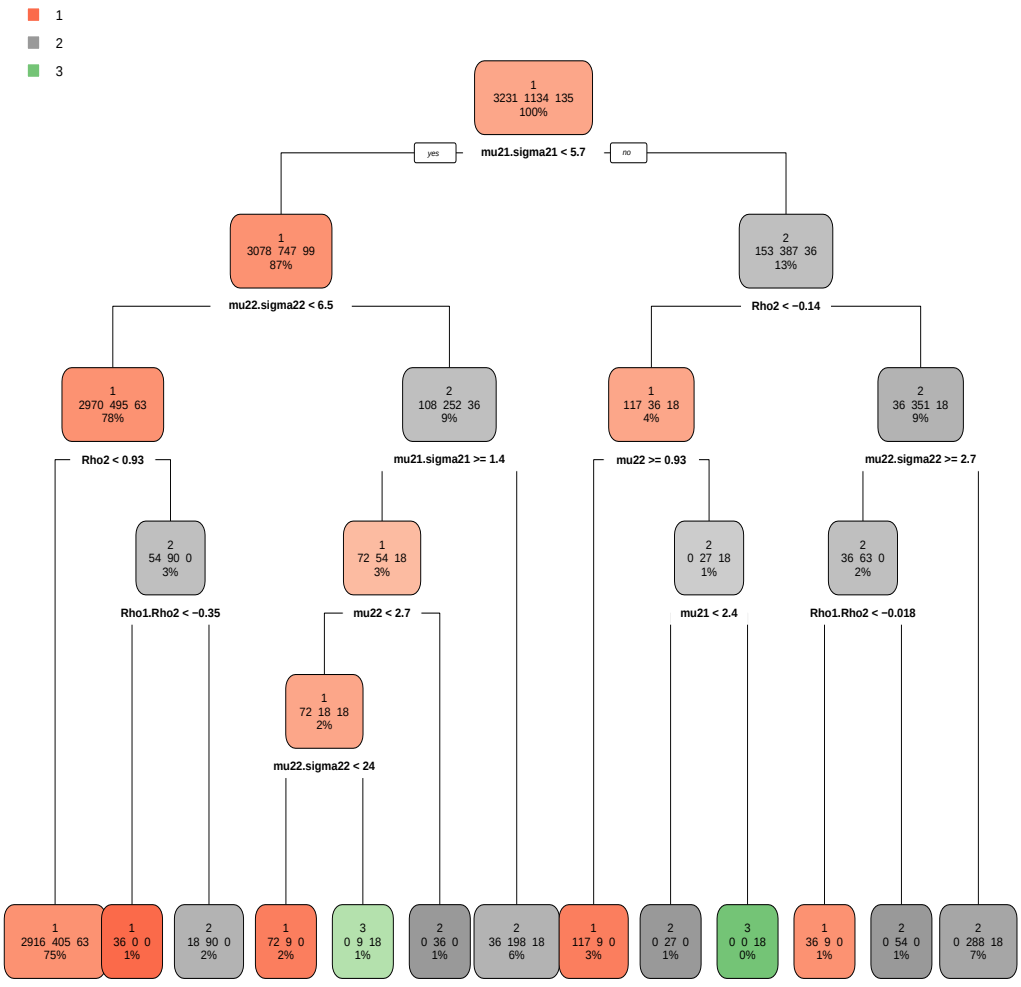


Figure 6.6: Classification tree shows the performance of Gini-midpoints stumps and bagging with stumps methods. Here $\mu_{21}\sigma_{21} \equiv \frac{\mu_{21}}{\sigma_{21}}$, $\mu_{22}\sigma_{22} \equiv \frac{\mu_{22}}{\sigma_{22}}$.

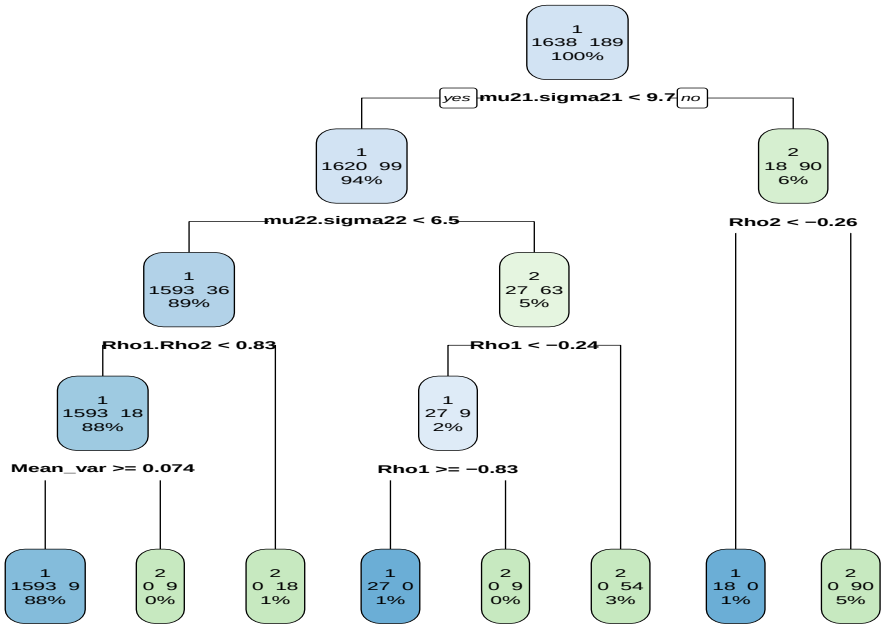


Figure 6.7: Classification tree shows the performance of Gini-midpoints stumps and bagging with stumps methods after excluding observations which have a difference between the two methods' performance which belongs to the interval $(-0.02, 0.02)$, i.e. Diff $\notin (-0.02, 0.02)$. Blue nodes are allocated to Gini-midpoints stumps method and green nodes represent bagging with stumps method.

- **Second Part**

We still do not have a specific area that shows a better performance in just bagging with stumps method, therefore, we will choose observations from data D_1 that satisfy one condition which is bagging with stumps performance is greater than Gini-midpoints stumps performance.

By doing this, we have 189 observations (datasets) out of the 2673 and refer to this dataset as D^*. Then, generate a new data as follows:

1. Sample from the rows of the data D^*
2. Add a small random noise to each parameter to make it a little different to the original one but not out of the range of the suggested intervals in Page 140
3. Use these values as parameters to generate a dataset that has $n = 1000$ observations
4. Use the dataset in Step 3 to fit $B = 500$ stumps from the two methods (bagging with stumps and Gini-midpoints stumps methods), then add this point to data D^* with its parameters and the performance accuracy, as before
5. Repeat 2000 times from Step 1 to Step 4
6. Dispose of the first 189 observations that are used to generate this data and refer to the dataset after the disposal as D_2, that is to make sure D_2 is not biased towards bagging with stumps method

We now have this new dataset D_2 that has $n = 2000$ observations and 8 variables. We take the first variable p_1 which is the correctly classified proportion of the performance of bagging with stumps method and the second one p_2 is the correctly classified proportion of Gini-midpoints stumps method performance and find the $\log \frac{p_1}{p_2}$, then code the resulted variable as follows: $\begin{cases} 1, & \log \frac{p_1}{p_2} < 0 \\ 2, & \log \frac{p_1}{p_2} > 0 \\ 3, & \log \frac{p_1}{p_2} = 0 \end{cases}$.

Note: we take the logarithms of the ratio which is equal to $\log p_1 - \log p_2$, and we know that log difference is almost the same number as the percentage change.

Figure 6.8 shows the classification tree from this dataset D_2. Dataset D_2 has $n = 2000$ observations, 106 observations belong to C_1, 1860 observations belong to C_2, and 34 observations belong to C_3.

Green nodes represent the areas where bagging with stumps method is doing better than Gini-midpoints stumps method. We can see from these green nodes that there are some regions which have datasets that gives more accurate performance with bagging with stumps method. The variable $\mu_{21} < 0.36$ produces a green node that has the largest percentage of observations in class C_2, which is 94% of the total number of observation in that class, i.e. $\frac{1742}{1860} \approx 0.94$. The variable μ_{21} has values in the range between 0 and 3 but only values that are greater than 0 and less than 0.36 produce observations in class C_2.

To verify this conclusion, we will generate a testing dataset which has the same underlying distribution as D_2. This testing dataset is generating by using D^* the same way used to generate D_2. The testing dataset D_t has $n = 2000$ observations (datasets), 91 observations belong to C_1, 1874 observations belong to C_2 and 35 observations belong to C_3. The accuracy of prediction on the testing dataset is 93.65% when comparing the actual and predicted classes of the testing dataset. We can conclude from this experiment that there are some datasets perform better with certain algorithms as we can see that bagging with stumps method has better performance than Gini stumps methods with some particular datasets.

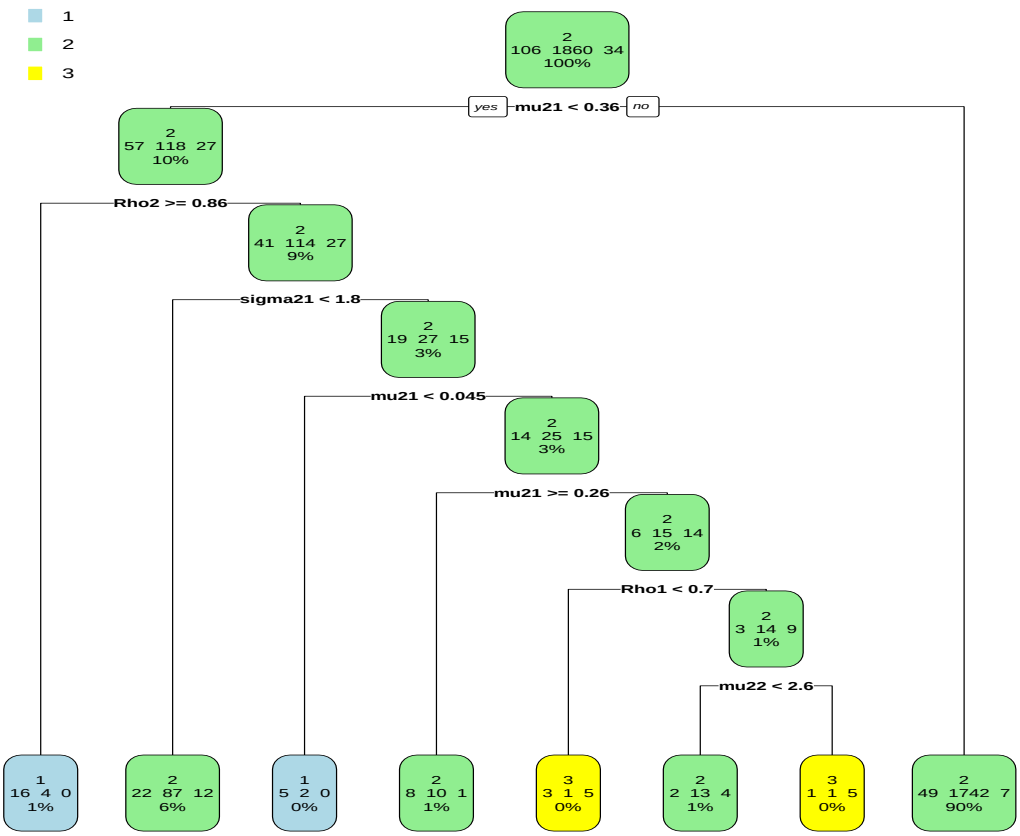


Figure 6.8: Classification tree fitted on D_2 dataset shows the performance of Gini-midpoints stumps and bagging with stumps methods. Class 1 shows where Gini-midpoints stumps method is doing better than bagging with stumps and class 2 is where bagging with stumps method performs more accurately than Gini-midpoints stumps, and class 3 where the performance of the two methods is equal.

