Optimal Thresholds for Classification Trees using Nonparametric Predictive Inference

ALRASHEEDI, MASAD,AWDH,MOHAMMAD

Optimal Thresholds for Classification Trees using Nonparametric Predictive Inference

Masad Awdh M. Alrasheedi

A Thesis presented for the degree of
Doctor of Philosophy

Statistics Group
Department of Mathematical Sciences
University of Durham
England

August 2022

Dedicated

To my father Awadh

To my mother Nawar

for their unlimited love, support and prayers

To my wife Asrar

To my daughter Deem

who make my life convivial during the hardest of times

My friends

for encouraging and believing

Optimal Thresholds for Classification Trees using Nonparametric Predictive Inference

Masad Awadh M. Alrasheedi

Submitted for the degree of Doctor of Philosophy
August 2022

Abstract

In data mining, classification is used to assign a new observation to one of a set of predefined classes based on the attributes of the observation. Classification trees are one of the most commonly used methods in the area of classification because their rules are easy to understand and interpret. Classification trees are constructed recursively by a top-down scheme using repeated splits of the training data set, which is a subset of the data. When the data set involves a continuous-valued attribute, there is a need to select an appropriate threshold value to determine the classes and split the data. In recent years, Nonparametric Predictive Inference (NPI) has been introduced for selecting optimal thresholds for two- and three-class classification problems, where the inferences are explicitly in terms of a given number of future observations and target proportions. These target proportions enable one to choose weights that reflect the relative importance of one class over another. The NPI-based threshold selection method has previously been implemented in the context of Receiver Operating Characteristic (ROC) analysis, but not for building classification trees. Due to the predictive nature of the NPI-based threshold selection method, it is well suited for the classification tree method, as the end goal of building classification trees is to use them for prediction as well.

In this thesis, we present new classification algorithms for building classification trees using the NPI approach for selecting the optimal thresholds. We first present

a new classification algorithm, which we call the NPI_2-Tree algorithm, for building binary classification trees; we then extend it to build classification trees with three ordered classes, which we call the NPI_3-Tree algorithm. In order to build classification trees using our algorithms, we introduce a new procedure for selecting the optimal values of target proportions by optimising classification performance on test data. We use different measures to evaluate and compare the performance of the NPI_2-Tree and the NPI_3-Tree classification algorithms with other classification algorithms from the literature. The experimental results show that our classification algorithms perform well compared to other algorithms.

Finally, we present applications of the NPI_2-Tree and NPI_3-Tree classification algorithms on noisy data sets. Noise refers to situations that occur when the data sets used for classification tasks have incorrect values in the attribute variables or the class variable. The performances of the NPI_2-Tree and NPI_3-Tree classification algorithms in the case of noisy data are evaluated using different levels of noise added to the class variable. The results show that our classification algorithms perform well in case of noisy data and tend to be quite robust for most noise levels, compared to other classification algorithms.

Declaration

The work in this thesis is based on research carried out in the Department of Mathematical Sciences at Durham University. No part of this thesis has been submitted elsewhere for any degree or qualification, and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

Alhamdulillah, Allah my God, I am truly grateful for everything you have given me in general and in accomplishing this thesis particularly.

My sincere gratitude and unlimited thanks go to my supervisors Dr. Tahani Coolen-Maturi and Prof. Frank Coolen for being a source of support in my academic life. I really appreciate all your patience, calm advice, and invaluable suggestions during the research. Without their help and guidance, this work would not have been possible.

I am eternally grateful to my parents for their unlimited support, prayers and belief in me throughout my life. I would also like to thank my brothers and sisters for their prayers and for encouraging me to undertake my PhD, I hope they are proud of this thesis. My special thanks to my wife Asrar for her support, patience and for understanding that my work kept me very busy.

Thanks to Durham University, Taibah University, and the Saudi government for the support that has enabled me to study smoothly.

Contents

Chapter 1

Introduction

In data mining techniques, classification is used to assign new observations to one of a set of predefined classes based on the attributes of the observations. The goal of classification is to predict the unknown class states (or labels) of observations when their attribute variable values are known. There are several classification methods available in the literature that one might use to predict the class states of observations. Classification (or decision) trees are one of the most commonly used because their rules are easy to understand and interpret with minimal experience. Classification trees are constructed recursively by a top-down scheme using repeated splits of the training data sets. When a data set contains a continuous-valued attribute, there is a need to select an appropriate threshold that can be used to determine the classes and split the data upon it. Thus, several methods have been developed in the literature using different approaches to find the optimal threshold value. However, these methods usually use a similar way to select the threshold values, which focus on maximising the chances of correct classification in the training data sets.

In recent years, Nonparametric Predictive Inference (NPI) has been developed as a statistical methodology based on imprecise probability theory. NPI is a statistical approach based on Hill's assumption $A_{(n)}$ [49], given in Section 2.7, with the use of lower and upper probabilities to make inferences about one or more future observations. Due to the predictive nature of the NPI approach, it has been applied in many different applications in statistics, finance, operations research, survival

analysis and reliability [18, 29, 34, 36]. Alabdulhadi [11] and Coolen-Maturi et al. [37] introduced NPI for selecting optimal thresholds for two- and three-class classification problems, where the inferences are explicitly in terms of a given number of future observations from each class. They present a direct criterion that enables one to choose target proportions that reflect the relative importance of one class over another. For example, if giving a medication to people with a disease is critical and this medication has no serious harmful effects for people without the disease, one can give more weight to the correctly classified for the diseased class than for the healthy class. It would be expected that this will increase the proportion of correctly classified people with the disease while decreasing the proportion of correctly classified people without it. The NPI-based threshold selection method has been implemented in the context of Receiver Operating Characteristic (ROC) analysis [11, 37], but not for building classification trees.

In this thesis, we present new classification algorithms for building classification trees using the NPI approach for selecting the optimal thresholds [11, 37]. We first present a new classification algorithm, which we call the NPI_2-Tree algorithm, for building binary classification trees; we then extend it to build classification trees with three ordered classes, which we call the NPI_3-Tree algorithm. The method of building classification trees using the NPI_2-Tree and NPI_3-Tree algorithms is novel in that it builds classification trees by employing the NPI approach for selecting the thresholds for data with two and three classes using predictive inference. In order to build a classification tree using our algorithms, we introduce a new procedure for selecting the optimal values of target proportions by choosing that to maximise classification performance on unseen data (testing data sets). We carry out experimental analysis on different data sets to evaluate and compare the performance of the NPI_2-Tree and the NPI_3-Tree algorithms with other classification algorithms using several evaluation measures. We also evaluate and compare the performance of our classification algorithms with others using different levels of noise added to the data sets. Noise refers to situations that occur when the data sets used for classification tasks have incorrect values in the attribute variables or class variable.

The rest of the thesis is organized as follows: Chapter 2 presents a general overview of classification trees and some classical methods for selecting the thresholds in classification trees. This is followed by background information about imprecise probability and the NPI approach.

We begin Chapter 3 by presenting the NPI method for selecting the optimal threshold for real-valued data and for the two-class classification scenario and provide an example to explain the overall methodology. We then present a new classification algorithm, which we call the NPI_2-Tree algorithm, for building binary classification trees using the NPI method for selecting the optimal threshold. A new procedure for selecting the target proportions is presented to be used with the NPI_2-Tree algorithm for building the binary classification tree. Then, we evaluate the performance of the algorithm NPI_2-Tree on six data sets taken from the UCI repository of machine learning databases [41], and we compare its performance with other classification tree algorithms. In addition, we evaluate the performance of the NPI_2-Tree algorithm using a different split criterion based on imprecise probability, and we compare the results with the classical split criterion. Some results of this chapter were presented at the 12th Workshop on Principles and Methods of Statistical Inference with Interval Probability (WPMSIIP 2019), at Durham University, UK, in September 2019. Moreover, part of this chapter's results was presented at the 12th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2019) and 13th International Conference on Computational and Financial Econometrics (CFE 2019) at the University of London, UK, in December 2019.

Chapter 4 extends the method of building the classification tree from two classes to three ordered classes. First, we present the NPI method for selecting the optimal threshold for the three ordered classes and provide a detailed example to explain the methodology. We then present a new classification algorithm, which we call the NPI_3-Tree algorithm, for building classification trees with three ordered classes. The

process of choosing the target proportions is extended to be used with the NPI_3-Tree algorithm for building classification trees. The performance of the NPI_3-Tree algorithm is evaluated on five data sets, and the results are compared with some other classification algorithms. Furthermore, we evaluate the performance of the NPI_3-Tree algorithm using imprecise split criteria and compare the results with the classical split criterion. Some results of this chapter were presented at the 12th International Symposium on Imprecise Probability: Theories and Applications (ISIPTA) at Granada, Spain, in July 2021. In addition, some results of this chapter were also presented at a seminar at Durham University.

In Chapter 5, we present applications of the NPI_2-Tree algorithm, presented in Chapter 3, and the NPI_3-Tree algorithm, presented in Chapter 4, on noisy data sets. This chapter begins with a brief summary of data noise and presents some methods for adding noise to the class variable. We then evaluate the performance of our classification algorithms on different levels of random noise added to the class variables, and we compare their results with other classification algorithms. This chapter's results were presented at a seminar at Durham University.

Finally, in Chapter 6, we provide the conclusions of this thesis, and we suggest some interesting future works to extend what is presented in this thesis. In the final sections of Chapters 3, 4 and 5, some of these future works are also discussed.

Chapter 2

Preliminaries

In this chapter, we present the basic concepts from the literature to provide background information for the topics considered in this thesis. An overview of noisy data is introduced at the beginning of Chapter 5. First, we present an overview of classification trees. We present some of the most commonly used split criteria for building classification trees and the methods used for selecting the threshold value. Then, we present some evaluation metrics used to measure the performance of classification trees. After that, we introduce the main idea of imprecise probability and present some studies that used imprecise probability to build classification trees. Finally, we provide a brief overview of Nonparametric Predictive Inference (NPI).

2.1 Classification

In data mining, classification is a form of data analysis where a classification algorithm assigns one of a set of predefined classes or categories to new observations based on the observed values of attribute variables. Basically, a data set comprises of attribute variables, also called features or input variables $\{X_1, X_2, \ldots, X_f\}$, and a class variable, also called target or output variable C, with a finite set of labels $\{C_1, \ldots, C_K\}$. The attribute variables used in classification tasks could be categorical or continuous. Categorical attributes are either nominal (non-ordered) such as colour, gender, etc., or ordered such as low, medium and high. The continuous attributes would indicate a numerical value, rather than a categorical value, e.g.

temperature, weight of a person, age of a person, etc. A classification algorithm is built using a subset of the data called the training set. On this set, the model finds the relationship between the values of attribute variables and the values of the class variable using different techniques. The quality of the model is then verified using the remaining set of data, called the testing set. The classification task is important in many fields such as health care, banking, and economics etc. For example, in healthcare, a classification model could be used to identify persons who are at low, medium or high risk of acquiring a specific disease. Similarly, in banking, a classification model could be used to classify bank loan applications if they are safe or risky. Various classification methods have been presented in the literature to predict the value or state of a class variable. For example, Support Vector Machines, Naive Bayes, Logistic Regression, Random Forests and Classification Trees. For more details and examples about these classification methods, we refer to [57, 71].

In this thesis, we focus only on classification trees because it is one of the most popular methods used in classification tasks as it provides a number of advantages over the other classification methods. For example, it is easy to be understood and interpreted, it can deal with large data without any assumptions and it can handle different types of attribute variables.

2.2 Classification trees

A classification tree is a predictive model that provides a visual representation of the relationship between the attribute variables and the class variable. In general, a classification tree is built in the form of a tree structure that contains three main parts: a root node, internal nodes and leaf nodes. A root node; it is the topmost node in the tree which has no incoming edges. An internal node: there can be many internal nodes, each has exactly one incoming edge and two or more outgoing edges. Finally, a leaf node: it has exactly one incoming edge and no outgoing edges. In a classification tree, the internal nodes and the root node contain an attribute variable, each branch represents an outcome of the attribute variable and each leaf node

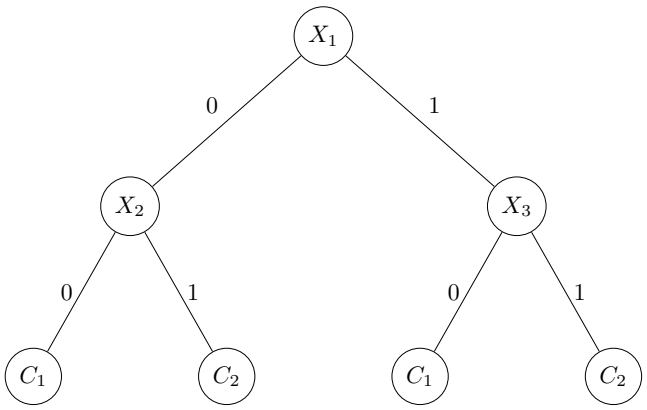


Figure 2.1: An example of a classification tree.

represents a class label. The paths from the root node to the leaf nodes denote the classification algorithm. Figure 2.1 shows an example of a classification tree with three attribute variables X_1, X_2 and X_3, each has two possible values $\{0, 1\}$, and a class variable C with two labels C_1 and C_2.

Once the classification tree has been built, it can be used to classify new observations. So, observations are classified based on their attribute variables by navigating them from the root node of the tree and moving toward the leaf node. For example, as shown in Figure 2.1, any new observations with $X_1 = X_2 = 0$ or $X_1 = 1, X_3 = 0$ would be both classified to class C_1.

2.3 Split criteria

During the process of building the classification tree, a classification algorithm requires a split criterion which is used to test all available attribute variables at each node of the tree and select the most useful one to split the data upon. The aim of using split criteria is to increase a node's purity and reduce a node's impurity. A node is 100% pure when all of its observations belong to the same class, and 100% impure when its observations are equally split between classes. Many classification tree algorithms have been developed in the literature using different split criteria. The most commonly used classic split criteria are: Information Gain, Information Gain Ratio and Gini index. These split criteria are used to implement the ID3 [66],

the C4.5 [67] and the CART [25] algorithms, respectively. In addition, we briefly review a split criterion based on imprecise probability, which is Imprecise Information Gain. In the following, we explain these split criteria.

The Information Gain (IG) is a measure which was introduced by Quinlan in 1986 as a split criterion for the ID3 algorithm [66]. It can deal only with categorical attributes. The IG is an impurity-based approach which uses entropy as an impurity measure. The formula of entropy [73], also called the Shannon Entropy, for a training data set S with a class variable C, is given by

$$H(C) = -\sum_{i=1}^{K} p_i \log_2(p_i) \tag{2.1}$$

where p_i is the proportion of the training data set S belonging to class C_i, for $i = 1, \ldots, K$. So, K is the total number of classes. Typically, the entropy represents a level of impurity or uncertainty. If all observations belong to a single class, then the entropy is equal to 0, and if all classes have equal proportions in S, then the entropy reaches its maximum value 1 [53], so $0 \leq H(C) \leq 1$. The IG of an attribute variable X with different values $\{x_1, \ldots, x_r\}$, relative to the training set S and the class variable C is given by

$$IG(C, X) = H(C) - H(C|X) \tag{2.2}$$

where $H(C|X)$ is the entropy of class C given attribute variable X, and is defined as

$$H(C|X) = \sum_{i=1}^{r} p(X = x_i) H(C|X = x_i) \tag{2.3}$$

In order to choose the best attribute variable for splitting the data at each node of the tree, the IG is used as a split criterion by the ID3 algorithm. Below we summarise the steps of building classification trees by the ID3 algorithm.

- Considering that all observations do not belong to the same class, calculate the IG (Formula 2.2) for each attribute variable in the training set S.

- Assign the attribute variable for which the IG is maximum for the root node.
- Split the training set into two or more subsets based on the values of the chosen attribute, then for each subset, repeat the process.

It has been proved that the IG split criterion is biased to attribute variables that have a large number of states, which could negatively affect the performance of the ID3 algorithm [66]. Therefore, the Information Gain Ratio (IGR) was introduced by Quinlan in 1993 [67] to overcome this weakness by using a normalization of the IG. Unlike the ID3, the C4.5 can handle both numerical and categorical attributes. In Section 2.4, we will explain how the C4.5 deals with numerical attributes. The IGR of an attribute X and a class variable C is given by:

$$IGR(C, X) = \frac{IG(C, X)}{SI(X)} \tag{2.4}$$

where $IG(C, X)$ is given by Equation (2.2), and $SI(X)$ is called the split Information, which is the entropy of the variable which does not depend on C, it is given by:

$$SI(X) = -\sum_{i=1}^{r} p(X = x_i) \log_2 p(X = x_i) \tag{2.5}$$

The value of the $SI(C, X)$ represents the information generated by splitting the training data set S into v partitions corresponding to the values of the attribute variable X. The IGR is used as a split criterion for the C4.5 algorithm [67]. As the IGR is an extension to the IG, hence, the C4.5 algorithm is an alternative version of the ID3 algorithm. In a similar method to the ID3 method, the C4.5 algorithm builds classification trees. The main difference is that the C4.5 algorithm uses the IGR (Formula 2.4) as a split criterion to select the attribute variable at each node.

The Gini Index (GI) was introduced by Breiman [25] as a split criterion for the Classification And Regression Tree (CART) algorithm. The CART algorithm uses a binary split when building trees, meaning that each internal node in the classification tree can have only two branches. In addition, the CART algorithm can work with both numerical and categorical attributes. We explain how the CART handles

numerical attributes in Section 2.4. The GI is a split criterion that measures the degree of an attribute's impurity with respect to the classes. It is defined as

$$GI(S) = 1 - \sum_{i=1}^{K} (p_i)^2 \tag{2.6}$$

where p_i is the proportion of the observations that belong to class C_i. If all observations belong to one class, then the Gini Index value becomes zero, on the other hand, if all classes are equally distributed, then the GI reaches its maximum value 0.5. After splitting the data set into two subsets, then the GI is computed as a weighted sum for each resulting split. For example, if a binary split on X divides S into S_1 and S_2, the GI of the split data is defined as

$$GI(S, X) = d_1 GI(S_1) + d_2 GI(S_2) \tag{2.7}$$

where d_1 and d_2 are the proportions of data sets in S_1 and S_2, respectively. The attribute variable that has the minimum GI value is chosen as the splitting attribute.

Finally, the split criterion Imprecise Information Gain (IIG), which is based on imprecise probability (given in Section 2.6) and general uncertainty measures was introduced by Abellán and Moral [6]. It is similar to the IG split criterion that is used in the ID3 algorithm, but the precise probabilities and entropy function have been replaced with imprecise probabilities and maximum of entropy function. The IIG for the attribute variable X is given by

$$IIG(C, X) = H^*(L(C)) - \sum_{i=1}^{r} p(X = x_i) H^*(L(C|(X = x_i)) \tag{2.8}$$

where $H^*(L)$ is the maximum entropy of a credal set, and $L(C)$ and $L(C|X = x_i)$ are credal sets for the class variable C and for C given the value x_i of the attribute variable X, respectively; and $i = 1, \ldots, n$ for a partition of the data set; and $p(X = x_i)$ is a probability distribution that belongs to the credal set of the attribute variable X, i.e. $L(X)$. Credal sets are closed and convex sets of probability distributions. Further details and explanations of the $IIG(C, X)$ and $L(.)$ are given in Section 3.5. Building the classification trees using the IIG split criterion can be done with different imprecise probability models. For example, one can use the maximum entropy

distributions from the credal set obtained from the Imprecise Dirichlet Model (IDM) [1] or from Nonparametric Predictive Inference for Multinomial data (NPI-M) [7]. These models are given in Section 2.6 and Subsection 2.7.4, respectively. The imprecise split criterion IIG was presented to compare it with our new algorithms and other classification algorithms. So, in this thesis, we refer to a classification tree created with the IIG and the IDM by the IDM algorithm, and the IIG and the NPI-M by the NPI-M algorithm.

2.4 Selecting the classification threshold

When a data set used for classification tasks contains a continuous-valued attribute variable, a classification algorithm requires a method which is used to select the optimal threshold, also known as a split point, to split the data and determine the classes. Several methods in the literature have been developed using different approaches for selecting these threshold values. This section will review the most commonly used methods in the classification tree algorithms. These methods are later compared with our new method for selecting the optimal threshold values, proposed in Chapters 3 and 4. We refer to [10, 58] for further information and explanations about these methods.

One of the most commonly used methods in the classification tree algorithms is to find the threshold value that maximises the split criterion IGR used in the C4.5 algorithm [68]. Assume that we have a training data set S and a continuous-valued attribute X with n distinct values in the ordinal sequence $\{v_1, \ldots, v_n\}$. The C4.5 algorithm uses a binary split on X to evaluate each midpoint between adjacent values v_i and v_{i+1} (for $i = 1, \ldots, n-1$), by computing the IGR, as given by Equation (2.4). So, the number of possible evaluations for X (assuming that all observations do not have tied values) is $n-1$. A threshold value that maximises the IGR criterion is selected as the optimal threshold for attribute X. After selecting the threshold value, the training data set is partitioned into two subsets based on

the threshold value. The C4.5 algorithm continues recursively by evaluating each midpoint between adjacent values for each new subset and selecting new thresholds for each branch. According to Quinlan [68], in order to ensure that any threshold value used in the classification tree is an actual value from the data set, the C4.5 algorithm chooses the largest value of X that does not exceed each of the midpoint values. I.e, the C4.5 algorithm chooses the largest value of X in the given training set that does not exceed the below interval midpoint:

$$t_i = \left\{ v \mid v \leq \frac{v_i + v_{i+1}}{2} \right\}$$

In classification tree applications, the C4.5 algorithm has been widely used in several studies for building classification trees with continuous-valued attributes. For example, in the medical field, Adri et al. [52] used the C4.5 algorithm to identify the optimal thresholds for a data set containing many continuous-valued attributes obtained from the Autonomic Nervous System (ANS) unit of University Hospital Avicenne in Morocco. The classification accuracy results were excellent, and they suggested that the C4.5 algorithm is useful in choosing the appropriate threshold for continuous-valued attributes. Mašetic and Subasi [62] used the C4.5 algorithm in building a classification tree that detects and separates normal and congestive heart failures (CHF) over a long time period. The experimental analysis showed that the C4.5 algorithm plays an important role in identifying and classifying ECG heartbeat signals with classification accuracy 99.86%. On the other hand, many researchers discussed the disadvantages of the C4.5 algorithm in selecting the optimal thresholds. For example, Fayyad and Irani [43] argued that the C4.5 algorithm does not select the optimal threshold that gives high classification accuracy for future observations, but it is a useful method for selecting the threshold value that correctly classifies the training data. Another disadvantage stated by Fayyad and Irani [43], is that the C4.5 algorithm must evaluate each attribute $n - 1$ times, which could result in a delay in the process of building the trees, particularly for large data sets. This disadvantage does not exist in our algorithms.

Another method for selecting the optimal threshold is used in the CART algorithm [25]. The CART algorithm selects the optimal threshold in a similar way to the C4.5 algorithm, which both use a binary split on X to evaluate each partition by computing the GI, as given by Equation (2.7), for all candidate thresholds. A threshold value that minimises the GI is chosen as the optimal threshold for that attribute X. Zhang et al. [80] examined the performance of the CART algorithm and compared its performance with the C4.5 algorithm using real data sets. The results showed that the CART algorithm performs better than the C4.5 algorithm for most data sets, so it is considered a suitable alternative to the C4.5 algorithm in the case of continuous attributes.

Fayyad and Irani [43, 44] showed that it is not necessary to evaluate all possible values of attribute X as done by the C4.5 and CART algorithms to find the optimal threshold, as they proved that the optimal threshold value t that maximises the information must be boundary point. For example, if v_i and v_{i+1} belong to the same class, then the midpoint between them does not give a partition that maximises the IGR criterion or minimises the GI criterion. In addition, they suggested using only the average class entropy to evaluate the partitions created by each candidate threshold. For example, for the class variable C and attribute X, if the training data set S is partitioned into two subsets, S_1 and S_2, based on a candidate threshold t value, then the average class entropy of the resulting partition is

$$H(C; X, t) = d_1 H_1(C) + d_2 H_2(C) \tag{2.9}$$

where d_i denotes the proportion of S that belongs to S_i, for $i = 1, 2$, and $H_i(C)$ is the entropy of the class variable, C, in S_i. Note that $H_i(C)$ is calculated in the same way as in $H(C)$, Formula (2.1). A threshold value t which minimises Equation (2.9) among all candidate thresholds is chosen as the optimal threshold for X. Kohavi and Sahami [56] examined the method of Fayyad and Irani [43, 44] for selecting the optimal threshold and compared it with the C4.5 algorithm's method for selecting the optimal threshold. Their results showed that Fayyad and Irani's method performed better than the method used by the C4.5 algorithm.