Chapter 7

Boosting Classification Stumps

Boosting is a powerful and popular ensemble algorithm. There are variants of boosting algorithms that differ in the choice of the base learner and the criterion used to reweight observations such as: AdaBoost (Adaptive boosting) (40) which has two algorithms (discrete AdaBoost and real AdaBoost); Gradient boosting and XGBoost (stands for eXtreme Gradient boosting) (Chapter 3, (46)). Boosting combines a sequence of weak learners, each of them compensates the weaknesses of its predecessors which leads to an accurate prediction of the full model (Page 23, (47)). In this chapter, we will focus on AdaBoost (discrete and real) with Gini stumps methods as our base learner. Section 7.1 introduces a brief summary of both AdaBoost techniques with examples to show how the weights are adapted in both methods. Discrete AdaBoost adapts the weights of the observations based on prediction errors whereas real AdaBoost uses the predicted class probabilities for adapting the weight. In Section 7.2, we try to find the expected value of weighted Gini theoretically, however, we stop due to reaching a very complicated integral. Weighted Gini stumps methods are explained in Section 7.3, then these methods are applied to simulations and real datasets. A comparison between bagging and boosting accuracies is carried out in Section 7.4. Finally, Section 7.5 shows a summary of the chapter.

7.1 AdaBoost

The AdaBoost algorithm was the first practical boosting algorithm, and still one of the most widely used, with applications in many fields (40). This section aims to combine stumps with AdaBoost and measure the accuracy of Gini stumps methods when combined with AdaBoost algorithms.

7.1.1 Discrete AdaBoost

Discrete AdaBoost reweights incorrectly classified observations by increasing their weights, so that the next classifier will focus more on these observations (48). Suppose we have a training dataset whose structure was as previously described in Chapter 2, y_i to denote the label of the i^{th} observation such that $y_i \in \{-1, 1\}$. Here, we have n observations with initial equal weight $w_i = \frac{1}{n}, i = 1, 2, \ldots, n$. We have a sequence of classifiers $d_q(x)$ which predicts values -1 or 1 on each iteration q, where, $q = 1, \ldots, \mathcal{Q}$. The weight w_i is updated for each observation at each iteration. The last step is to output a weighted vote of the classifiers by multiplying by γ_q which is the weight of the classifier $d_q(x)$. Algorithm 8 gives more details of this method.

Algorithm 8 The discrete AdaBoost (40).

1. Start with weights $w_i = \frac{1}{n}, i = 1, 2, \ldots, n$
   For$(q \leftarrow 1 \quad to \quad \mathcal{Q})$
   (a) Apply weight w_i on x_i in the training data and fit the classifier $d_q(x) \in \{-1, 1\}$
   (b) Compute $\epsilon_q = \frac{\sum_{i=1}^{n} w_i I(y_i \neq d_q(x_i))}{\sum_{i=1}^{n} w_i}$
   (c) Compute $\gamma_q = \frac{1}{2}\log(\frac{1-\epsilon_q}{\epsilon_q})$
   (d) Update $w_i = w_i \exp\left[\gamma_q I(y_i \neq d_q(x_i))\right]$, and renormalise the weights by dividing by $\sum_i w_i$
2. Output the classifier: $\text{sign}\left[\sum_{q=1}^{\mathcal{Q}} \gamma_q d_q(x)\right]$

7.1.2 Real AdaBoost

A generalised version of discrete AdaBoost is called real AdaBoost as in Algorithm 9. Both methods are applied to a binary classifier, however, real AdaBoost weights based on predicted class probabilities whereas discrete uses the errors in predicted class labels (49).

The first step from Algorithm 9 is the same as discrete AdaBoost. Then, it fits a classifier and obtains a class probability by using the weight of training dataset observations. Following this, each leaf node is updated to output half the value of the logit transform of its previous value instead of multiplying the whole classifier by a fixed value γ_q, as in case of discrete AdaBoost, due to the fact that this method depends on the probabilities of classes not on the labels. Finally, the weight is updated for all observations.

Algorithm 9 The real AdaBoost (40).
$\hat{P}_w$ represents class probability estimate.

1. Start with weights $w_i = \frac{1}{n}, i = 1, 2, \ldots, n$
   For$(q \leftarrow 1 \quad to \quad \mathcal{Q})$
   (a) Apply weight w_i on x_i in the training data and fit the classifier to obtain a class probability estimate $P_q(x) = \hat{P}_w\big(y = 1|x\big)$
   (b) Set $h_q(x) = \frac{1}{2}\log\left(\frac{P_q(x)}{1-P_q(x)}\right) \in \mathbb{R}$
   (c) Update $w_i = w_i \exp\left[-y_i\, h_q(x_i)\right]$, and renormalise the weights by dividing by $\sum_i w_i$.
2. Output the classifier: $\text{sign}\big[\sum_{q=1}^{\mathcal{Q}} h_q(x)\big]$

Example 9

We will track the weight of specific observations to see how the weights behave in each iteration with discrete and real AdaBoost. We have model 2 from Subsection 4.3.2 and the number of observations is $n = 1000$. This model has one explanatory variable and this variable has *pdf* which appears in Figure 7.1 with a binary response variable. We will track weight of four observations to observe how it is changing in

every iteration. These four observations have different locations and classes, as in Figure 7.1. A class probability estimate $P_q(x)$ is found by finding the prediction probability of $y = 1$, i.e. the prediction probability of having a class equal to 1.

Figure 7.2 indicates the way that the weight is changing for the observations after applying decision stumps with AdaBoost in each iteration for each of them. Figures 7.2(a) and 7.2(b) show discrete and real AdaBoost, respectively.

From Figure 7.2(a), observations 2 and 4 are more likely to be misclassified because of their locations, which are in the overlapping areas, especially observation 2. One can see from Figure 7.2(a) that the observation weights increased (boosted) if the locations of these observations are in the overlapping areas, i.e. observations 2 and 4; and the weight of the other two observations 1 and 3 are decreased.

From Figure 7.2(b), the weight of observations 3 and 4 decreases, while the weight of observations 1 increases and weight of observation 2 decreases, then increases at the same point when weight of observation 1 decreases. The changing in weight in real AdaBoost is slow. Figure 7.2(c) shows the changing in the weight with real AdaBoost until the 200^{th} iteration. In Figure 7.2(c), observation 2 reaches around 0.0013 weight after 200 iterations with real AdaBoost, while it reaches around 0.007 in Figure 7.2(a) with discrete AdaBoost after 50 iterations.

Figures 7.3 and 7.4 show the observations classes after each updates for the first 50 weight updates with discrete and real AdaBoost. We can notice from Figures 7.3(a) and 7.4(a) how the homogeneity of a node is improved after the first iteration, i.e. first and second horizontal lines.

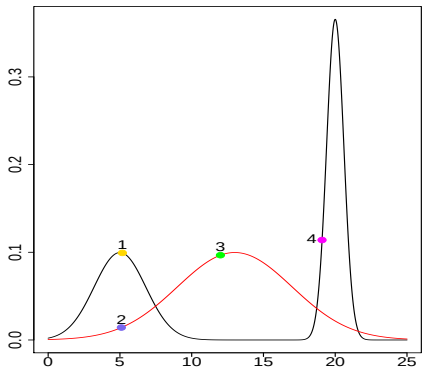


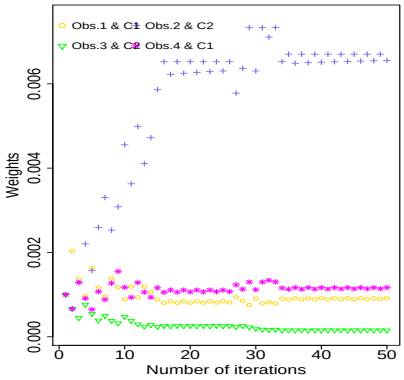
Figure 7.1: The location and classes of four observations from model 2.

Example 10

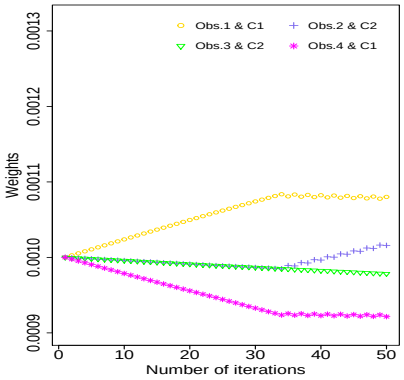
We will measure the way discrete and real AdaBoost algorithms with stumps perform. We use `rpart` command to grow a forest of stumps with either discrete or real AdaBoost algorithms where an `rpart`-stump replaces the classifier $d_q(x)$ or $P_q(x)$ in Algorithm 8 or Algorithm 9, respectively. The settings of `rpart.control` used parameters are as follows: `maxdepth=1, cp = 0, minsplit = 1`, and changing observations' weights is done by using `weight` parameter.

We will apply these methods to the five models mentioned in Subsection 4.3.2. Training and testing datasets have 1000 observations each.

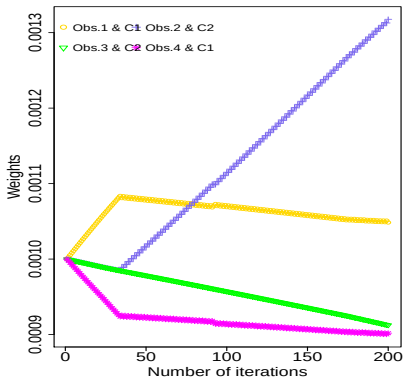
Table 7.1 shows the accuracy of stumps with AdaBoost algorithms. We have 50 simulations from each model of the five models and the number of weight updates is $\mathcal{Q} = 100$ in each simulation. Stumps perform almost the same with both (real and discrete) AdaBoost algorithms and the accuracy performance of the five models is very close to the maximal model accuracies. However, the real AdaBoost performance is slightly less accurate than discrete AdaBoost performance for four of the five models.



(a) Discrete AdaBoost until the 50^{th} iteration.



(b) Real AdaBoost until the 50^{th} iteration.

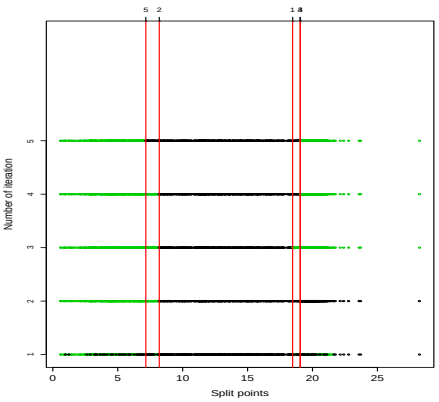


(c) Real AdaBoost until the 200^{th} iteration.