Several other methods for selecting the optimal threshold have been introduced in the literature, which are not based on the split criteria. However, these methods have been used less in classification tree applications than those based on the split criteria. Some of these methods are based on the Receiver Operating Characteristic (ROC) curve. The ROC curve is a mapping of Sensitivity (Sn) against 1-Specificity (Sp) over all possible threshold values $t \in \mathbb{R}$. In classification tasks, sensitivity is the ratio of true positives predicted by the algorithm, while specificity is the ratio of true negatives predicted by the algorithm. Some of the methods that select the optimal threshold based on the ROC analyses is the Youden index [46] and the Closest-to-(0,1) corner methods [26, 76]. In this thesis, we will not use these methods during the experimental analysis, so we refer to [11, 13, 46] for further details and explanations about these methods.

2.5 Evaluation metrics

Several evaluation metrics have been used to measure and compare the performance of classification trees created by different classification algorithms. One of the most commonly used metrics is the classification accuracy, which is the ratio of the number of correctly classified observations on the testing set to the total number of observations in the testing set. Given a confusion matrix, as shown in Table 2.1, which is used to provide a summary of the predictive performance of an algorithm, the classification accuracy is calculated as follows

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \tag{2.10}$$

where TP, TN, FP and FN denote true positive, true negative, false positive and false negative, respectively. Note that the accuracy as it is defined on Formula (2.10) is only for binary cases. Similarly, we can calculate the accuracy for any case by dividing the number of correct predictions (the corresponding diagonal in the confusion matrix, Table 2.1) by the total number of observations in the testing set. In practical applications, the classification accuracy is calculated using a k-fold cross-validation technique [53]. In this method, the model is trained and evaluated

	Class 1 (**Predicted**)	Class 2 (**Predicted**)
Class 1 (**Actual**)	TN	FP
Class 2 (**Actual**)	FN	TP

Table 2.1: A sample confusion matrix.

using several different subsets of the data set instead of one. The data sets are randomly divided into k subsets of approximately equal size, called folds. Each fold of the k folds is used as a testing set to evaluate the performance of the classification algorithm, and the $k-1$ remaining folds are mixed together to use as a training set. This method is performed k times so that training and testing are performed k times. In the end, the classification accuracy is computed by taking the average of the k classification accuracies attained from the k test sets. In this thesis, the 10-fold cross-validation method is used for all of the experimental analyses. We have chosen $k = 10$ because it is the most common choice in practice.

Other studies, such as [4, 22, 64], also use other metrics to evaluate the performance of the classification algorithms. Some of these metrics are in-sample accuracy and tree size. The in-sample accuracy is the classification accuracy on the training data sets. It is calculated as the ratio of correctly classified observations in the training data set to the total number of observations in the training data set. The tree size is defined as the number of leaf nodes in the tree. Note that it is referred here to the tree size as the total number of leaf nodes in the tree, as was done by Bertsimas and Dunn [22], and by Murthy and Salzberg [64]. However, some researchers may refer to the tree size as the total number of all nodes in the tree. In this thesis, we use all three metrics, which are classification accuracy, in-sample accuracy and tree size, to evaluate and compare the performance of our classification algorithms with other classification algorithms from the literature. It will be interesting to use further evaluation measures, such as sensitivity and specificity, or consider different costs associated with misclassifications, but we leave this as a topic for future research as the aim of this work is to maximise the total classification accuracy rate, not to minimise misclassification costs.

2.6 Imprecise Probability

In classical probability theory, the probability $p(A)$ that is used to quantify uncertainty about an event A is given by a single value, that is $p(A) \in [0, 1]$, where p is a probability satisfying Kolmogorov's axioms [17]. The precise probability obtained from the data in real-world data sets may not be accurate, therefore, the use of imprecise probability became an alternative approach that may give advantages over the use of precise probability. The idea of using imprecise probabilities was first proposed by Boole [23], in the middle of the nineteenth century. In recent years, many research areas in statistics have been developed based on imprecise probability. Augustin et al. [17] have presented a detailed introduction to the main aspects of imprecise probabilities theory and applications.

Unlike classical probability, in imprecise probability theory an interval probability is assigned for an event A, that is $[\underline{P}(A), \overline{P}(A)]$, where $\underline{P}(A) \in [0, 1]$ is the lower probability and $\overline{P}(A) \in [0, 1]$ is the upper probability. The classical probability value is a special case in imprecise probability with $\underline{P}(A) = \overline{P}(A)$, whereas $\underline{P}(A) = 0$ and $\overline{P}(A) = 1$ represents complete lack of knowledge about an event A. The structure, $\mathcal{M}$, for set of events $\mathcal{A}$ is defined by Weichselberger [78] as

$$\mathcal{M} = \{p(.) \mid \underline{P}(A) \leq p(A) \leq \overline{P}(A), \forall A \in \mathcal{A}\} \tag{2.11}$$

where $p(.)$ is a probability satisfying Kolmogorov's axioms. The lower and upper probabilities for an event A are:

$$\underline{P}(A) = \inf_{p(.) \in \mathcal{M}} p(A) \tag{2.12}$$

and

$$\overline{P}(A) = \sup_{p(.) \in \mathcal{M}} p(A) \tag{2.13}$$

One of the most widely used imprecise probability models is the Imprecise Dirichlet Model (IDM). This model was introduced by Walley [77] for statistical inference based on multinomial data. We use a similar notation to [77] to explain this model.

Assume that there is a data set with n observations. Let X be a variable whose values or categories belong to $\{x_1, \ldots, x_r\}$. Let n_i denote the total number of observations in x_i, for $i = 1, \ldots, r$. According to the IDM model, the lower and upper probabilities for the event that the next future observation, X_{n+1} will be in x_i are

$$\underline{P}_{IDM}(X_{n+1} = x_i) = \frac{n_i}{n + \tilde{s}} \tag{2.14}$$

$$\overline{P}_{IDM}(X_{n+1} = x_i) = \frac{n_i + \tilde{s}}{n + \tilde{s}} \tag{2.15}$$

where $\tilde{s}$ is a given parameter. Note that this parameter was written as s in [77], but we denote a different symbol because we use s for another value later in the thesis. This parameter determines the convergence speed of the lower and upper probabilities when the sample size increases [77]. Walley [77] proposed to choose the values $\tilde{s} = 1$ or $\tilde{s} = 2$, nevertheless, mainly recommends the use of $\tilde{s} = 1$. As it is presented in [1], the imprecise probability obtained by the IDM gives rise to the following credal set (closed and convex set) of probability distributions on the variable X,

$$L(X) = \left\{ p|\ p(x_i) \in \left[\frac{n_i}{n + \tilde{s}}, \frac{n_i + \tilde{s}}{n + \tilde{s}} \right], i = 1, \ldots, r, \sum_{i=1}^{r} p(x_i) = 1 \right\} \tag{2.16}$$

The IDM model has been widely used in many areas of statistics. Bernard [20] presented some of these applications. However, a number of criticisms of using the IDM model were made [65]. Some of these criticisms were discussed by Walley himself [77], and by other researchers. These disadvantages motivate researchers to develop alternative models for such inference. Coolen and Augustin [30, 31] proposed a new model for inference from multinomial data called Nonparametric Predictive Inference for Multinomial data (NPI-M). The NPI-M model, which is based on the NPI approach, does not make any previous assumptions about the data as the IDM does. In recent years, building classification trees from an imprecise probability perspective have been presented in many papers. Abellán and Moral [6] introduced one of the first applications of building classification trees using imprecise probability theory, where they presented a new split criterion based on the Imprecise Dirichlet Model. After this paper, many researchers have presented many papers in which they build classification trees from imprecise probability approach [4, 6, 61]. Several other

researches also built classification trees based on Nonparametric Predictive Inference [2, 7, 8, 18]. It has been shown by Abellán et al. [8] and Baker [18] that using the NPI-M to build classification trees gives slightly better results than the IDM in terms of the classification accuracy. In this thesis, we also build classification trees based on the Nonparametric Predictive Inference approach. We build classification trees using a new method for selecting the optimal thresholds based on NPI. We first review the Nonparametric Predictive Inference approach in Section 2.7.

2.7 Nonparametric Predictive Inference (NPI)

2.7.1 NPI for real-valued observations

Nonparametric predictive inference (NPI) is a statistical methodology based on Hill's assumption $A_{(n)}$ [49], which gives direct probabilities for one or more future observations based on n observed values of related random quantities. Inference based on $A_{(n)}$ is nonparametric and predictive. It was introduced particularly for situations where there is no prior information about the probability distribution for a random quantity of interest, or in cases where one explicitly does not want to use any such information. Let $X_1, \ldots, X_n, X_{n+1}$ be exchangeable real-valued random quantities. Suppose that the ordered observed values of $X_1, X_2, \ldots X_n$ are denoted by $x_1 < x_2 \ldots < x_n$, where the assumption is made that there are no ties between observations. For ease of notation, let $x_0 = -\infty$ and $x_{n+1} = \infty$. Note that $x_{n+1} = \infty$ is not an observation of the variable X_{n+1}. These n ordered observations divide the real-line into $n+1$ open intervals $I_j = (x_{j-1}, x_j)$, for $j = 1, 2, \ldots, n+1$. The assumption $A_{(n)}$ states that the future observation X_{n+1} falls equally likely in any interval (x_{j-1}, x_j), for each $j = 1, 2, \ldots, n+1$,

$$P(X_{n+1} \in I_j) = \frac{1}{n+1} \tag{2.17}$$

It is important to emphasize that Hill's assumption $A_{(n)}$ does not make any further assumptions on the distribution of probability $\frac{1}{n+1}$ within an interval I_j.

NPI uses $A_{(n)}$ for predictive inferences about future observations in the form

of lower and upper probabilities, also known as imprecise probabilities. Augustin and Coolen [16] introduced predictive lower and upper probabilities for events of interest based on assumption $A_{(n)}$, which is essentially an application of De Finetti's fundamental theorem of probability [39]. The lower and upper probabilities for the event $X_{n+1} \in B$, with $B \subset \mathbb{R}$, based on the intervals I_j, $j = 1, 2, \ldots, n+1$, and Hill's assumption $A_{(n)}$, are given by

$$\underline{P}(X_{n+1} \in B) = \frac{1}{n+1} \sum_{j=1}^{n+1} \mathbf{1}\{I_j \subseteq B\} \tag{2.18}$$

$$\overline{P}(X_{n+1} \in B) = \frac{1}{n+1} \sum_{j=1}^{n+1} \mathbf{1}\{I_j \cap B \neq \emptyset\} \tag{2.19}$$

where $\mathbf{1}\{A\}$ is equal to 1 if A is true and equal to 0 else. The NPI lower probability (2.18) is obtained by taking only probability mass into account that is necessary within B, which is only the case for the probability mass $\frac{1}{n+1}$ per interval I_j if this interval is totally contained within B. The NPI upper probability (2.19) is obtained by taking all probability mass into account that could possibly be within B, which is the case for the probability mass $\frac{1}{n+1}$ per interval I_j if the intersection of I_j and B is non-empty. NPI has been introduced for a wide range of applications such as survival analysis, reliability testing, finance and topics in operational research [29]. In addition, NPI has been introduced for different types of data, including multinomial data [18, 30], ordinal data [42], and right-censored data [33]. The following subsections briefly present some NPI results that will be used later in the thesis, such as NPI for multiple future observations, NPI for Bernoulli data and NPI for multinomial data. For further details about NPI, we refer to `www.npi-statistics.com`.

2.7.2 NPI for multiple future observations

Section 2.7.1 summarised NPI for one future observation, but NPI has also been introduced for $m \geq 1$ future observations, X_{n+i} for $i = 1, \ldots, m$ [35]. It is important to emphasize that the future observations X_{n+i} are assumed to derive from the same data collection process as the n data observations. The data and future observations are linked by consecutive application of $A_{(n)}, A_{(n+1)}, \ldots, A_{(n+m-1)}$ [49].