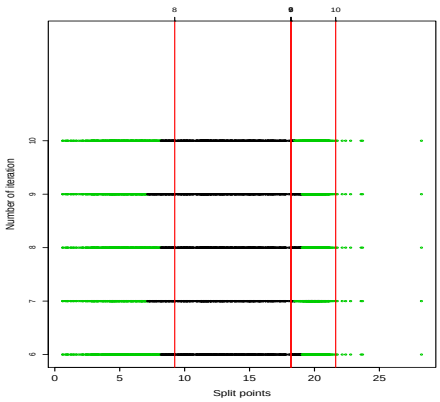
Figure 7.2: Changing in the weight for the four points mentioned in Figure 7.1 from model 2 for discrete and real AdaBoost.

Model	1	2	3	4	5
Discrete	93.90	89.95	82.81	86.65	77.67
Real	93.80	89.69	83.15	86.51	77.84
Maximal accuracy	94.21	92.87	83.51	87.37	78.51

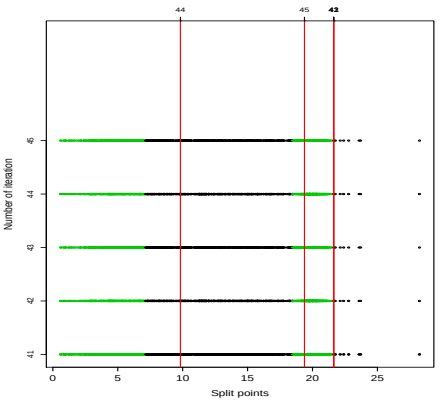
Table 7.1: The averages of percentages of correctly predicted classes of 50 simulations from five models by combining `rpart`-stumps with AdaBoost methods.



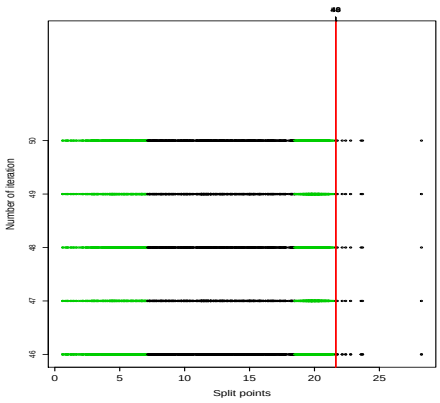
(a) Weight updates from 1 to 5 for the observations with the split points.



(b) Weight updates from 6 to 10 for the observations with the split points.

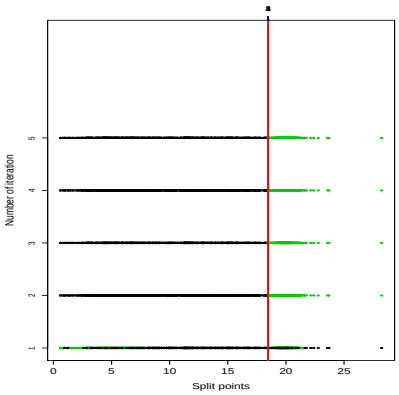


(c) Weight updates from 41 to 45 for the observations with the split points.

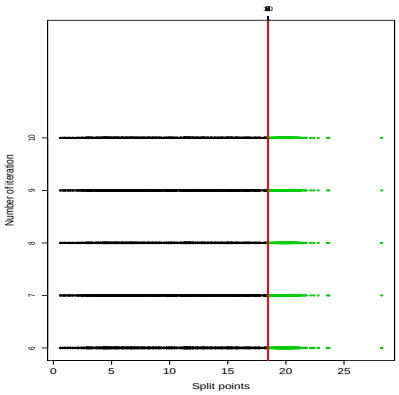


(d) Weight updates from 46 to 50 for the observations with the split points.

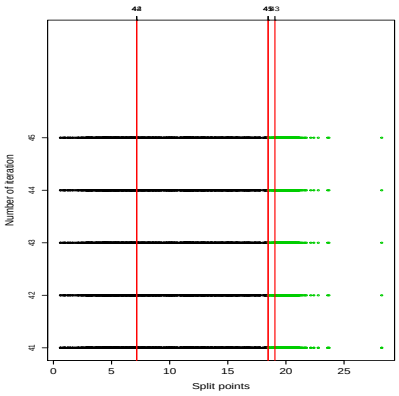
Figure 7.3: Observation classes for the first and last ten weight updates with discrete AdaBoost. The red lines represent the split points, and the green and black points show the classification partition.



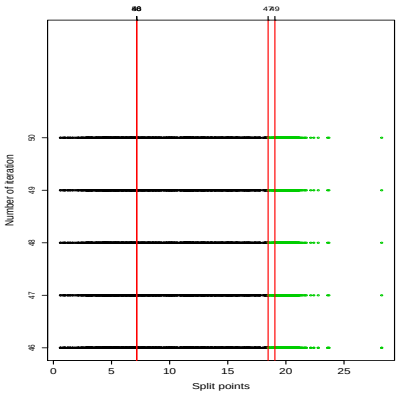
(a) Weight updates from 1 to 5 for the observations with the split points.



(b) Weight updates from 6 to 10 for the observations with the split points.



(c) Weight updates from 41 to 45 for the observations with the split points.

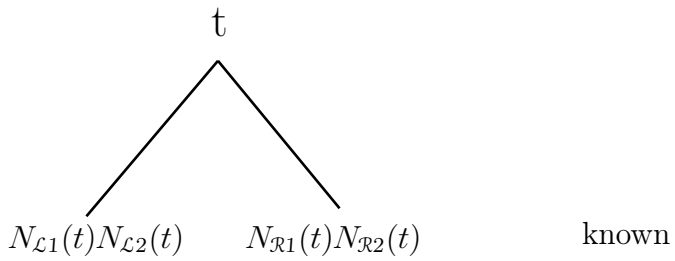


(d) Weight updates from 46 to 50 for the observations with the split points.

Figure 7.4: Observation classes for the first and last ten weight updates with real AdaBoost. The red lines represent the split points, and the green and black points show the classification partition.

7.2 The Expected Value of Weighted Gini

It is of interest to see how the expected value of the weighted Gini compares to the expected value of unweighted, when the weights are determined by the first split. This can provide further insights into the way that boosting works. The expected value of unweighted Gini is obtained in Section 3.3. The weight is updated by using discrete AdaBoost and the base learner is a Gini stump. We have two nodes left $\mathcal{L}$ and right $\mathcal{R}$, and a training data with a binary response variable, so we have two classes $J = \{1, 2\}$. The observation numbers in each class and node are: $N_{\mathcal{L}1}(t)$, $N_{\mathcal{L}2}(t)$, $N_{\mathcal{R}1}(t)$ and $N_{\mathcal{R}2}(t)$. Here, t is the first split point which is obtained randomly by sampling from the Gini function, i.e. the split point of the first Gini stump d_1 in Algorithm 8.



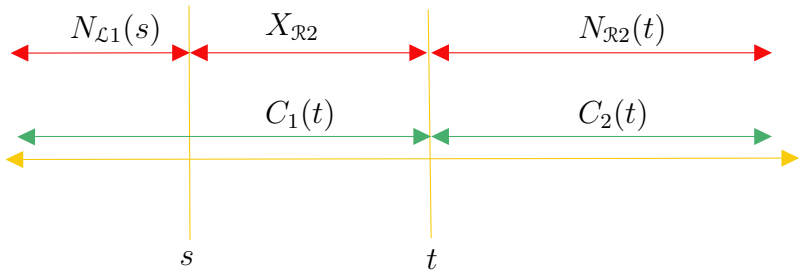
We initially suppose that t is given and these $N_{\mathcal{L}1}(t)$, $N_{\mathcal{L}2}(t)$, $N_{\mathcal{R}1}(t)$ and $N_{\mathcal{R}2}(t)$ are known and the weights are not random, where w_{cj} is the weight of an observation at node $c = \{\mathcal{L}, \mathcal{R}\}$ and class j. The summation of weight of observations in class j and node c is W_{cj} and the total weight of observations in both nodes and classes is $W_N = \sum_{c=\{\mathcal{L},\mathcal{R}\}} \sum_{j=1}^{2} W_{cj} = 1$, where N is the number of observations in just this chapter to avoid confusion with other notations.

Suppose we have

$$N_{\mathcal{L}1}(t) > N_{\mathcal{L}2}(t) \qquad N_{\mathcal{R}1}(t) < N_{\mathcal{R}2}(t).$$

So, the left node is allocated to class $C1$ and the right node to class $C2$.

The first case:
Suppose $s < t$, where s is the split point after the first weight adjust, i.e. the split point of the second Gini stump d_2 in Algorithm 8. The second split will depend on the weight from the first split.



The number of observations in the left node $\mathcal{L}$ and class C_1 is $N_{\mathcal{L}1}(s)$, and it is random and binomially distributed as follows $N_{\mathcal{L}1}(s) \sim Bin\left(N_{\mathcal{L}1}(t), \frac{F_1(s)}{F_1(t)}\right)$, where $F_j(t)$ and $F_j(s)$ are the cumulative distribution functions of class j for t and s, respectively.

To find the expected value of the change in Gini as in Equation (7.1), and for $j = 2$, $N_{\mathcal{L}} = N_{\mathcal{L}1} + N_{\mathcal{L}2}$ and $N_{\mathcal{R}} = N_{\mathcal{R}1} + N_{\mathcal{R}2}$, that was introduced on Page 31.

$$\triangle_\Gamma = \Gamma(\mathcal{P}) - \left(\frac{N_{\mathcal{L}}}{N}\Gamma(\mathcal{L}) + \frac{N_{\mathcal{R}}}{N}\Gamma(\mathcal{R})\right). \tag{7.1}$$

Then,

$$\left(\frac{N_{\mathcal{L}}}{N}\Gamma(\mathcal{L}) + \frac{N_{\mathcal{R}}}{N}\Gamma(\mathcal{R})\right) = \frac{2}{N}\left(N_{\mathcal{L}1} - \frac{N_{\mathcal{L}1}^2}{N_{\mathcal{L}}} + N_{\mathcal{R}1} - \frac{N_{\mathcal{R}1}^2}{N_{\mathcal{R}}}\right). \tag{7.2}$$

The weighted Gini index is given by

$$\Gamma_w(c) = 1 - \sum_{j=1}^{J}\left(\frac{W_{cj}}{W_c}\right)^2, \tag{7.3}$$

where $W_c = \sum_{j=1}^{J}\sum_{x_i \in c} w_i I(y_i = j)$ and for $J = 2$, the change in weighted Gini is given by Equation (7.4)

$$\triangle_{\Gamma_w} = \Gamma_w(\mathcal{P}) - \frac{2}{W_N}\left(W_{\mathcal{L}1} - \frac{W_{\mathcal{L}1}^2}{W_{\mathcal{L}}} + W_{\mathcal{R}1} - \frac{W_{\mathcal{R}1}^2}{W_{\mathcal{R}}}\right). \tag{7.4}$$

The misclassified Classes are C_2 in the left node and C_1 in the right node, so $\epsilon = \dfrac{N_{\mathcal{L}2}(t) + N_{\mathcal{R}1}(t)}{N} < \dfrac{1}{2}$, and $\beta = \dfrac{1}{2}\log\left(\dfrac{1-\epsilon}{\epsilon}\right)$ from the discrete AdaBoost algorithm as in Algorithm 8. Here, β represents γ_q in this Algorithm.