These together are referred to as the $A_{(.)}$ assumptions, which can be considered a post-data version of a finite exchangeability assumption for $n+m$ random quantities. The $A_{(.)}$ assumptions imply that all possible orderings of n data observations and m future observations are equally likely, where the n data observations are not distinguished among each other and neither is the m future observations. Let $S_j = \#\{X_{n+i} \in I_j, i = 1, \ldots, m\}$, then assuming $A_{(.)}$ we have [35]

$$P(\bigcap_{j=1}^{n+1}\{S_j = s_j\}) = \binom{n+m}{n}^{-1} \tag{2.20}$$

where s_j are non-negative integers with $\sum_{j=1}^{n+1} s_j = m$. Equation (2.20) implies that all $\binom{n+m}{n}$ orderings of m future observations among the n observations are equally likely. Let $X_{(r)}$, for $r = 1, \ldots, m$, be the r-th ordered future observation, so $X_{(r)} = X_{n+i}$ for one $i = 1, \ldots, m$ and $X_{(1)} < X_{(2)} < \cdots < X_{(m)}$. The probabilities given in Equation (2.21) are based on Equation (2.20) and derived by counting the relevant orderings, and hold for $j = 1, \ldots, n+1$, and $r = 1, \ldots, m$ [35],

$$P(X_{(r)} \in I_j) = \binom{j+r-2}{j-1}\binom{n-j+1+m-r}{n-j+1}\binom{n+m}{n}^{-1} \tag{2.21}$$

For this $X_{(r)} \in I_j$, NPI gives a precise probability, as each of the $\binom{n+m}{n}$ equally likely orderings of n past and m future observations has the r-th ordered future observation in precisely one interval I_j [32]. Following Equations (2.18) and (2.19) in Subsection 2.7.1, the NPI lower and upper probabilities for the event $X_{(r)} \in B$, where $B \subset \mathbb{R}$ are derived as follows:

$$\underline{P}(X_{(r)} \in B) = \sum_{j=1}^{n+1} \mathbf{1}\{I_j \subseteq B\} P(X_{(r)} \in I_j) \tag{2.22}$$

$$\overline{P}(X_{(r)} \in B) = \sum_{j=1}^{n+1} \mathbf{1}\{I_j \cap B \neq \emptyset\} P(X_{(r)} \in I_j) \tag{2.23}$$

The event that the number of future observations in an interval (x_α, x_β), with $1 \leq \alpha < \beta \leq n+1$, and denoted by $S^m_{\alpha,\beta}$, is greater than or equal to a particular value $v \in \mathbb{N}$, has the following precise probability [13],

$$P(S^m_{\alpha,\beta} \geq v) = \sum_{i=v}^{m} \binom{n+m}{n}^{-1} \binom{\beta-\alpha-1+i}{i}\binom{n-\beta+\alpha+m-i}{m-i} \tag{2.24}$$

The result of NPI for multiple future observations presented in this subsection is important in this thesis which will be used in Chapters 3 and 4, for selecting the optimal thresholds.

2.7.3 NPI for Bernoulli quantities

Coolen [27] presented NPI for Bernoulli quantities (NPI-Bern) which will be used in Chapter 3. NPI-Bern is based on the assumptions $A_{(.)}$ for m future observations given n observed data, and a latent variable representation of Bernoulli data. A latent variable represents the data on the real-line, with a threshold such that data to one side are successes and to the other side are failures. Suppose there is a sequence of $n+m$ exchangeable Bernoulli trials where the possible outcomes of each trial are binary, e.g. 'success' or 'failure,' and the data set consists of s successes in n trials. Let Y_1^n be the random number of successful outcomes in trials 1 to n and let Y_{n+1}^{n+m} be the random number of successful outcomes in trials $n+1$ to $n+m$. Due to the assumption that all trials are exchangeable, a sufficient representation of the data for the inferences considered is $Y_1^n = s$. As we are using in this thesis the case of only one future observation, i.e. $m = 1$, the NPI lower and upper probabilities for this case are

$$\underline{P}(Y_{n+1}^{n+1} = 1 | Y_1^n = s) = \frac{s}{n+1} \tag{2.25}$$

and

$$\overline{P}(Y_{n+1}^{n+1} = 1 | Y_1^n = s) = \frac{s+1}{n+1} \tag{2.26}$$

The NPI lower and upper probabilities for general events involving m future Bernoulli quantities, i.e. $m > 1$, are given in [27]. Further details and examples about the NPI for Bernoulli quantities can be found in [9, 11, 27, 38].

2.7.4 NPI for multinomial data

Coolen and Augustin [16, 30, 31] introduced a new model for inference from multinomial data, which is called Nonparametric Predictive Inference for Multinomial data (NPI-M). The NPI-M model was developed based on the circular-$A_{(n)}$ assumption, which is different from Hill's assumption $A_{(n)}$ [30]. Because multinomial data are represented as observations on a probability wheel rather than a real-line, we use the assumption made by Coolen and Augustin [28, 30], which is the circular-$A_{(n)}$ assumption. The NPI-M model has been introduced for the case of unknown number of classes [30] and for the case of known number of classes [31]. In this thesis, we only consider the situation where the number of classes, denoted by K, is known.

Suppose that there are $K \geq 3$ possible different classes, denoted by $C_1, \ldots, C_K$. It is possible to use the NPI-M model for the case $K = 2$, however, for Bernoulli data, it is better to use NPI for Bernoulli data, as presented in Section 2.7.3, because it gives slightly less imprecision. We assume that $C_1, \ldots, C_k$, for $1 \leq k \leq K$, are the observed classes, and the $C_{k+1}, \ldots, C_K$ are unobserved classes. Suppose that there are n_j observations in class C_j, for $j = 1, \ldots, k$, and that $\sum_{j=1}^{k} n_j = n$. The general event of interest can be represented in the following way:

$$E = Y_{n+1} \in \bigcup_{j \in J} C_j \tag{2.27}$$

where $J \subseteq \{1, \ldots, K\}$. Let $OJ = J \bigcap \{1, \ldots, k\}$ represent the index-set for the classes in the event of interest that have been observed and let $UJ = J \bigcap \{k + 1, \ldots, K\}$ represent the index-set for the unobserved classes. Here $r = |OJ|$ and $l = |UJ|$, hence $0 \leq r \leq k$ and $0 \leq l \leq K - k$. According to Coolen and Augustin [31], the NPI-M lower and upper probabilities for the event of interest in (2.27) based on n observations are

$$\underline{P}(E) = \frac{n_j - \min(K - r - l, r)}{n} \tag{2.28}$$

and

$$\overline{P}(E) = \frac{n_j + \min(r + l, k - r)}{n} \tag{2.29}$$

where n_j is the number of observations in class C_j, for $j = 1, \ldots, k$. For further explanation of the derivation of the NPI-M lower and upper probabilities, we refer to [31]. Also, some basic properties are presented in [31].

For the singleton events, $Y_{n+1} \in C_j$, the NPI-M lower and upper probabilities are

$$\underline{P}\left(Y_{n+1} \in C_j\right) = \max\left(0, \frac{n_j - 1}{n}\right) \tag{2.30}$$

and

$$\overline{P}\left(Y_{n+1} \in C_j\right) = \min\left(\frac{n_j + 1}{n}, 1\right) \tag{2.31}$$

It should be clarified that events with only observed classes are generally considered for classification tasks. Therefore, in this thesis, we will only consider the case that these singleton events have been observed, i.e. $n_j > 0$.

The NPI-M lower and upper probabilities, for unobserved classes, are

$$\underline{P}\left(Y_{n+1} \in C_j\right) = 0 \tag{2.32}$$

and

$$\overline{P}\left(Y_{n+1} \in C_j\right) = \frac{1}{n}. \tag{2.33}$$

In this chapter, we have presented the main concepts from the literature to provide background information for the topic considered in this thesis. In the next chapter, we introduce a new method for building classification trees using the NPI approach for selecting the optimal thresholds. This method is different from the classical methods in that it selects the threshold value using predictive inference. We first introduce this method for building binary classification trees in Chapter 3; we then extend it to build classification trees with three ordered classes in Chapter 4.

Chapter 3

NPI-based binary classification tree

3.1 Introduction

In this chapter, we present a new method based on the NPI approach for building classification trees for data containing continuous-valued attributes and a binary class variable, while in Chapter 4, we extend this method to data with continuous-valued attributes and a class variable with three labels. Data sets with continuous attributes and a binary class variable frequently occur in many real-world applications. For example, in healthcare, individuals may be classified into one of two classes, healthy or diseased, based on the body temperature results. The critical point in building binary classification trees involving continuous-valued attributes is to select the optimal threshold values that are used to determine the classes and split the data. The wrong selection of such classification threshold values generally results in two types of misclassification: observations from class C_1 may be classified as class C_2, and observations from class C_2 may be classified as class C_1. A threshold value is considered perfect if both classes are correctly classified. Alabdulhadi [11] and Coolen-Maturi et al. [37] introduced the NPI approach for selecting the optimal threshold for the two-class classification problem, where the inference is based on a given number of future observations. A direct criterion for introducing the relative importance of the two classes has been presented. This relative importance is

referred to as target proportions, providing weights that are selected to reflect the desired importance of one class over another.

In this chapter, we present a new classification algorithm for building binary classification trees using the NPI approach for selecting the thresholds, which we call NPI_2-Tree algorithm. The NPI_2-Tree algorithm is a novel method that builds classification trees by employing the NPI approach for selecting the threshold values for data sets with continuous attributes and a binary class variable using predictive inference. To build classification trees using our classification method, we introduce a new procedure for selecting the optimal values of target proportions by choosing that to maximise classification accuracy on testing data sets. In order to evaluate the performance of the NPI_2-Tree algorithm, we conduct an experimental analysis using different evaluation measures on several data sets. We also compare the performance of the NPI_2-Tree algorithm with other classification algorithms. In addition, we evaluate the performance of the NPI_2-Tree algorithm using a different split criterion based on imprecise probability, and the results are compared with the classical split criterion.

This chapter is organised as follows: Section 3.2 summarises the NPI method for selecting the optimal threshold for the two-class classification problem [11, 37], and illustrates the method with an example. In Section 3.3, we present our method for building classification trees using the NPI apprach for selecting optimal thresholds. First, we present the new procedure for choosing the optimal values of target proportions in classification trees. Then, we present the new algorithm, the NPI_2-Tree algorithm, which is used to build classification trees for data with two classes. We also provide an illustrative example to describe the process of building classification trees based on the NPI_2-Tree algorithm. In Section 3.4, we conduct an experiment analysis on several data sets to evaluate the performance of the NPI_2-Tree algorithm using a classical split criterion, and compare its performance with other classification algorithms. In Section 3.5, we also present the performance of the NPI_2-Tree algorithm when it is used to build classification trees using an imprecise split crite-

rion. Finally, some concluding remarks are given in Section 3.6.

3.2 NPI-based thresholds for two classes

This section presents the NPI method for selecting the optimal threshold t for a real-valued random quantity and for a two-class classification scenario as introduced in [11, 37]. This method is different from the classical methods as it selects the optimal threshold value which focuses on a number of future observations to which the threshold will be applied. Assume that we have a continuous random variable X whose values belong to two classes, C_1 and C_2, and small values of X are more likely to belong to class C_1, i.e. $X \leq t$, and large values of X are more likely to belong to class C_2, i.e. $X > t$. Let n_1 denote the number of observations for class C_1 and n_2 the number of observations for class C_2. It is assumed that there is full independence between the two classes, meaning that any information about the observations in one class does not contain information about the observations in the other class. In other words, Any information about random quantities from one class does not affect any (lower and upper) probabilities for events involving only random quantities of the other class. Let $x_1^1 < x_2^1 < \cdots < x_{n_1}^1$ denote the ordered data from class C_1 and $x_1^2 < x_2^2 < \cdots < x_{n_2}^2$ denote the ordered data from class C_2. For ease of notation, let $x_0^1 = x_0^2 = -\infty$ and $x_{n_1+1}^1 = x_{n_2+1}^2 = \infty$. The data for C_1 partition the real-line into $n_1 + 1$ intervals $I_i^1 = (x_{i-1}^1, x_i^1)$, for $i = 1, 2, \ldots, n_1 + 1$, and the data for C_2 partition the real-line into $n_2 + 1$ intervals, for $I_j^2 = (x_{j-1}^2, x_j^2)$, for $j = 1, \ldots, n_2 + 1$. Throughout this thesis, it is assumed that there are no ties between the data observations, which occur when two or more observations have the same data. We can break the ties by adding a small amount to the tied observations, which tend to be zero. This is a popular method for breaking ties in statistics [50].

As the NPI inferences are based on multiple future observations, we consider m_1 future observations from class C_1, with data values denoted by $X_{n_1+r}^1$, where $r = 1, \ldots, m_1$, and m_2 future observations from class C_2, with data values denoted

by $X^2_{n_2+s}$, where $s = 1, \ldots, m_2$. Let the m_1 ordered future observations from classes C_1 be denoted by $X^1_{(1)} < X^1_{(2)} < \cdots < X^1_{(m_1)}$, and the m_2 ordered future observations from class C_2 be denoted by $X^2_{(1)} < X^2_{(2)} < \cdots < X^2_{(m_2)}$. As the NPI-based inferences are in terms of multiple future observations, Alabdulhadi [11] and Coolen-Maturi et al. [37] have selected the threshold value t that gives the best classification based on the m_1 and m_2 future observations. To this end, the result of NPI for multiple future observations presented in Section 2.7.2 is used, but we need first to introduce further notation.

For a particular value of t, we denote the number of correctly classified future observations from class C_1 by L^1_t, that is those with data values $X^1_{n_1+r} \leq t$, for $r = 1, \ldots, m_1$, and we denote the number of correctly classified future observations from class C_2 by L^2_t, that is those with data values $X^2_{n_2+s} > t$, for $s = 1, \ldots, m_2$. Let a and b be any two values in the range (0, 1] that are chosen to represent the relative importance of the correct classification of each one of the classes. We consider the general event of interest that the number of correctly classified future observations from class C_1 is at least am_1 and the number of correctly classified future observations from class C_2 is at least bm_2, that is $L^1_t \geq am_1$ and $L^2_t \geq bm_2$. Note that choice of a and b depends on a person's beliefs of which class may be more important to be correctly classified than another. For example, in medicine, one can choose a and b to reflect that giving a drug to patients is crucial, while it may hvae no serious harmful effects on healthy people. In such a case the target proportion of correct classifications of diseased people should be greater than for healthy people. It should be noted that these values a and b are the target proportions per class, hence the values of these proportions are not constrained except that they must be in (0, 1). Of course one can choose a and b to be equal if one gives the same importance of correct classifications to both classes, but choosing large or small values for a and b may not change classification performance; this will be illustrated in Example 3.3.1. In this thesis, we will present a new procedure for choosing the optimal values of target proportions a and b by optimising classification performance on test data; this will be given in Sections 3.3.1 and 4.3.1. It shoud be clarified that there is no

conflict between saying these values represent one's beliefs of which class may be important to correctly classify, and choosing these by optimisation. The first approach is useful if one would prefer to give one class more importance than another, whereas the second approach works if one aims to maximise the total classification accuracy.

Because of the independence assumption of the two classes, the joint NPI lower and upper probabilities are derived as the products of the NPI corresponding lower and upper probabilities for the events that involve L_t^1 or L_t^2 as follows [11, 37]

$$\underline{P}\left(L_t^1 \geq am_1, L_t^2 \geq bm_2\right) = \underline{P}\left(L_t^1 \geq am_1\right) \times \underline{P}\left(L_t^2 \geq bm_2\right) \tag{3.1}$$

$$\overline{P}\left(L_t^1 \geq am_1, L_t^2 \geq bm_2\right) = \overline{P}\left(L_t^1 \geq am_1\right) \times \overline{P}\left(L_t^2 \geq bm_2\right) \tag{3.2}$$

The NPI lower and upper probabilities in Equations (3.1) and (3.2) are derived using NPI for multiple future observations as given in Section 2.7.2, in particular Equation (2.21), as shown below. It is noticed that the event $L_t^1 \geq am_1$ is equal to $X^1_{(\lceil am_1 \rceil)} \leq t$, where $\lceil am_1 \rceil$ denotes the smallest integer greater than or equal am_1. Similarly, the event $L_t^2 \geq bm_2$ is equal to $X^2_{(m_2 - \lceil bm_2 \rceil + 1)} > t$. To show how to use the Equation (2.21) of the NPI results for multiple future observations, we first consider class C_1 and then class C_2.

For $I_i^1 = (x_{i-1}^1, x_i^1), i = 1, \ldots, n_1 + 1$, and $t \in I_{i_t}^1 = (x_{i_t-1}^1, x_{i_t}^1)$, where $i_t = 2, \ldots, n_1$ is defined as that interval $I_{i_t}^1$ which contains t, the NPI lower and upper probabilities for the event $L_t^1 \geq am_1$ are given as follow [11, 37]:

$$\underline{P}(L_t^1 \geq am_1) = \underline{P}(X^1_{(\lceil am_1 \rceil)} \leq t) = \sum_{i=1}^{i_t - 1} P(X^1_{(\lceil am_1 \rceil)} \in I_i^1) \tag{3.3}$$

$$\overline{P}(L_t^1 \geq am_1) = \overline{P}(X^1_{(\lceil am_1 \rceil)} \leq t) = \sum_{i=1}^{i_t} P(X^1_{(\lceil am_1 \rceil)} \in I_i^1) \tag{3.4}$$

where the precise probabilities on the right-hand sides of Equations (3.3) and (3.4) are obtained from Equation (2.21). For $i_t = 1$, we have $\underline{P}(L_t^1 \geq am_1) = 0$ and $\overline{P}(L_t^1 \geq am_1) = P(X^1_{(\lceil am_1 \rceil)} \in I_1^1)$, and for $i_t = n_1 + 1$, we have $\underline{P}(L_t^1 \geq am_1) = 1 -$

$P(X^1_{(\lceil am_1 \rceil)} \in I^1_{n_1+1})$ and $\overline{P}(L^1_t \geq am_1) = 1$. If t is equal to one of the observations x^1_i, i.e. $t = x^1_{i_t}$, then this event has precise probability,

$$P(L^1_t \geq am_1) = P(X^1_{(\lceil am_1 \rceil)} \leq t) = \sum_{i=1}^{i_t} P(X^1_{(\lceil am_1 \rceil)} \in I^1_i) \tag{3.5}$$

Of course, this implies that we have for such a value of t that $\underline{P}(L^1_t \geq am_1) = \overline{P}(L^1_t \geq am_1) = P(L^1_t \geq am_1)$, in this case. Similarly, the NPI lower and upper probabilities for the event $L^2_t \geq bm_2$ are derived. For $I^2_j = (x^2_{j-1}, x^2_j), j = 1, \ldots, n_2 + 1$, and $t \in I^2_{j_t} = (x^2_{j_t-1}, x^2_{j_t}), j_t = 2, \ldots, n_2$, the NPI lower and upper probabilities for the event $L^2_t \geq bm_2$ are

$$\underline{P}(L^2_t \geq bm_2) = \underline{P}(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t) = \sum_{i=j_t+1}^{n_2+1} P(X^2_{(m_2-\lceil bm_2 \rceil+1)} \in I^2_j) \tag{3.6}$$

$$\overline{P}(L^2_t \geq bm_2) = \overline{P}(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t) = \sum_{i=j_t}^{n_2+1} P(X^2_{(m_2-\lceil bm_2 \rceil+1)} \in I^2_j) \tag{3.7}$$

For $j_t = 1$, we have

$$\underline{P}(L^2_t \geq bm_2) = 1 - P(X^2_{(m_2-\lceil bm_2 \rceil+1)} \in I^2_1) \text{ and } \overline{P}(L^2_t \geq bm_2) = 1.$$

And for $j_t = n_2 + 1$, we have

$$\underline{P}(L^2_t \geq bm_2) = 0 \text{ and } \overline{P}(L^2_t \geq bm_2) = P(X^2_{(m_2-\lceil bm_2 \rceil+1)} \in I^2_{n_2+1})$$

Furthermore, when $t = x^2_{j_t}$

$$P(L^2_t \geq bm_2) = P(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t) = \sum_{i=j_t+1}^{n_2+1} P(X^2_{(m_2-\lceil bm_2 \rceil+1)} \in I^2_j) \tag{3.8}$$

so

$$\underline{P}(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t) = \overline{P}(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t) = P(X^2_{(m_2-\lceil bm_2 \rceil+1)} > t), \text{ if } t = x^2_{j_t}.$$

Now, we can obtain the optimal threshold t for the two classes by maximising either Equation (3.1) for the lower probability or Equation (3.2) for the upper probability. To search for the optimal threshold value t, one does not need to go through each of the $n_1 + n_2 + 1$ intervals produced by the data observations. The optimal threshold t can be only in intervals where the left-end point of the interval is an

observation that belongs to class C_1 and the right-end point of the interval is an observation that belongs to class C_2 [11, 37]. It is important to clarify that the NPI lower and NPI upper probabilities, Equations (3.1) and (3.2), may lead to different optimal thresholds because they are different criteria. In this thesis, we consider only the optimal threshold value, which is based on the NPI lower probability in Equation (3.1), for building classification trees. This is because the NPI lower probabilities are based on evidence in favour of events while the NPI upper probabilities are based on evidence against events. Next, we provide an example illustrating the NPI method for selecting the optimal threshold. For more details, examples and discussions of NPI for selecting the optimal thresholds, we refer to [11, 37].

Example 3.2.1 Assume that we have a data set of 20 people, where 10 people from class C_1, i.e. $n_1 = 10$, and 10 people from class C_2, i.e. $n_2 = 10$. Suppose the data set that belong to class C_1 are {25, 27, 28, 29, 30, 36, 37, 40, 63, 68 } and the data set that belong to class C_2 are {48, 53, 67, 70, 73, 75, 82, 86, 89, 90}. In Figure 3.1, we show the box plots for the data sets from the two classes. As we can see from the figure, there is little overlap between the two classes. In order to illustrate the NPI method for selecting the threshold values, we have presented the NPI method for $m_1 = m_2$ and for $m_1 \neq m_2$, and we have considered four different scenarios for target proportions a and b. Note that, these target proportions are previously chosen to represent the relative importance of the correct classification of one class over another. However, in Section 3.3.1, we will present a new procedure for choosing these values in classification trees.

To select the optimal threshold t, we need to search within each of the n_1+n_2+1 intervals created by the data observations and then we choose the value t that maximises the NPI lower probabilities method, Equation (3.1), or the NPI upper probabilities method, Equation (3.2). Note that as shown in [11, 37], the optimal threshold value t can only be found in intervals in where the left-end point of the interval is an observation that belongs to class C_1 and the right-end point is an observation that belongs to class C_2. The first and last intervals should also be considered. Thus, we

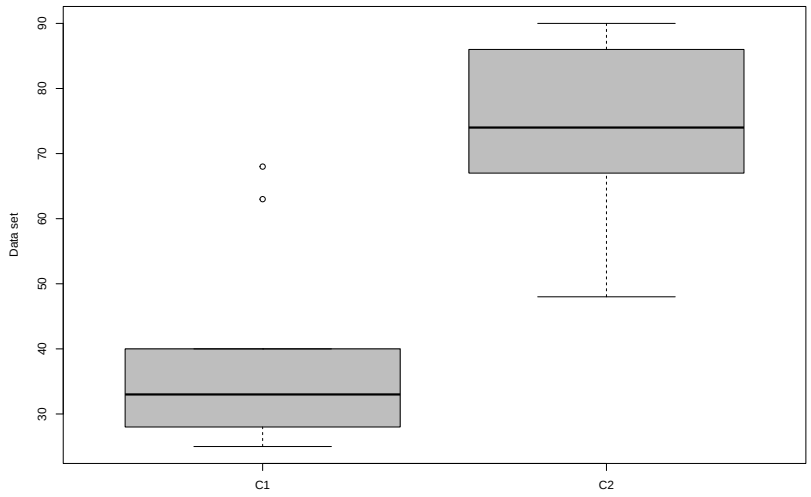


Figure 3.1: Box-plots for the data set used in Example 3.2.1.

just consider the intervals as shown in [11, 37].