The error ϵ is chosen to be less than $\dfrac{1}{2}$ in order "to reduce the error of any base learning algorithm that generates classifiers whose performance is a little better than random guessing" (40). Therefore, as soon as $\epsilon > \dfrac{1}{2}$, the iteration stopped.

The weight of correctly classified observation is

$$w_{\mathcal{L}1} = w_{\mathcal{R}2} = \frac{e^{-\beta}}{W_N}.$$

The weight of misclassified observation is

$$w_{\mathcal{L}2} = w_{\mathcal{R}1} = \frac{e^{\beta}}{W_N},$$

and

$$e^{\beta} = \left(\frac{N - N_{\mathcal{L}2}(t) - N_{\mathcal{R}1}(t)}{N_{\mathcal{L}2}(t) + N_{\mathcal{R}1}(t)}\right)^{1/2}.$$

By taking the expected value of Equation (7.4) and considering $W_{\mathcal{L}}$, $W_{\mathcal{L}1}$, $W_{\mathcal{L}2}$, $W_{\mathcal{R}}$, $W_{\mathcal{R}1}$, and $W_{\mathcal{R}2}$ as random variables, we need to find $E(W_{\mathcal{L}1})$, $E\left(\dfrac{W_{\mathcal{L}1}^2}{W_{\mathcal{L}}}\right)$, $E(W_{\mathcal{R}1})$ and $E\left(\dfrac{W_{\mathcal{R}1}^2}{W_{\mathcal{R}}}\right)$.

The left node $\mathcal{L}$:

Since

$N_{\mathcal{L}1}(s) \sim Bin\ (n,\ p)\,;$ where $n = N_{\mathcal{L}1}(t)$ and $p = \dfrac{F_1(s)}{F_1(t)}$, and

$N_{\mathcal{L}2}(s) \sim Bin\ (m,\ q)\,;$ where $m = N_{\mathcal{L}2}(t)$ and $q = \dfrac{F_2(s)}{F_2(t)}$,

then,

- $E(W_{\mathcal{L}1}(s)) = E\big(w_{\mathcal{L}1}\, N_{\mathcal{L}1}(s)\big) = w_{\mathcal{L}1}\, E\big(N_{\mathcal{L}1}(s)\big) = \dfrac{e^{-\beta}}{W_N}\, n\, p.$

From the law of total expectation

- $$E\left(\frac{W^2_{\mathcal{L}1}(s)}{W_{\mathcal{L}}(s)}\right) = \sum_{\substack{N_{\mathcal{L}1}(s)=n_{\mathcal{L}1}(s)\\ N_{\mathcal{L}2}(s)=n_{\mathcal{L}2}(s)}} E\left(\frac{W^2_{\mathcal{L}1}(s)}{W_{\mathcal{L}}(s)}\,\middle|\, N_{\mathcal{L}1}(s)=n_{\mathcal{L}1}(s),\ N_{\mathcal{L}2}(s)=n_{\mathcal{L}2}(s)\right)\cdot Pr\big(N_{\mathcal{L}1}(s)=n_{\mathcal{L}1}(s),\ N_{\mathcal{L}2}(s)=n_{\mathcal{L}2}(s)\big)$$
$$= \underbrace{\sum_{\substack{n_{\mathcal{L}1}(s)\\ n_{\mathcal{L}2}(s)}} \frac{n^2_{\mathcal{L}1}(s)\frac{e^{-2\beta}}{W_N^2}}{n_{\mathcal{L}1}(s)\frac{e^{-\beta}}{W_N}+n_{\mathcal{L}2}(s)\frac{e^{\beta}}{W_N}}\ Pr\Big(N_{\mathcal{L}1}(s)=n_{\mathcal{L}1}(s)\Big)\cdot Pr\Big(N_{\mathcal{L}2}(s)=n_{\mathcal{L}2}(s)\Big).}_{E(g(x))=\sum g(x)Pr(X=x)}$$

Then,

$$E\left(\frac{W^2_{\mathcal{L}1}(s)}{W_{\mathcal{L}}(s)}\right) = E\left(\frac{N^2_{\mathcal{L}1}(s)\,\dfrac{e^{-2\beta}}{W_N^2}\div e^{\beta}}{N_{\mathcal{L}1}(s)\dfrac{e^{-\beta}}{\cancel{W_N}}\div e^{\beta}+N_{\mathcal{L}2}(s)\dfrac{e^{\beta}}{\cancel{W_N}}\div e^{\beta}}\right)$$
$$= \frac{1}{W_N}\,E\left(\frac{N^2_{\mathcal{L}1}(s)\,e^{-3\beta}}{N_{\mathcal{L}1}(s)\,e^{-2\beta}+N_{\mathcal{L}2}(s)}\right)$$
$$= \frac{e^{-3\beta}}{W_N}\,E\left(\frac{N^2_{\mathcal{L}1}(s)}{\alpha N_{\mathcal{L}1}(s)+N_{\mathcal{L}2}(s)}\right)\ ; \qquad \text{where}\ \ \alpha = e^{-2\beta}.$$

Now let $N_{\mathcal{L}1}(s) \equiv X$ and $N_{\mathcal{L}2}(s) \equiv Y$, so that

$$E\left(\frac{W^2_{\mathcal{L}1}(s)}{W_{\mathcal{L}}(s)}\right) = \frac{e^{-3\beta}}{W_N}\,E\left(\frac{X^2}{\alpha X+Y}\right). \tag{7.5}$$

By finding Taylor Series to the second order for $f(x,y) = \dfrac{X^2}{\alpha X + Y}$, we take the series around the mean of each random variable $(X^*,\, Y^*) \equiv (np,\, mq)$.

$$E(f(X,Y)) \approx E\Bigg[f(X^*,\, Y^*) + f_X(X^*,\, Y^*)(X - X^*) + f_Y(X^*,\, Y^*)(Y - Y^*) + \frac{1}{2!}\Big(f_{XX}(X^*,\, Y^*)(X - X^*)^2 + 2f_{XY}(X^*, Y^*)(X - X^*)(Y - Y^*) + f_{YY}(X^*,\, Y^*)(Y - Y^*)^2\Big)\Bigg].$$

Then, we need to find the following derivatives

$$f_X(x,y) = \frac{2x(\alpha x + y) - \alpha x^2}{(\alpha x + y)^2} = \frac{\alpha x^2 + 2xy}{(\alpha x + y)^2} \tag{7.6}$$

$$f_Y(x,y) = -\frac{x^2}{(\alpha x + y)^2} \tag{7.7}$$

$$f_{XX}(x,y) = \frac{(2\alpha x + 2y)(\alpha x + y)^{\cancel{2}} - (\alpha x^2 + 2xy) \cdot 2\alpha \cancel{(\alpha x + y)}}{(\alpha x + y)^{\cancel{4}}} = \frac{2y^2}{(\alpha x + y)^3} \tag{7.8}$$

$$f_{YY}(x,y) = \frac{2x^2}{(\alpha x + y)^3} \tag{7.9}$$

$$f_{XY}(x,y) = -\frac{2xy}{(\alpha x + y)^3} \tag{7.10}$$

By substituting Equations from (7.6) to (7.10) in $E\left(\dfrac{X^2}{\alpha X + Y}\right)$ and the approximation is used in the next equation because we find the Taylor series to the

second order

$$
\begin{aligned}
E\left(\frac{X^2}{\alpha X+Y}\right) \approx E\Bigg\{ & \frac{n^2p^2}{\alpha np+mq}+\frac{\alpha n^2p^2+2npmq}{(\alpha np+mq)^2}(X-np)-\frac{n^2p^2}{(\alpha np+mq)^2}(Y-mq)+ \\
& \frac{1}{2}\Big[\frac{2m^2q^2}{(\alpha np+mq)^2}\cdot(X-np)^2-\frac{4nmpq}{(\alpha np+mq)^3}(X-np)(Y-mq)+ \\
& \frac{2n^2p^2}{(\alpha np+mq)^3}(Y-mq)^2\Big]\Bigg\} \\
= & \frac{n^2p^2}{\alpha np+mq}+\frac{m^2q^2}{(\alpha np+mq)^3}\,np(1-p)+\frac{n^2p^2}{(\alpha np+mq)^3}\,mq(1-q) \\
= & \frac{(\alpha np+mq)^2n^2p^2+nm^2pq^2-nm^2p^2q^2+n^2mp^2q-n^2mp^2q^2}{(\alpha np+mq)^3} \\
= & \frac{\alpha n^3p^3(\alpha np+2mq)+n^2m^2p^2q^2\left(1-\frac{1}{n}-\frac{1}{m}+\frac{1}{np}+\frac{1}{mq}\right)}{(\alpha np+mq)^3} \\
= & \frac{\alpha n^3p^3(\alpha np+2mq)+n^2m^2p^2q^2\left(1+\frac{1-p}{np}+\frac{1-q}{mq}\right)}{(\alpha np+mq)^3}.
\end{aligned}
$$

And by substituting the above expression in Equation (7.5), we have

$$
E\left(\frac{W_{\mathcal{L}1}^2}{W_{\mathcal{L}}}\right) \approx \frac{e^{-3\beta}}{W_N}\left(\frac{\alpha n^3p^3(\alpha np+2mq)+n^2m^2p^2q^2\left(1+\frac{1-p}{np}+\frac{1-q}{mq}\right)}{(\alpha np+mq)^3}\right),
$$

where n, m, β and α are functions of t.