Table 3.1 presents the optimal threshold value t obtained from the NPI lower probabilities method, Equation (3.1), and the NPI upper probabilities method, Equation (3.2), along with the corresponding lower and upper probabilities, for $m_1 \neq m_2$, while Table 3.2 presents the optimal threshold value t obtained from the NPI lower and upper probabilities method along with the corresponding lower and upper probabilities, for $m_1 = m_2$. As shown in Table 3.1, the optimal threshold value t differs for different values of a and b. In Scenario 1, for $a = b = 0.25$, the optimal threshold value is $t = 40$. In this scenario, the values of lower and upper probabilities for the NPI method are quite high because the required proportions seem easy to achieve. In Scenario 2, we put more emphasis on the number of correctly classified future observations from class C_1 than from class C_2, that is $a = 0.50$ and $b = 0.25$. As we can see from Table 3.1, the optimal threshold value increased to $t = 63$ in order to optimise the probability, compared to Scenario 1. For Scenario 3, for $a = 0.85$ and $b = 0.20$, we again need this scenario to give a higher proportion of correctly classified future observations from class C_1 than from class C_2. The optimal threshold value increases to $t = 68$ compared to Scenario 2, while the corresponding NPI lower probability is less than in Scenario 2, indicating

that these proportions a and b are harder to be jointly achieved. In Scenario 4, for $a = b = 0.70$, the optimal threshold value is the same as for Scenario 1, where we have large values of $a = b$. Of course, the corresponding values of the NPI lower and upper probabilities are lower than from those in Scenario 1 as a and b here are larger. For $m_1 = 100$ and $m_2 = 80$, with respect to the optimal threshold, the optimal thresholds are found to be the same as for $m_1 = 4$ and $m_2 = 6$ regardless of the values of a and b, except for $a = 0.50$ and $b = 0.25$, the optimal threshold is $t = 68$. The corresponding lower and upper probabilities are very high compared to $m_1 = 4$ and $m_2 = 6$.

The results of the NPI method for selecting the optimal threshold value t, as well as their corresponding NPI lower and upper probabilities for $m_1 = m_2$, are presented in Table 3.2. We have used the same scenarios as in Table 3.1. For $m_1 = m_2 = 8$, the results are quite similar to the results in Table 3.1. With respect to the optimal threshold, we found that the optimal threshold for $a = b = 0.25$, $a = 0.85$ and $b = 0.20$, and $a = b = 0.70$ are the same in both tables. The corresponding NPI lower and upper probabilities in Table 3.2 are slightly larger than in Table 3.1. A change in the NPI lower and upper probabilities is here due to the nature of the event we consider, for example, for $m_2 = 6$ and $m_2 = 8$, and $b = 0.25$, we need at least 2 good classifications in both cases, which is easier for $m_2 = 8$ than for $m_2 = 6$, so then for the latter, the lower and upper are probabilities smaller. We can also note from Tables 3.1 and 3.2 that the NPI lower and the upper methods provide the same optimal threshold regardless of the a and b values considered, which is likely because the data sets used in Example 3.2.1 are small with little overlapping. For $m_1 = m_2 = 100$, the results are the same as for $m_1 = 100, m_2 = 80$, given in Table 3.1, for all scenarios of a and b.

To summarise, we have presented the NPI method for selecting the optimal threshold for a continuous-valued random quantity and for two classes, with illustrative example. In the next section, we employ this method to build classification trees. Due to the predictive nature of the NPI method for selecting the threshold

Scenario	Target proportions		NPI lower method		NPI upper method	
#	a	b	t	value	t	value
$m_1 = 4$, $m_2 = 6$						
1	0.25	0.25	40	0.98	40	0.99
2	0.50	0.25	63	0.93	63	0.98
3	0.85	0.20	68	0.67	68	0.97
4	0.70	0.70	40	0.28	40	0.51
$m_1 = 100$, $m_2 = 80$						
1	0.25	0.25	40	0.99	40	1.00
2	0.50	0.25	68	0.99	68	1.00
3	0.85	0.20	68	0.80	68	0.99
4	0.70	0.70	40	0.60	40	0.84

Table 3.1: Optimal threshold t and corresponding value of the NPI lower and upper probabilities, for $m_1 \neq m_2$.

value, it is well suitable for classification trees, as the nature of classification trees is explicitly predictive as well.

3.3 NPI-based binary classification trees

In this section, we consider a new method for building binary classification trees using the NPI approach for selecting the optimal threshold value presented in Section 3.2. As clarified earlier, the NPI method determines the optimal threshold in a predictive way, considering a number of future observations and the target proportion values a and b, without adding any further assumptions or information. Note that maximising the NPI lower probability, Equation (3.1), and maximising the NPI upper probability, Equation (3.2), are different criteria that may give different optimal thresholds for a particular attribute. In this chapter, we only consider the optimal threshold value based on the NPI lower probability, as given in Equation (3.1).

In order to use the NPI approach for selecting the optimal threshold, we need to set the values of target proportions a and b for each class, respectively. However, choosing small or high values for a and b is not recommended, as will be illustrated

Scenario	Target proportions		NPI lower method		NPI upper method	
#	a	b	t	value	t	value
$m_1 = 8, m_2 = 8$						
1	0.25	0.25	40	0.99	40	1.00
2	0.50	0.25	68	0.97	68	0.99
3	0.85	0.20	68	0.79	68	0.99
4	0.70	0.70	40	0.31	40	0.58
$m_1 = m_2 = 100$						
1	0.25	0.25	40	0.99	40	1.00
2	0.50	0.25	68	0.99	68	1.00
3	0.85	0.20	68	0.80	68	0.99
4	0.70	0.70	40	0.60	40	0.84

Table 3.2: Optimal threshold t and corresponding value of the NPI lower and upper probabilities, for $m_1 = m_2$.

and discussed in Example 3.3.1. Researchers should choose wisely these target proportions for their analysis. This section introduces a new method for choosing the optimal a and b values in classification trees. We need first to introduce further notation before introducing the proposed method of choosing the values a and b.

Assume that we have a data set $\mathcal{D}$, which has continuous-valued attribute variables $\{X_1, \ldots, X_f\}$ and a binary class variable C, where $C = C_1$ or $C = C_2$. Throughout this thesis, we assume that the two classes are fully independent, meaning that information about observations in one class does not contain information about observations in the other. In addition, we assume that there are no ties between observations, and we can break the ties by adding a small amount to the tied observations. We divide the data set $\mathcal{D}$ into two subsets: training set S and testing set T. For the training set S, let n_1 represent the total number of observations that belong to class C_1, and n_2 the total number of observations that belong to class C_2, where $n = n_1 + n_2$. As the NPI-based inferences are in terms of future observations, and because we do not actually know the number of future observations and the class to which future observations belong, we set the values of the m_1 and m_2 future observations based on the total number of observations in the training data set.

Thus, we set m_1 equal to the number of observations that belong to class C_1 in S, i.e. $m_1 = n_1$, and m_2 equal to the number of observations that belong to class C_2 in S, i.e. $m_2 = n_2$.

Example 3.3.1 Assume that we have a data set of 22 observations, which belong to two classes. The data has been divided into two sets: training set S with $n_1 = 7$ and $n_2 = 8$, consists of data {25, 28, 36, 37, 48, 63, 68} and {53, 67, 70, 73, 75, 82, 86, 89}, respectively. The testing set T consists of data {36, 52, 65} and {70, 82, 86, 89}, respectively. In Table 3.3, we present the optimal thresholds and the classification accuracy for different scenarios of predefined target proportions a and b. These optimal thresholds maximise the NPI lower probability in Equation (3.1) with $m_1 = 7$ and $m_2 = 8$. The classification accuracy is the percentage of correctly classified observations in the testing set. As can be seen from Table 3.3, for the first scenario, $a = b = 0.33$, the optimal threshold value is $t = 63$ and the classification accuracy of the prediction is 85.7 %. For the second scenario, the target proportions $a = b = 0.85$ are very high, however, the optimal threshold is the same as in the first scenario and therefore the classification accuracy is the same of course. In the third scenario, a and b are very close to the training data proportions, that is $a = 0.45$ and $b = 0.50$. The optimal threshold value is now $t = 68$ and the classification accuracy has become 100%.

It is clear from the above results that the optimal threshold t can change depending on the values of a and b, which also leads to a change in classification performance. Choosing small or large values for a and b might not change classification performance as presented in Example 3.3.1, Table 3.3, where the NPI method gives the same threshold values for the first two scenarios. However, when we have chosen the values of the target proportions very close to the data proportions, as in the third scenario, the optimal threshold and accuracy changed. Note that there was no specific reason for choosing large or small values of the target proportions or choosing these close to the data proportions, but we show what can be done. Therefore,

Scenario	Target proportions		Threshold	Accuracy
#	a	b	t	(%)
1	0.33	0.33	63	85.7
2	0.85	0.85	63	85.7
3	0.45	0.50	68	100

Table 3.3: Comparisons of different scenarios of the a and b fixed.

setting meaningful target proportions for selecting the optimal threshold values for classification trees should be considered. This will be presented in the next section.

3.3.1 Selecting the target proportions

Instead of prefixing the target proportions a and b, as in Example 3.3.1, in this section we introduce a data-driven approach for selecting a and b, which aims to improve the classification performance. Consider the NPI method for selecting the optimal threshold which is based on the NPI lower probability, Equation (3.1),

$$\underline{P}(L_t^1 \geq am_1, L_t^2 \geq bm_2) = \underline{P}(L_t^1 \geq am_1) \times \underline{P}(L_t^2 \geq bm_2)$$

where $\underline{P}(L_t^1 \geq am_1)$ and $\underline{P}(L_t^2 \geq bm_2)$ are given in Equations (3.3) and (3.6), respectively, and $a, b \in (0, 1]$. We now consider a and b as parameters instead of desirable target proportions, and we aim to choose values for a and b that maximise classification performance. The main questions are how to find or select these values a and b, and how to validate their performance in classification trees. To this end, we suggest to use two stages of the k-fold cross-validation (k-fold CV) procedure, also known as double k-fold cross-validation [74]. This procedure enables us to train our classification method in which the values of a and b also need to be optimised. Without this procedure, one can use the same data for finding the optimal values of a and b and simultaneously evaluate their performance, which may result in a biased evaluation of the algorithm [74].

Figure 3.2 presents the diagram of the proposed procedure, which is the two stages of the k-fold cross-validation procedure, where $k = 5$. We have chosen $k = 5$

because it reduces the required computation as done in [19]. As shown in Figure 3.2, there is an outer 5-fold cross-validation loop which is used to validate our method of selecting the parameters a and b. In addition to the outer loop, there is an inner 5-fold cross-validation loop that is used to optimise the parameters a and b. The outer loop is repeated 5 times, producing five different training and test sets, resulting from the entire dataset. Each fold of the outer training set is again divided into 5 folds and the inner loop is repeated 5 times as well. The inner loop will return only the model with the most optimal values for a and b to the outer loop, which will use its test set to evaluate the model's quality. In the outer loop, we will get 5 different performance that can be averaged to obtain the final performance. However, as we still want to use a best value of a and b as the target proportions for a real data set. Therefore we will take those results from the outer loop, and extracting the best a and b to be the target proportions for the data set.

For more explanation, we present the method step by step as follows. First, we randomly divide the data into five folds, $k = 5$, each containing a training and testing set. Secondly, each outer training fold (starting with outer training fold 1, as in the red box in A) is again divided into five inner folds, each containing training and testing set as in B. In the inner folds, as in B, we discover possible optimal values of a and b using optimisation techniques. So, we have five possible values of parameters a and b. After that, we choose the values for the parameters a and b from the inner folds that give the best classification accuracy on inner testing folds to test on the outer test fold in A (for example test fold 1). There are many optimisation methods in the literature that can be used to discover these optimal values, where a Genetic Algorithm (GA) [47, 48] is one of the most commonly used optimisation methods in the literature. The GA is a search-based optimisation technique based on the rules of genetics and natural selection to provide solutions to problems. We use the GA as additional tuning in order to discover the best values of a and b in the inner stage. More details about the GA method are given in the appendix. Thirdly, as in C, we record the result of this outer fold including the values of a and b and the classification accuracy. Then, we repeat this process for the remaining

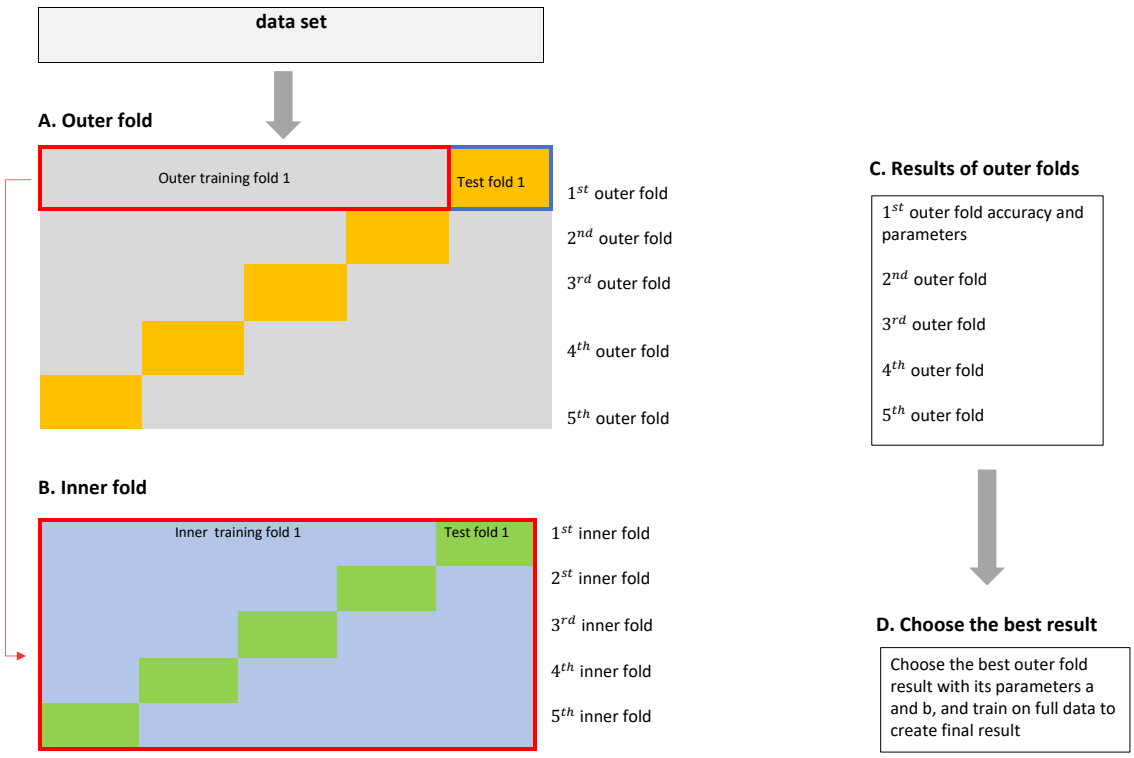


Figure 3.2: A diagram of a two-stage 5-fold cross-validation procedure to find the optimal values of the target proportions a and b.

outer folds. Finally, as in D, we choose the best values of a and b that give the highest classification accuracy; we then use these values as the optimal values for the data set. Note that we build the whole classification tree in this process using all the available attributes, not based on a single attribute. So this is a joint optimisation problem, where at each stage we build a full tree. If the number of such attributes increases, then we probably have an exponential increase in the computation time for the optimization. Therefore, it would be of interest to investigate the use of suitable fast optimization techniques, this is left as a topic for future research.

Having explained the process of choosing the values of a and b, we present in Section 3.3.2 a new algorithm for building binary classification trees using the NPI approach for selecting the optimal thresholds and the proposed method for choosing

a and b.

3.3.2 NPI$_2$-Tree algorithm

We present a new classification algorithm for building binary classification trees which we call the Nonparametric Predictive Inference for Binary Classification Trees (NPI$_2$-Tree) algorithm. Note that this algorithm can only be used to build trees with two classes. For this reason, we added the number 2 to the abbreviation. The procedure for building classification trees using the NPI$_2$-Tree algorithm is quite similar to Quinlan's C4.5 algorithm [67], given in Section 2.3. The main difference is that we use the NPI method presented in Section 3.2 as a criterion for selecting the optimal threshold for each continuous-valued attribute with our proposed method of choosing the values of a and b presented in Section 3.3.1. Note that the NPI$_2$-Tree algorithm uses the information gain ratio as a split criterion to select the attribute variable at each node. In Section 3.5, we also evaluate the performance of the NPI$_2$-Tree algorithm using a different split criterion based on imprecise probability.

Suppose that we have a data set, $\mathcal{D}$, which has continuous-valued attributes $\{X_1, \ldots, X_f\}$, and a binary class variable, $C \in \{C_1, C_2\}$. As a first step, we divide the data set, $\mathcal{D}$, into two subsets: training data set, S, and testing data set, T. Let n_1 represent the total number of observations that belong to class C_1, and n_2 represent the total number of observations that belong to class C_2. We set the number of future observations, m_1 and m_2, based on the S data distribution, that is, we choose the value of m_1 equal to the number of observations that belong to class C_1 in S and the value of m_2 equal to the number of observations that belong to class C_2 in S. As a starting point, we set the initial values of the a and b equal to the data proportions, meaning that we choose the value of a equal to the proportion of the training data S belonging to class C_1, i.e. $a = \frac{n_1}{n}$, and the value of b equal to the proportion of S belonging to class C_2, i.e. $b = \frac{n_2}{n}$. We have set the initial values for a and b to be equal to the data proportion because this could help us to make the optimisation process faster in discovering these optimal values, as this choice provided a good performance in Example (3.3.1). In Example 3.3.2, we illustrate

how we set the values of m_1, m_2 and a, b to build classification trees at the beginning.

Example 3.3.2 Assume we have a training data set with 40 observations, 25 observations from class C_1 and 15 observations from class C_2. We set the values of the number of future observations, m_1 and m_2, based on the training set distribution, so we have $m_1 = 25$ and $m_2 = 15$. In addition, as a starting point to build the trees, the values of a and b are set based on the data proportion in S, so we have $a = 0.42$ and $b = 0.58$.

For the training data set S we find the optimal threshold values for each of the continuous-valued attributes, X_i, for $i = 1, \ldots, f$, by maximising the NPI lower probability given in Equation (3.1). As shown in [11, 37], the optimal threshold value t can only be found in intervals in where the left-end point of the interval is an observation that belongs to class C_1 and the right-end point is an observation that belongs to class C_2. The first and last intervals should also be considered. This property is useful when building the classification trees because it speeds up the process of determining the optimal thresholds.

After selecting the optimal threshold t for X_i, we compute the IGR value (Equation formula (2.4)) for all attributes X_i, for $i = 1, \ldots, f$, to find the best attribute variable for the root node. Once the IGR values are computed for all attributes, the attribute variable with the highest IGR value is chosen as the best attribute for the root node. Then, based on the chosen attribute's threshold, we split the training data set S into two disjoint subsets, S_1 and S_2, where $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$. Here, S_1 is the subset of the training data set S with $X_i \leq t$, and S_2 is the subset of the data with $X_i > t$. I.e. small data values are more likely to be from C_1 and large data values from C_2, i.e. $X_i \leq t$ from class C_1 and $X_i > t$ from class C_2. After selecting the best attribute variable at the root node and splitting the training data set into two subsets, we then again find the optimal thresholds and the IGR values for both subsets S_1 and S_2. The NPI$_2$-Tree algorithm continues recursively by selecting

further splitting the data, and hence, the algorithm creates new subtrees for each branch. The tree branching is stopped when all observations in the subset belong to a single class, or if there is no attribute left, or when the number of observations per leaf node reaches the minimum split value. A minimum split number is a value that must exist in a node before splitting the data . Algorithm 1 summarises the process of building binary classification trees using the NPI_2-Tree classification algorithm.

Note that the value of the minimum split number is sometimes fixed for a particular value and is considered as a stopping rule when building a classification tree, as was done by Berry and Linoff [21], and by Bertsimas and Dunn [22]. This value is selected based on the aim of the analysis and the characteristics of the data set. For example, Berry and Linoff suggested setting the value of the minimum split number to be between 0.25% and 1.00% of the training data set in order to avoid overfitting. Overfitting occurs when a classification algorithm works very well on the training set but does not work very well on the testing set. In this thesis, in order to prevent our classification algorithms from building large trees that may overfit the data and reduce classification performance, we set the value of the minimum split number equal to five. I.e. if any node in a classification tree contains more than five observations, we continue to split a node further, otherwise, we stop the tree branching and fix a leaf node with the most frequent class in that node. This value of the minimum split number is also used for all considered classification algorithms, hence the comparison is fair. We have chosen the value of the minimum split number equal to five after conducting many study experiments and five was shown to give good performance. It would be interesting to study also further stopping rules that may be used to prevent one from overfitting. For example, setting the value of the tree's maximum depth may also prevent the trees from overfitting. The tree's maximum depth is the distance from the root node to the farthest leaf node in the tree [21].

Algorithm 1 Pseudocode NPI_2-Tree algorithm

1. **Input**:($\mathcal{S}$, C, Ω)
2. $\mathcal{S}$: Training data set
3. C: Binary class variable $C = \{C_1, C_2\}$
4. Ω: Set of continuous attributes $\Omega = \{X_1, \ldots, X_f\}$
5. **Procedure** NPI_2-Tree($\mathcal{S}$, C, Ω)
6. Create a Root node for the tree
7. **if** all observations in $\mathcal{S}$ have the same class C, **then**
8. Return the single-node tree with class C
9. **if** Ω is empty (i.e. there are no attributes available), **then**
10. Return the single-node tree with most common class C in $\mathcal{S}$
11. Otherwise
12. Select the values of a, b and m_i for $i = 1, 2$
13. Make the initial values of a and b equal to the class proportion in S,
14. i.e. make $a = \frac{n_1}{n}$ and $b = \frac{n_2}{n}$
15. Make the values of m_i equal to the number of observations in class C_i in S,
16. i.e. make $m_1 = n_1$ and $m_2 = n_2$
17. **for** each attribute, X_i in Ω, **do**
18. Find the threshold value that maximise the NPI lower probability, Eq. (3.1)
19. Compute the IGR value, given in Eq. (2.4)
20. Choose attribute X from Ω, with the highest IGR value
21. Assign the attribute X for the Root node
22. Add a branch below the Root node, corresponding to $X \leq t$ and $X > t$
23. Let S_i, for $i = 1, 2$ be the subset of S that has $X \leq t$ and $X > t$, respectively
24. **if** S_i is not empty, **then**
25. Add the subset created by NPI_2-Tree (S_i, C, $\Omega - \{X\}$)
26. **return** Root

3.3.3 Examples

This section presents two examples, one illustrates the process of building a classification tree using the NPI_2-Tree algorithm with the proposed method of choosing the values of a and b, and one is to show a comparison of classification trees using our method of choosing the values of a and b, and the predefined method of choosing these values. Note that the performance of the NPI_2-Tree algorithm with our proposed method of choosing the values of a and b is evaluated and compared with other classification algorithms in Section 3.4.

Example 3.3.3 This example illustrates how we build a classification tree using the NPI_2-Tree algorithm, presented in Section 3.3.2, with our proposed method of choosing the target proportions, presented in Section 3.3.1. To this end, we have used the Cryotherapy data sets obtained from the UCI repository of machine learning databases [41], consisting of 90 observations, 5 attribute variables, and a binary class variable. This data set has been created by medical experts [54, 55], which contains information about the results of wart treatment with cryotherapy for 90 patients. It is important to clarify that the Cryotherapy data set also has two more other attributes, which are categorical. However, as mentioned earlier, we only work with continuous-valued attributes at this stage of developing the NPI_2-Tree algorithm, so the categorical attributes in this data set were ignored. Table 3.4 shows a brief overview of this data set.

The first step to build the NPI_2-Tree classification tree is to select the optimal target proportions a and b using the method introduced in Section 3.3.1. As explained in this method, the data set is divided into two levels of the 5-fold cross-validation procedure. In the first level, which is the outer level, we train our classification algorithm with optimal values of a and b, while in the second level, which is the inner fold, we use a search function, which is the Genetic Algorithm, given in the appendix, to tune the values a and b. Using this method, the optimal values that maximise classification accuracy are $a = 0.56$ and $b = 0.50$. Now, we use these

Attribute	Attribute Description	Attribute type
X_1	Age of patient	continuous
X_2	Time elapsed before treatment	continuous
X_3	Number of warts	continuous
X_4	Surface area of the warts	continuous
X_5	Induration diameter ofinitial test	continuous
C	Class variable	binary

Table 3.4: Attribute description of Cryotherapy data set

values as the optimal target proportions for building the classification tree for the Cryotherapy data set. In order to show how we build the classification tree using the NPI$_2$-Tree algorithm with the values of a and b, we divide the Cryotherapy data set into two subsets: a training data set, about 80% of the data, where $n_1 = 38$ and $n_2 = 34$, and a testing data set, about 20% of the data. So we set $m_1 = n_1 = 38$ and $m_2 = n_2 = 34$. Note that this binary splitting of the data set is only used here to illustrate how the NPI$_2$-Tree classification tree is built. However, in the experimental analysis given in Section 3.4, the performance of the NPI$_2$-Tree algorithm is evaluated and compared with other classification methods using the 10-fold cross-validation procedure, introduced in Section 2.5. Thus, splitting the data set into a training set with 80% and a testing set with 20% is only applied here to illustrate our examples.