The expected value of the second and third terms in the Gini index in Equation (7.4) is

$$
E\left(W_{\mathcal{L}1}-\frac{W_{\mathcal{L}1}^2}{W_{\mathcal{L}}}\right) \approx \frac{e^{-\beta}}{W_N}np-\frac{e^{-3\beta}}{W_N}\left(\frac{\alpha n^3p^3(\alpha np+2mq)+n^2m^2p^2q^2\left(1+\frac{1-p}{np}+\frac{1-q}{mq}\right)}{(\alpha np+mq)^3}\right) \tag{7.11}
$$

The right node $\mathcal{R}$:
We know from the plot on Page 157 that

- $N_{\mathcal{R}1}(s) = N_{\mathcal{R}1}(t) + X_{\mathcal{R}1} \quad , \quad X_{\mathcal{R}1} \sim Bin\left(N_{\mathcal{L}1}(t), 1 - \frac{F_1(s)}{F_1(t)}\right)$

- $N_{\mathcal{R}2}(s) = N_{\mathcal{R}2}(t) + X_{\mathcal{R}2} \quad , \quad X_{\mathcal{R}2} \sim Bin\left(N_{\mathcal{L}2}(t), 1 - \frac{F_2(s)}{F_2(t)}\right)$

For the right node, we consider two terms as follows:

- $$\begin{aligned} E(W_{\mathcal{R}1}(s)) &= E\left(w_{\mathcal{L}1}\, X_{\mathcal{R}1} + w_{\mathcal{R}1}\, N_{\mathcal{R}1}(t)\right) \\ &= w_{\mathcal{L}1}\, n(1-p) + w_{\mathcal{R}1}\, a\, \bar{F}_1(t) \\ &= \frac{e^{-\beta}}{W_N}\, n(1-p) + \frac{e^{\beta}}{W_N}\, a\, \bar{F}_1(t), \end{aligned}$$

where $\bar{F}_j(t) = 1 - F_j(t)$, $a = N_{\mathcal{R}1}(t)$, $b = N_{\mathcal{R}2}(t)$, $p = \frac{F_1(s)}{F_1(t)}$ and $q = \frac{F_2(s)}{F_2(t)}$.

The second term for the right node is

- $$E\left(\frac{W^2_{\mathcal{R}1}(s)}{W_{\mathcal{R}}(s)}\right) = \sum_{\substack{n_{\mathcal{R}1(s)} \\ n_{\mathcal{R}2(s)}}} \left(\frac{\frac{e^{-\beta}}{W_N} X_{\mathcal{R}1} + \frac{e^{\beta}}{W_N} a}{\frac{e^{-\beta}}{W_N} X_{\mathcal{R}1} + \frac{e^{\beta}}{W_N} a + \frac{e^{\beta}}{W_N} X_{\mathcal{R}2} + \frac{e^{-\beta}}{W_N} b} \right) Pr\Big(N_{\mathcal{R}1}(s) = n_{\mathcal{R}1}(s)\ ,\ N_{\mathcal{R}2}(s) = n_{\mathcal{R}2}(s)\Big)$$
$$= \frac{e^{-\beta}}{W_N} \sum_{\substack{n_{\mathcal{R}1(s)} \\ n_{\mathcal{R}2(s)}}} \left(\frac{(X_{\mathcal{R}1} + e^{2\beta}\, a)^2}{(X_{\mathcal{R}1} + e^{2\beta}\, a) + (b + e^{2\beta}\, X_{\mathcal{R}2})} \right) Pr\Big(N_{\mathcal{R}1}(s) = n_{\mathcal{R}1}(s)\ ,\ N_{\mathcal{R}2}(s) = n_{\mathcal{R}2}(s)\Big).$$

Then,

$$E\left(\frac{W^2_{\mathcal{R}1}(s)}{W_{\mathcal{R}}(s)}\right) = \frac{e^{-\beta}}{W_N} E\left(\frac{X'^2}{X' + Y'}\right), \tag{7.12}$$

where $X' = X_{\mathcal{R}1} + e^{2\beta}\, a$ and $Y' = b + e^{2\beta}\, X_{\mathcal{R}2}$.

By finding Taylor Series to the second order for $E\left(\frac{X'^2}{X'+Y'}\right)$ around the mean of each random variable $(X',\ Y') \equiv \big(n\,(1-p) + e^{2\beta}\, a,\ b + e^{2\beta} m\,(1-q)\big)$.

$$E\left(\frac{X'^2}{X'+Y'}\right) \approx \frac{\Big(n(1-p)+e^{2\beta}\,a\Big)^2}{\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)}+$$
$$\frac{\Big(b+e^{2\beta}m\,(1-q)\Big)^2}{\left(\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)\right)^3}\; n\,p\,(1-p)+$$
$$\frac{\Big(n(1-p)+e^{2\beta}\,a\Big)^2}{\left(\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)\right)^3}\; e^{4\beta}m\,q\,(1-q).$$

And by substituting the above expression in Equation (7.12), we have

$$E\left(\frac{W^2_{\Re 1}(s)}{W_{\Re}(s)}\right) \approx \frac{e^{-\beta}}{W_N}\left(\frac{\Big(n(1-p)+e^{2\beta}\,a\Big)^2}{\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)}+\right.$$
$$\frac{\Big(b+e^{2\beta}m\,(1-q)\Big)^2}{\left(\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)\right)^3}\; n\,p\,(1-p)+$$
$$\left.\frac{\Big(n(1-p)+e^{2\beta}\,a\Big)^2}{\left(\Big(n(1-p)+e^{2\beta}\,a\Big)+\Big(b+e^{2\beta}m\,(1-q)\Big)\right)^3}\; e^{4\beta}m\,q\,(1-q)\right).$$

The expected value of the fourth and fifth terms in the Gini index in Equation (7.4) is

$$E\left(W_{\Re 1}-\frac{W_{\Re 1}^2}{W_{\Re}}\right)\approx\frac{e^{-\beta}}{W_N}\,n(1-p)+\frac{e^{\beta}}{W_N}\,a\,\bar{F}_1(t)-\frac{e^{-\beta}}{W_N}\left(\frac{\left(n(1-p)+e^{2\beta}\,a\right)^2}{\left(n(1-p)+e^{2\beta}\,a\right)+\left(b+e^{2\beta}m\,(1-q)\right)}+\right.$$
$$\frac{\left(b+e^{2\beta}m\,(1-q)\right)^2}{\left(\left(n(1-p)+e^{2\beta}\,a\right)+\left(b+e^{2\beta}m\,(1-q)\right)\right)^3}\;\;n\,p\,(1-p)+$$
$$\left.\frac{\left(n(1-p)+e^{2\beta}\,a\right)^2}{\left(\left(n(1-p)+e^{2\beta}\,a\right)+\left(b+e^{2\beta}m\,(1-q)\right)\right)^3}\;\;e^{4\beta}m\,q\,(1-q)\right). \tag{7.13}$$

The change in the weighted Gini is given in Equation (7.4), and by substituting $E\left(W_{\mathcal{L}1}(s) - \frac{W^2_{\mathcal{L}1}(s)}{W_{\mathcal{L}}(s)}\right)$ and $E\left(W_{\Re 1}(s) - \frac{W^2_{\Re 1}(s)}{W_{\Re}(s)}\right)$ in Equation (7.4), we have

$$
\begin{aligned}
& E(W_{\mathcal{L}1}) - E\left(\frac{W^2_{\mathcal{L}1}}{W_{\mathcal{L}}}\right) + E(W_{\Re 1}) - E\left(\frac{W^2_{\Re 1}}{W_{\Re}}\right) \\
& \approx \frac{e^{-\beta}}{W_N}\, n\, p - \frac{e^{-3\beta}}{W_N}\left[\frac{\alpha n^3 p^3(\alpha np + 2mq) + n^2 m^2 p^2 q^2 (1 + \frac{1-p}{np} + \frac{1-q}{mq})}{(\alpha np + mq)^3}\right] + \\
& \frac{e^{-\beta}}{W_N}\, n(1-p) + \frac{e^{\beta}}{W_N}\, a\, \bar{F}_1(t) - \frac{e^{-\beta}}{W_N}\left[\frac{\left(n(1-p) + e^{-\alpha}\, a\right)^2}{\left(n(1-p) + e^{-\alpha}\, a\right) + \left(b + e^{-\alpha} m\,(1-q)\right)} + \right. \\
& \frac{\left(b + e^{-\alpha} m\,(1-q)\right)^2}{\left(\left(n(1-p) + e^{-\alpha}\, a\right) + \left(b + e^{-\alpha} m\,(1-q)\right)\right)^3}\; n\, p\,(1-p) + \\
& \left. \frac{\left(n(1-p) + e^{-\alpha}\, a\right)^2}{\left(\left(n(1-p) + e^{-\alpha}\, a\right) + \left(b + e^{-\alpha} m\,(1-q)\right)\right)^3}\; e^{4\beta} m\, q\,(1-q)\right].
\end{aligned}
\tag{7.14}
$$

Defining the magnitude of terms in Equation (7.14)

- $\dfrac{e^{-\beta}}{W_N}\, n + \dfrac{e^{\beta}}{W_N}\, a\, \bar{F}_1(t) \equiv O(1)$
- $\dfrac{\alpha n^3 p^3(\alpha np + mq)}{W_N(\alpha np + mq)^3} + \dfrac{\alpha n^3 p^3 mq}{W_N(\alpha np + mq)^3} + \dfrac{n^2 m^2 p^2 q^2}{W_N(\alpha np + mq)^3} \equiv O(1)$
- $\dfrac{\Big(n(1-p) + e^{2\beta}\, a\Big)^2}{W_N\left(\Big(n(1-p) + e^{2\beta}\, a\Big) + \Big(b + e^{2\beta} m\,(1-q)\Big)\right)} \equiv O(1)$
- The rest are $\quad O\left(\dfrac{1}{n}\right)$

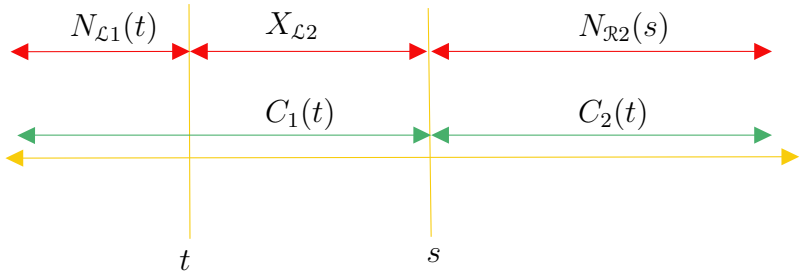
From Equation (7.14) and by ignoring terms that are $O\left(\dfrac{1}{n}\right)$, we will have

$$\begin{aligned} E(W_{\mathcal{L}1}) - E\left(\frac{W_{\mathcal{L}1}^2}{W_{\mathcal{L}}}\right) + E(W_{\mathcal{R}1}) - E\left(\frac{W_{\mathcal{R}1}^2}{W_{\mathcal{R}}}\right) \approx & \frac{e^{-\beta}}{W_N}\, n + \frac{e^{\beta}}{W_N}\, a\, \bar{F}_1(t) - \\ & \frac{e^{-\beta}}{W_N} \frac{\Big(n(1-p) + e^{2\beta}\, a\Big)^2}{\Big(n(1-p) + e^{2\beta}\, a\Big) + \Big(b + e^{2\beta} m\,(1-q)\Big)} \\ & - \frac{e^{-3\beta}}{W_N}\left[\frac{\alpha n^3 p^3}{(\alpha np + mq)^2} + \frac{\alpha n^3 p^3 mq}{(\alpha np + mq)^3} + \frac{\alpha n^3 p^3 mq + n^2 p^2 q^2}{(\alpha np + mq)^3}\right], \end{aligned} \tag{7.15}$$

and by substituting Equation (7.15) in Equation (7.4), we have

$$E\left(\triangle_{\Gamma_w(s,t)}\right) \approx \Gamma_w\left(\mathcal{P}\right) - \frac{2}{W_N}\left\{\frac{e^{-\beta}}{W_N}\, n + \frac{e^{\beta}}{W_N}\, a\, \bar{F}_1(t) - \frac{e^{-\beta}}{W_N}\frac{\left(n(1-p)+\frac{1}{\alpha}\, a\right)^2}{\left(n(1-p)+\frac{1}{\alpha}\, a\right)+\left(b+\frac{1}{\alpha}m\,(1-q)\right)}\right.$$
$$\left. - \frac{e^{-3\beta}}{W_N}\left[\frac{\alpha n^3 p^3}{(\alpha np+mq)^2} + \frac{\alpha n^3 p^3 mq}{(\alpha np+mq)^3} + \frac{\alpha n^3 p^3 mq + n^2 p^2 q^2}{(\alpha np+mq)^3}\right]\right\}. \tag{7.16}$$

The second case:
Similarly, in case of $s > t$, and from the plot as follows



We know that $N_{\mathcal{L}1}(s) = X_{\mathcal{L}2} + N_{\mathcal{L}1}(t)$, and $X_{\mathcal{L}1}$, $N_{\mathcal{R}1}(t)$ and $N_{\mathcal{L}1}(t)$ are binomially distributed as follows:

$$\bullet N_{\mathcal{L}1}(t) \sim Bin\left(N, F_1(t)\right) \quad \bullet\, N_{\mathcal{R}1}(s) \sim Bin\left(N_{\mathcal{R}1}(t), \bar{p}\right)$$

From Equation (7.16), we find the reflection by replacing every n by a, m by b, $F(t)$ by $\bar{F}(t)$, $e^{-\beta}$ by e^{β}, p by $\bar{p}$, and q by $\bar{q}$, so we will have

$$E\left(\triangle_{\Gamma_w(s,t)}\right) \approx \Gamma_w\left(\mathcal{P}\right) - \frac{2}{W_N}\Bigg\{\frac{e^{\beta}}{W_N}\,a + \frac{e^{-\beta}}{W_N}\,n\,F_1(t) - \frac{e^{\beta}}{W_N}\frac{\Big(a(1-\bar{p}) + \alpha\,n\Big)^2}{\Big(a(1-\bar{p}) + \alpha\,n\Big) + \Big(m + \alpha b\,(1-\bar{q})\Big)}$$
$$- \frac{e^{3\beta}}{W_N}\left[\frac{\frac{1}{\alpha}a^3\bar{p}^3}{(\frac{1}{\alpha}a\bar{p} + b\bar{q})^2} + \frac{\frac{1}{\alpha}a^3\,\bar{p}^3\,b\,\bar{q}}{(\frac{1}{\alpha}a\bar{p} + b\bar{q})^3} + \frac{\frac{1}{\alpha}a^3\bar{p}^3\,b\,\bar{q} + a^2\bar{p}^2\bar{q}^2}{(\frac{1}{\alpha}a\bar{p} + b\bar{q})^3}\right]\Bigg\}, \tag{7.17}$$

where $e^{\beta} = \left(\dfrac{N - N_{\mathcal{L}2}(t) - N_{\mathcal{R}1}(t)}{N_{\mathcal{L}2}(t) + N_{\mathcal{R}1}(t)}\right)^{\frac{1}{2}}$, $\alpha = e^{-2\beta}$, $n = N_{\mathcal{L}1}(t)$, $m = N_{\mathcal{L}2}(t)$, $a = N_{\mathcal{R}1}(t)$, $b = N_{\mathcal{R}2}(t)$, $p = \frac{F_1(s)}{F_1(t)}$ and $q = \frac{F_2(s)}{F_2(t)}$. Equation (7.17) matches our calculation to find the results in case of $s > t$.

From Section 3.3, we have the expected values of Gini as in Equation (7.18)

$$E\big(\triangle_{\Gamma}(t)\big) \propto \Gamma\left(\mathcal{P}\right) - 2\Bigg\{1 - \frac{\big(F_1(t) - F_2(t)\big)^2}{F(t)(1-F(t))}\Bigg\}, \tag{7.18}$$

the expected value of Gini as in Equation (7.18) is $Pr(T = t)$ after normalisation. From the law of total expectations and by adding Equations (7.16) and (7.17), then multiplying by Equation (7.18), we will have Equation (7.19). The second split s depends on the first split t since s is initially conditioned on the first split point t, then we need to integrate Equation (7.19) out to get the marginal distribution.

$$E\left(\triangle_{\Gamma_w}(s,T)\right) = \int_{s<t} E\left(\triangle_{\Gamma_w}(s,T)|T=t\right) Pr(T=t)\mathrm{d}t + \int_{s>t} E\left(\triangle_{\Gamma_w}(s,T)|T=t\right) Pr(T=t)\mathrm{d}t. \tag{7.19}$$

We will stop our calculation at this point as it is a very complicated integral and all terms are functions of t in the numerator and denominator.

7.3 Weighted Gini Stumps Algorithms and Applications

We will explain in this section weighted Gini stumps methods and apply these methods on simulated and real datasets.

7.3.1 Weighted Gini Stumps Algorithms

We introduced Gini stumps methods in Chapter 5, and here we will combine these methods with AdaBoost algorithms.

In this section, we use the same stump methods as were explained in Algorithms 3 and 5 in Subsection 5.1.2 but now including a weight update. The classifier $d_q(x)$ or $P_q(x)$ in AdaBoost Algorithms 8 or 9 is replaced by a Gini stump combined with weights, we will refer to this combination as "weighted Gini stumps methods". Weighted Gini stumps methods are explained in more detail in Algorithms 10 and 11. The weighted Gini index is given by Equation (3.6) which is

$$\Gamma(c) = 1 - \sum_{j=1}^{J} \left(\frac{W_{cj}}{W_c} \right)^2,$$

where W_{cj} is the weight of observations at node c and class j, and W_c is the total weight of observations at that node.

This version of Gini index is used in weighted Gini stumps methods to have the ability to change the weights of observations according to AdaBoost techniques. We have two methods of Gini stumps, which are Gini-sampled stumps and Gini-midpoints stumps, as it was introduced in Chapter 5, and with two AdaBoost techniques, we will have four methods as in Table 7.2.

	AdaBoost techniques	
Method	Discrete AdaBoost (DA)	Real AdaBoost (RA)
Weighted Gini-sampled stumps	Weighted Gini-sampled stumps with DA	Weighted Gini-sampled stumps with RA
Weighted Gini-midpoints stumps	Weighted Gini-midpoints stumps with DA	Weighted Gini-midpoints stumps with RA

Table 7.2: Gini stumps with AdaBoost methods.

Algorithms 10 and 11 explain how to combine Gini stumps methods with AdaBoost techniques. These algorithms start by initialising the weights $w_i = \frac{1}{n}, i = 1, \ldots, n$, for x_i, then computing the weighted Gini indices of the explanatory variable, after that one of the Gini stumps algorithms that are introduced in Chapter 5 is used as the base classifier. Following this, the weight of observation is updated according to either discrete or real AdaBoost. Finally, outputting the classifiers.

Algorithm 10 Weighted Gini stumps with discrete AdaBoost.

1. Start with weights $w_i = \frac{1}{n}, i = 1, 2, \ldots, n$
2. Compute the weighted Gini indices
   For$(q \leftarrow 1 \quad to \quad \mathcal{Q})$
   (a) Apply Algorithm 3 or Algorithm 5 in Chapter 5 with $B = 1$ and use it as the classifier $d_q(x) \in \{-1, 1\}$, and with weight w_i for a single stump
   (b) Compute $\epsilon_q = \frac{\sum_{i=1}^{n} w_i I(y_i \neq d_q(x_i))}{\sum_{i=1}^{n} w_i}$
   (c) Compute $\gamma_q = \frac{1}{2} \log(\frac{1-\epsilon_q}{\epsilon_q})$
   (d) Update $w_i = w_i \exp[\gamma_q I(y \neq d_q(x_i))]$, and renormalise the weights by dividing by $\sum_i w_i$.
3. Output the classifier: $\text{sign}\big[\sum_{q=1}^{\mathcal{Q}} \gamma_q d_q(x)\big]$

Algorithm 11 Weighted Gini stumps with real AdaBoost.

1. Start with weights $w_i = \frac{1}{n}, i = 1, 2, \ldots, n$
2. Compute the weighted Gini indices
   For$(q \leftarrow 1 \quad to \quad \mathcal{Q})$
   (a) Apply Algorithm 3 or Algorithm 5 in Chapter 5 with $B = 1$ and use it as the classifier to obtain a class probability estimate $P_q(x) = \hat{P}_w\big(y = 1|x\big)$, and with weights w_i
   (b) Set $h_q(x) = \frac{1}{2}\log\left(\frac{P_q(x)}{1-P_q(x)}\right) \in \mathbb{R}$
   (c) Update $w_i = w_i \exp\left[-y_i h_q(x_i)\right]$, and renormalise the weights by dividing by $\sum_i w_i$
3. Output the classifier: $\text{sign}\big[\sum_{q=1}^{\mathcal{Q}} h_q(x)\big]$

7.3.2 Applying Weighted Gini Stumps to Simulated Data

The performance accuracy of Algorithms 10 and 11 is measured with the five simulated models that are mentioned in Subsection 4.3.2 with different κ values and number of weight updates $\mathcal{Q} = 500$.

Table 7.3 presents the average of percentages of accuracy of 50 simulations from each model. The two algorithms with two AdaBoost techniques are tested with different values of $\kappa \in \{0, \frac{1}{3}, \frac{1}{2}, 1, 2, 3, 4\}$.

Model 1 shows a similar accuracy with different κ values with slightly higher percentages at $\kappa = 1$ with real AdaBoost. While model 1 shows similar accuracy with different κ values, we can see that model 2 has a lower performance with smaller κ and real AdaBoost method. Discrete AdaBoost method is doing better than real AdaBoost method with this model.

Model 3 shows similar percentages with real AdaBoost method and percentages decrease with increasing of κ values with discrete AdaBoost method. The highest percentages are at $\kappa = \frac{1}{2}$ and real AdaBoost. Percentages decrease with discrete AdaBoost from around 82% to 75% with weighted Gini-sampled stumps and

from 82% to almost 80% with weighted Gini-midpoints stumps. Weighted Gini-midpoints stumps method gives a better performance with this model.

Model 4 has quite similar performance to model 2, where it performs fine with discrete AdaBoost and gives poor performance with real AdaBoost but the only difference between the performance with model 2 and this model is: there is a large difference between the percentages from discrete and real methods. Real AdaBoost with weighted Gini-sampled stumps percentages start from 68% to 75%, and with weighted Gini-midpoints stumps from 73% to 78%.

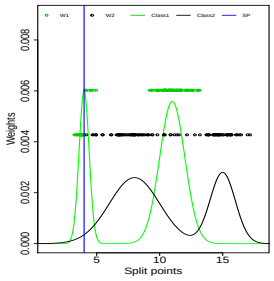
Finally, both methods show similar results with discrete AdaBoost, and increasing in the percentages with real AdaBoost when κ values increase in model 5.

Discrete AdaBoost shows better results than real AdaBoost in three out of five models and more stable performance with small differences in four out of five models. Weighted Gini-midpoints stumps method performs better than weighted Gini-sampled stumps in three out of five models.

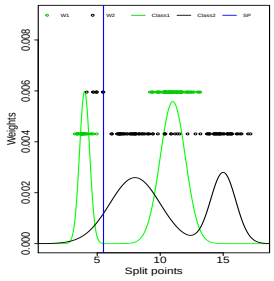
As we can see from Table 7.3, the performance of model 4 differs noticeably according to the two AdaBoost methods, due to that we will study the way the weights change in the two AdaBoost methods with the two Gini stumps methods. Figures 7.5 and 7.6 show the first weight update according to either discrete or real AdaBoost in a simulation from model 4 with a set of split points $\{4, 5.5, 7, 8.5, 10, 11.5, 13, 14.5, 16\}$. The number of observations in this simulation is $n = 200$ and the initial weight is $w_i = \frac{1}{200} = 0.005$ for observation x_i. The misclassified data points are then boosted with extra weights after the first weight update by the two AdaBoost methods. However, the changing in the weights in real AdaBoost is small as in Figures 7.6 compared to the change in weights in discrete AdaBoost in Figure 7.5.

Model	κ	Weighted Gini-sampled stumps		Weighted Gini-midpoints stumps	
		Discrete	Real	Discrete	Real
1	0	93.51	93.63	93.57	93.70
	$\frac{1}{3}$	93.68	93.78	93.64	93.82
	$\frac{1}{2}$	93.67	93.80	93.59	93.89
	1	93.72	93.85	93.75	93.91
	2	93.63	93.79	93.77	93.84
	3	93.69	93.78	93.70	93.81
	4	93.53	93.78	93.61	93.74
2	0	89.89	87.87	89.74	85.97
	$\frac{1}{3}$	89.85	88.04	89.57	86.58
	$\frac{1}{2}$	89.83	88.16	89.75	86.81
	1	89.84	88.63	89.73	87.65
	2	89.78	89.01	89.79	88.30
	3	89.79	89.19	89.71	88.52
	4	89.74	89.24	89.30	88.71
3	0	82.47	82.50	82.61	82.41
	$\frac{1}{3}$	82.57	82.63	82.41	82.50
	$\frac{1}{2}$	82.58	82.68	82.44	82.65
	1	82.28	82.65	82.28	82.59
	2	80.76	82.57	81.26	82.50
	3	77.52	82.47	80.35	82.40
	4	75.71	82.41	79.29	82.34
4	0	86.85	68.32	86.56	73.05
	$\frac{1}{3}$	86.81	68.37	86.69	73.76
	$\frac{1}{2}$	86.65	68.83	86.62	74.17
	1	86.79	68.76	86.64	74.14
	2	86.67	70.40	86.40	76.24
	3	86.65	72.96	86.53	76.98
	4	86.55	75.14	86.41	78.45
5	0	77.42	75.51	77.24	72.78
	$\frac{1}{3}$	77.34	75.31	77.20	73.70
	$\frac{1}{2}$	77.38	75.46	77.15	74.35
	1	77.37	75.93	77.05	75.61
	2	77.32	76.58	76.69	76.70
	3	76.86	76.80	76.54	77.03
	4	76.53	77.01	76.01	77.06

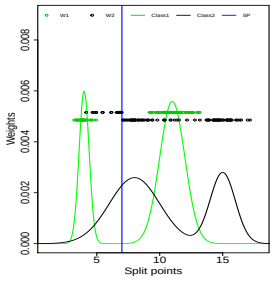
Table 7.3: The average of percentages of correctly predicted classes of 50 simulations from five models by using weighted Gini stumps methods and AdaBoost methods. The red results indicate the largest results of Gini-sampled stumps and the green results show the largest results of Gini-midpoints stumps.



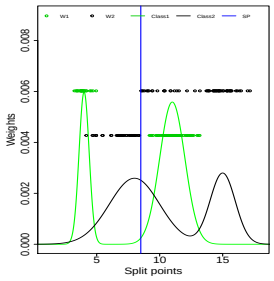
(a) Split point = 4



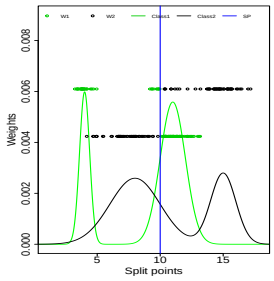
(b) Split point = 5.5



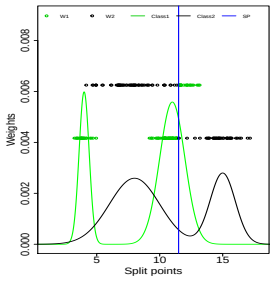
(c) Split point = 7



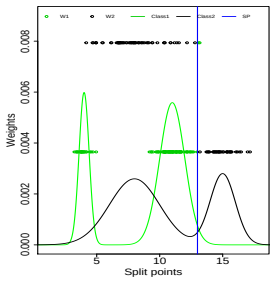
(d) Split point = 8.5



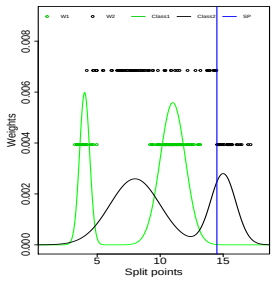
(e) Split point = 10



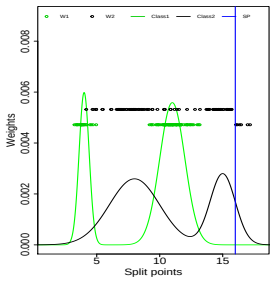
(f) Split point = 11.5



(g) Split point = 13

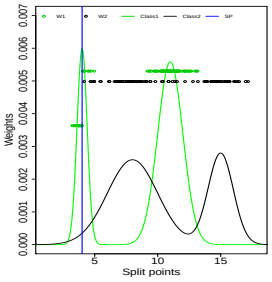


(h) Split point = 14.5

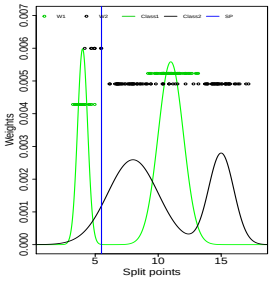


(i) Split point = 16

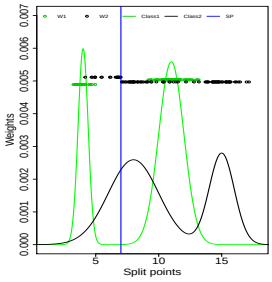
Figure 7.5: First weight update for observations from model 4 by using a set of split points $\{4, 5.5, 7, 8.5, 10, 11.5, 13, 14.5, 16\}$ with discrete AdaBoost algorithm. The green and black points are the weight of observation based on their classes and the blue lines are the split points.



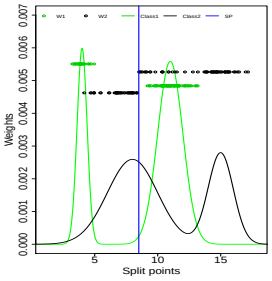
(a) Split point = 4



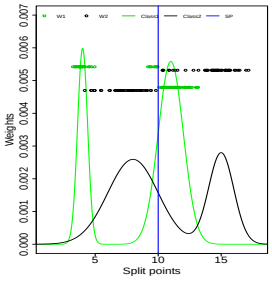
(b) Split point = 5.5



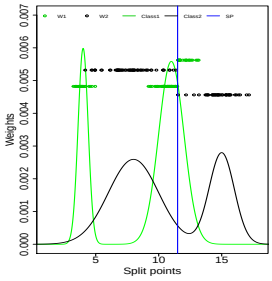
(c) Split point = 7



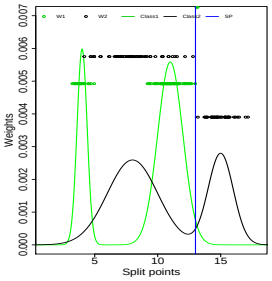
(d) Split point = 8.5



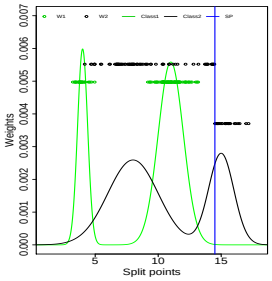
(e) Split point = 10



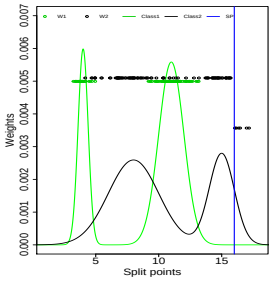
(f) Split point 11.5



(g) Split point = 13



(h) Split point = 14.5



(i) Split point = 16

Figure 7.6: First weight update for observations from model 4 by using a set of split points $\{4, 5.5, 7, 8.5, 10, 11.5, 13, 14.5, 16\}$ with real AdaBoost algorithm. The green and black points are the weight of observation based on their classes and the blue lines are the split points.

It is clearly seen that the weight update is slower in case of real AdaBoost and because of that real AdaBoost might not classify the observations correctly after the same number of iterations as discrete AdaBoost does. We saw in Table 7.1 that real AdaBoost method gave less accurate prediction than discrete AdaBoost.

To investigate this further, we will increase the number of iterations in these three models 2, 4 and 5 with real AdaBoost and the two Gini stumps methods. Table 7.4 shows the percentages correct of real AdaBoost with Gini stumps methods after increasing the number of weight updates to become $\mathcal{Q} = 1500$ instead of $\mathcal{Q} = 500$ as in Table 7.3. Increasing the number of iterations has a noticeable positive impact on model 4 performance which has the more obvious difference between the accuracy of the two AdaBoost methods.

Finally, comparing between the results in Table 7.1 and results in Tables 7.3 and 7.4, we find that regular stumps with AdaBoost and weighted Gini stumps methods have similar results in the case of $\kappa = 1$ and discrete AdaBoost. However, the Gini stumps method is faster than regular stumps as it is approved in Chapter 5.

Model	κ	Weighted Gini-sampled stumps	Weighted Gini-midpoints stumps
2	0	87.87	85.94
	$\frac{1}{3}$	88.80	87.74
	$\frac{1}{2}$	89.02	88.17
	1	89.37	88.87
	2	89.49	89.27
	3	89.52	89.35
	4	89.50	89.36
4	0	69.32	72.67
	$\frac{1}{3}$	75.54	80.82
	$\frac{1}{2}$	79.33	83.72
	1	85.18	86.37
	2	86.65	86.86
	3	86.87	86.81
	4	86.96	86.86
5	0	75.41	72.87
	$\frac{1}{3}$	76.77	76.86
	$\frac{1}{2}$	77.07	77.35
	1	77.36	77.40
	2	77.38	77.40
	3	77.37	77.32
	4	77.29	77.37

Table 7.4: The average of percentages of correctly predicted classes of 50 simulations from three models by using weighted Gini stumps methods with real AdaBoost and increasing the number of weight updates to $\mathcal{Q} = 1500$.

7.3.3 Applying Weighted Gini Stumps to Real Datasets

The two weighted Gini stumps methods, and discrete and real AdaBoost are applied to real datasets that were introduced in Subsection 6.1.1. Table 7.5 represents the average of percentages of correctly predicted classes of 10-folds from each dataset and the number of iterations is $\mathfrak{Q} = 100$.

Banknote dataset has a different accuracy performance with the AdaBoost methods, and discrete AdaBoost provides a better accuracy by almost 8%. Vertebral datasets performs as Banknote dataset but with a large difference between the average of the percentages that is almost 17% in weighted Gini-sampled stumps and 14% in weighted Gini-midpoints stumps. The other three datasets have a better performance with discrete AdaBoost method regardless of the weighted Gini stumps methods. However, the differences in the latter three datasets between discrete and real AdaBoost methods is around 4%.

By comparing between the weighted Gini stumps methods, we find that weighted Gini-sampled stumps with discrete AdaBoost is doing better than weighted Gini-midpoints stumps method with discrete AdaBoost especially with Vertebral dataset. This pattern of performance also appears with the weighted Gini stumps methods with real AdaBoost.

To conclude, weighted Gini-sampled stumps with discrete AdaBoost is the most accurate method out of the four methods with AdaBoost and these real datasets.

Dataset	Weighted Gini-sampled stumps		Weighted Gini-midpoints stumps	
	Discrete	Real	Discrete	Real
Banknotes	94.83	86.15	93.66	86.52
Vertebral	87.74	70.74	82.68	68.06
Cancer	96.65	91.40	95.43	91.38
Survival	92.47	90.15	91.90	88.90
Blood	96.65	92.45	95.43	91.38

Table 7.5: The average of percentages of correctly predicted classes of 10-folds from the five real datasets by using weighted Gini stumps methods and AdaBoost.

7.4 Comparing between Bagging and Boosting

We will compare between bagging and boosting performance with simulated and real datasets.

7.4.1 Simulated Datasets

By comparing between bagging results in Table 5.5 and boosting results in Table 7.3, we find that weighted Gini stumps methods are slightly better than Gini stumps methods in general with model 1, as the weighted version of Gini stumps gives higher percentages at $\kappa = 0$ and $\frac{1}{3}$ with both AdaBoost techniques.

Models 2 and 3 have the same pattern of performance which is better with weighted Gini stumps with both AdaBoost methods and at all κ values, whereas Gini stumps methods are doing well with just weighted vote aggregation method and and at $\kappa = 0$.

Model 4 has higher results with weighted Gini stumps methods than Gini stumps methods and especially with discrete AdaBoost. We can see here that real AdaBoost has less accurate results than discrete AdaBoost but still higher than Gini stumps with both aggregation methods.

Finally, model 5 has very accurate results with discrete AdaBoost method with both weighted Gini stumps methods. Gini stumps with no weight has slightly higher performance with weighted vote aggragation method but it is not as accurate as weighted Gini stumps. Weighted Gini stumps achieved 77.42% and the maximal accuracy is 78.51%, while the percentage is achieved by Gini stumps is 75.84%.

7.4.2 Real Datasets

The performance of bagging in Table 6.3 and boosting with real datasets in Table 7.5 show that weighted Gini stumps methods perform more accurately than Gini stumps with aggregation methods with these datasets.

Generally, weighted Gini stumps method is doing better than the unweighted version of Gini stumps method with both simulated and real datasets.

7.5 Summary of the Chapter

In this chapter, we combine between Gini stumps methods with weight updates and AdaBoost methods, and refer to these algorithms as Weighted Gini stumps methods. Then, we try to calculate the expected values of Gini after the first update and reach very complected integrals. Finally, we investigate weighted Gini stumps methods performance and find that they are more accurate than forests of Gini stumps with simulations and real datasets especially with discrete AdaBoost.

Chapter 8

General Conclusion and Future Work

8.1 General Conclusion

In this thesis, we investigated stumps ensemble methods with weight updates. We introduced a new method of forests of stumps by using Gini gain to generate the split points and called it Gini stumps method. Our new method uses a probabilty density function, which is proportional to Gini gain, to generate split points.

A review of decision trees was delivered in Chapter 3. This review includes the constructing of classification trees and an explanation of splitting criteria. Trees have many splitting criteria but in our research we focused on Gini index as it is the default splitting criterion for CART. We found the expected value of Gini gain theoretically and established a link to two sample Anderson-Darling type test statistics, also noting that the optimal Gini split point differs from that obtained by Bayes rule if the priors are equal. A short summary of the tree pruning process and some of the aggregation methods were given. Finally, a brief description of random forests was explained.

Chapter 4 showed the impact of changing weights or priors on tree decisions. We investigated different weighting schemes to compare between bagging scheme

on the one hand and changing observation weights according to different distributions scheme on the other hand. The comparison focused on the first split variable and split point. We found that changing the observation weights from bagging to weights according to a distribution does not have an effect on the split variable distribution but can have an effect on the split points distribution in some cases.

The developed methods were shown in Chapter 5. We explained in detail the two methods of stumps; one of them was bagging with stumps and the other method was Gini stumps method (with two sub-methods). The performance of these methods was measured by using simulations. Gini stumps method with weighted vote aggregation method especially had promising results in most cases. We also compared in this chapter between the bagging with stumps method and Gini stumps method in terms of their consumed time to generate split points and the distribution of the first split point.

Bagging with stumps and Gini stumps methods were investigated further with real-world datasets in Chapter 6. We found that there was a noticeable difference in the performance of the two methods in one of the datasets. There were two potential reasons of this variation in the performance, either the way each method selects the split variable or the nature of the datasets. These two reasons were discussed in more detail through this chapter and we found that there are some datasets perform better with bagging with stumps. However, most datasets which were investigated in this chapter were performing better with Gini stumps method.

Finally, Chapter 7 introduced boosting with Gini stumps where we combined between the new way of generating split points and AdaBoost methods. We tried in this chapter to find the weighted Gini theoretically after the first weight update but reached a very complicated integral and stopped. The combination of Gini and AdaBoost methods was investigated with simulations and real-world datasets.

8.2 Future Work

One of the important topics that probably worth investigating is the impact of split points variation in enhancing the overall performance of the random forest. Also, currently Gini stumps method is dealing with one-level trees as is clearly obvious from its name. We believe some of the future work from this thesis can be generalising this method to grow fully-grown-trees instead of stumps. We tried two-level trees in this thesis and it showed an accurate performance in case of datasets that have one explanatory variable. Another potential idea is to design this method to deal with datasets which have more than two classes in the response variable. A useful extension of this work would be calculating the expected value of the weighted Gini gain and combining Gini fully-grown-trees with Boosting methods. Gini stumps method showed promising results with a simulation that has a larger number of variables than observations, however, this idea needs further investigations, as it looks almost intractable. Investigating Gini stumps methods with more real-world datasets is going to give a more reliable conclusion and what kind of data are predicted to be more suitable for using with this method.

References

[1] S. K. Kwak and J. H. Kim. Statistical data preparation: management of missing values and outliers. *Korean Journal of Anaesthesiology*, 70(4):407, 2017.

[2] G. E. Hinton, T. J. Sejnowski, T. A. Poggio, et al. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999.

[3] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole Statistics-Probability Series. Taylor & Francis, 1984.

[4] C. Zhang and Y. Ma. *Ensemble Machine Learning: Methods and Applications*. Springer, 2012.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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

[8] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2009.

[9] L. Breiman. *Classification and Regression Trees*. Routledge, 2017.

[10] T. Y. Young and T. W. Calvert. *Classification, Estimation and Pattern Recognition*. North-Holland, 1974.

[11] E. Frank and R. R. Bouckaert. Naive Bayes for text classification with unbalanced classes. pages 503–510, 2006.

[12] J Engel. Polytomous logistic regression. *Statistica Neerlandica*, 42(4):233–252, 1988.

[13] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[14] V. K. Ojha, A. Abraham, and V. Snášel. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60:97–116, 2017.

[15] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: Data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[16] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004.

[17] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics, Springer, Berlin, 2001.

[18] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. 14(2):1137–1145, 1995.

[19] B. Efron and R. Tibshirani. Improvements on cross-validation: the .632+ bootstrap method. *Journal of The American Statistical Association*, 92(438):548–560, 1997.

[20] C.C. Aggarwal and S. Sathe. *Outlier Ensembles: An Introduction*. Springer, 2017.

[21] W. Loh. Improving the precision of classification trees. *The Annals of Applied Statistics*, 3:1710–1737, 2009.

[22] M. Hansen, R. Dubayah, and R. DeFries. Classification trees: an alternative to traditional land cover classifiers. *International Journal of Remote Sensing*, 17(5):1075–1081, 1996.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 2014.

[24] L. E. Raileanu and K. Stoffel. Theoretical comparison between the Gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004.

[25] C. D. Ghilani. *Adjustment Computations: Spatial Data Analysis*. John Wiley & Sons, 2017.

[26] C. E. Leiserson, R. L. Rivest, T. H. Cormen, and C. Stein. *Introduction to Algorithms*, volume 6. MIT Press Cambridge, MA, 2001.

[27] T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of The American Statistical Association*, 49(268):765–769, 1954.

[28] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley. Pruning decision trees with misclassification costs. pages 131–136, 1998.

[29] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. (2):396–404, 1990.

[30] T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive partitioning and regression trees*, Apr 2017. R package version 4.1-11.

[31] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[32] E. Anderson. The species problem in Iris. *Annals of the Missouri Botanical Garden*, 23:457–509, 1936.

[33] M. Grabisch, J. Marichal, R. Mesiar, and E. Pap. Aggregation functions: means. *Information Sciences*, 181(1):1–22, 2011.

[34] T. K. Ho. Random decision forests. 1:278–282, 1995.

[35] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[36] R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of *p*. *Journal of The Royal Statistical Society*, 85(1):87–94, 1922.

[37] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of The American Statistical Association*, 69(347):730–737, 1974.

[38] W. Iba and P. Langley. Induction of one-level decision trees. pages 233–240, 1992.

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

[40] Y. Freund, R. E Schapire, et al. Experiments with a new boosting algorithm. 96:148–156, 1996.

[41] E. Dimitriadou, A. Weingessel, and K. Hornik. A cluster ensembles framework. In *Design and Application of Hybrid Intelligent Systems*, pages 528–534. 2003.

[42] B. Riemann and R. Dedekind. *Ueber Die Darstellbarkeit Einer Function Durch Eine Trigonometrische Reihe*. In Der Dieterichschen Buchhandlung, 1867.

[43] C. R. Vogel. *Computational Methods For Inverse Problems*, volume 23. SIAM, 2002.

[44] D. Dua and C Graff. UCI Machine Learning Repository, 2017. http://archive.ics.uci.edu/ml.

[45] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*, volume 112. Springer, 2013.

[46] R. Polikar, C. Zhang, and Y. Ma. *Ensemble Machine Learning: Methods and Applications*. Springer. Chapter: Ensemble Learning, 2012.

[47] Z. Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC Press, 2012.

[48] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[49] T. Hastie, S. Rosset, J. Zhu, and H. Zou. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360, 2009.