As the next step, we find the threshold value that maximises the NPI lower probability (Equation (3.1)) for each attribute variable in the training set. Then we calculate the IGR value (Equation (2.4)) for all attribute variables. These thresholds and the IGR values for each attribute variable are given in Table 3.5. We now choose the attribute variable with the highest IGR value, which is X_2 (Time elapsed before treatment), to be assigned to the root node for the tree. Hence, we divide the data set according to the threshold value of X_2. Note that we do not use X_2 again for splitting the data in the next stages of building the NPI$_2$-Tree classification tree.

After that, we divide the training data set S into two subsets, S_1 and S_2, where

Attribute	t	IGR value
X_1	27.66	0.075
X_2	10.64	0.280
X_3	6.00	0.018
X_4	48.00	0.009
X_5	6.22	0.008

Table 3.5: The optimal thresholds and the IGR values for all attributes

$S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$. Here S_1 is the subset of the training data set S with $X_2 \leq 10.64$, and S_2 is the subset of the training data set with $X_2 > 10.64$. In the subset S_1 of the training data sets, we get a pure subset in which all patients have class C_1. Therefore, this subset is not splitting again, and we fix a leaf node with class C_1. For the second subset S_2, we do not get a pure set, i.e. some patients have class C_1, and some patients have class C_2. We now need to select the optimal thresholds and calculate the IGR value for S_2, these optimal thresholds and IGR values are presented in Table 3.6. We then choose the attribute variable with the highest IGR value as a second splitting variable in the data set, hence, we choose X_3 (Number of warts). The data set must be again splitted into two subsets according to the threshold value of X_3. For the subset below the branch of $X_3 > 5.85$, we get a pure subset, thus, we fix a leaf node with class C_2. Similarly, we select X_1 as a third split below the branch of $X_3 \leq 5.85$ because we do not get pure subset. We terminate the tree when all patients in each subtree have the same class. So, we stop here and show the result of the full classification tree as in Figure 3.3, with classification accuracy 89%. In Section 3.4, we present further analysis of this data set and when we evaluate the performance of the NPI_2-Tree algorithm against other classification algorithms.

Example 3.3.4 In this example, we compare two way of choosing the target proportions, a and b, in order to select the optimal thresholds and build classification trees. We first pre-define these proportions, and then we determine them based on our optimal way, presented in Section 3.3.1. To this end, we have used a data set

Attribute	t	IGR value
X_1	27.66	0.246
X_3	5.85	0.267
X_4	13.84	0.170
X_5	7.87	0.080

Table 3.6: The optimal thresholds and the IGR values for data set with $X_2 > 10.64$

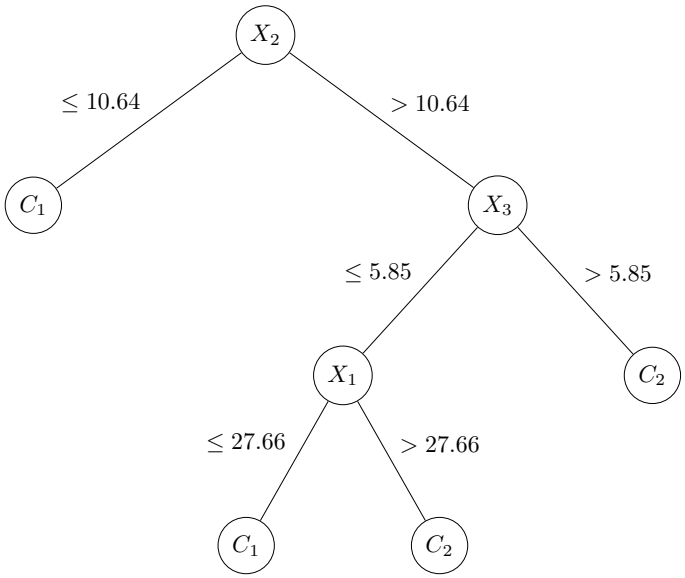
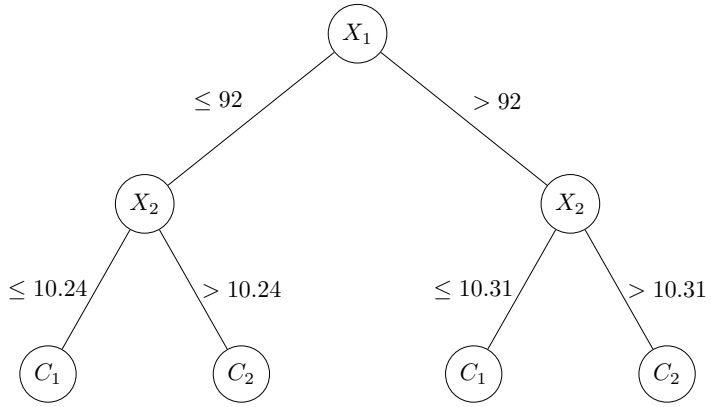
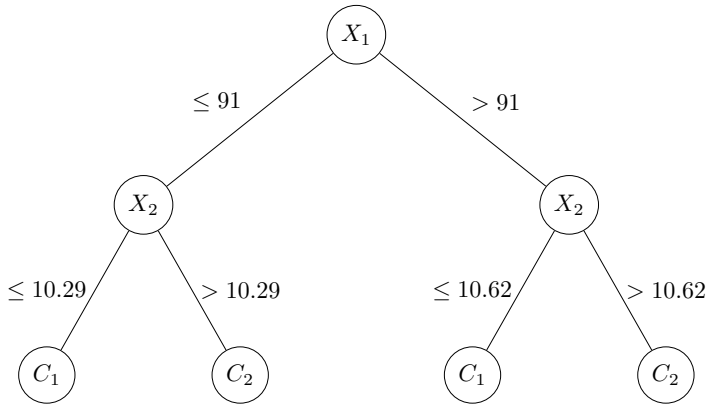


Figure 3.3: Classification tree created by NPI_2-Tree algorithm, with $a = 0.56$ and $b = 0.50$.



(a) Classification tree with accuracy 72% and $a = b = 0.30$



(b) Classification tree with accuracy 76% and $a = b = 0.85$

Figure 3.4: Different classification trees with different scenarios of the predefined target proportions.

obtained from the UCI repository of machine learning databases [41], consisting of 116 observations, two attribute variables, X_1 and X_2, and a binary class variable $C \in \{C_1, C_2\}$. The data set is divided into two subsets: a training data set, about 80% of the data, and hence $n_1 = 42$, $n_2 = 49$, and a testing data set, about 20% of the data. So we have $m_1 = n_1 = 42$, $m_2 = n_2 = 49$.

First, we have built classification trees considering two different scenarios for the target proportions, which are $a = b = 0.30$ and $a = b = 0.85$. Figure 3.4 presents the classification trees created by each of these scenarios along with their corresponding classification accuracy results. As we see from this figure, the optimal thresholds change depending on the a and b values, but they create the same structure of the

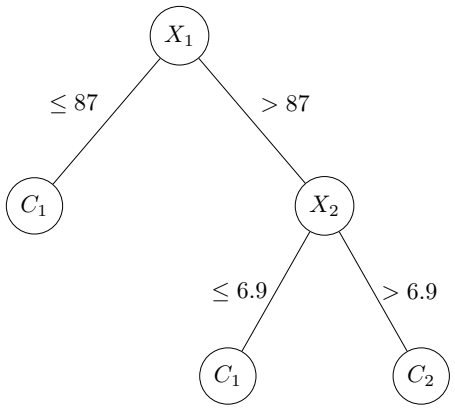


Figure 3.5: Classification tree with our optimal way of choosing a and b, where the classification accuracy is 84% and the obtained values are $a = 0.50$ and $b = 0.65$.

trees. In the first scenario, we have chosen small values, that is $a = b = 0.30$, as we can see from Figure 3.4(a), the tree size, which is the number of leaf nodes in the tree, is equal to 4. The total classification accuracy of this scenario is 72%. In the second scenario, we have increased the target proportions to $a = b = 0.85$, as shown in Figure 3.4(b). The classification accuracy in this scenario increased slightly to 76%, while the structure of the tree remains the same as the tree created in the first scenario. Secondly, we have built a classification tree using our suggested method for choosing the values of a and b as presented in Section 3.3.1. Using this method, the optimal values are $a = 0.50$ and $b = 0.65$. As showing in Figure 3.5, the structure of the classification tree is different from the trees in Figure 3.4, where the tree in Figure 3.5 has the smallest tree size (number of leave nodes) compared to the other trees. In addition, the classification accuracy increased to 84% compared to the trees in Figure 3.4. The result differs from the previous results because we set meaningful target proportions for the predictive inferences. In Section 3.4, we further evaluate the performance of the NPI_2-classification trees using our method of choosing the values of a and b, and we compare its performance with other classification algorithms from the literature.

3.4 Experiment

In this section, we aim to examine the performance of the NPI_2-Tree algorithm on 6 data sets taken from the UCI repository of machine learning databases [41]. We used only six data sets because the NPI_2-Tree algorithm works with only continuous-valued attributes and a binary class variable, and such data sets are uncommon in public databases. However, it would be interesting to fully automate the NPI_2-Tree algorithm to enable us to analyse more data sets including categorical attributes. A brief description of the main properties of each data set is given in Table 3.7. Column 'N' gives the number of observations in the data set, column 'Attr' gives the number of attribute variables, column 'Pro of class 1' gives the data proportion in class 1 and column 'Pro of class 2' gives the data proportion in class 2. Further details about these data sets can be found in [41]. It is important to note that, as we only work with continuous-valued attributes, the categorical attributes in the data sets were ignored. So, the number of attributes in the 'Attr' column is only the number of continuous-valued attributes. Therefore, we may not be surprised if the results found in the literature are different from our results. For example, the Liver Patients data set has ten attribute variables; nine are continuous and one is categorical. The classification accuracy obtained in our experiment using the C4.5 algorithm is 77.25%, but in [70], the classification accuracy result using the C4.5 algorithm is 84.86%, where the categorical attribute, which is the gender of the patient, was used. As the NPI_2-Tree algorithm presented in this thesis only functions for continuous attribute variables, it would be interesting to further develop the NPI_2-Tree algorithm to work with categorical attributes, we leave that for future research.

As a first step, we use the NPI_2-Tree algorithm to build a classification tree for each data set. This was done using the tree-building process presented in Section 3.3.2, with the proposed method of choosing the values of a and b as presented in Section 3.3.1. We compare the performance of the NPI_2-Tree algorithm for each data set with the most commonly used classical classification trees, which are the C4.5 and the CART algorithms. More details about these algorithms have been presented in Sections 2.3 and 2.4. We also compare the performance of the NPI_2-

Data set	N	Attr	Pro of class 1	Pro of class 2
Breast Cancer	116	9	0.45	0.55
Blood Transfusion	748	3	0.76	0.24
Liver Patients	583	9	0.28	0.72
Haberman's Survival	306	3	0.73	0.27
Cryotherapy	90	5	0.53	0.47
QSAR Biodeg	1055	14	0.66	0.34

Table 3.7: A brief description of the data sets

Tree algorithm with two imprecise algorithms, which are the NPI-M and the IDM algorithms. Further details about these algorithms have been presented in Sections 2.3, 2.6 and 2.7.4. As the IDM algorithm depends on the value of the parameter $\tilde{s}$, we use one recommended value of $\tilde{s}$, which is $\tilde{s} = 1$. We refer to the IDM with $\tilde{s} = 1$, by the IDM1 algorithm.

The R software [69] has been used to carry out this experiment. We used the `RWeka` package [51, 79] to implement the C4.5 algorithm, the `rpart` package [75] to implement the CART algorithm and the `imptree` package [45] to implement both the NPI-M and IDM1 algorithms. Some pre-processing steps in our data sets have been carried out. The missing values for continuous attributes were replaced with mean values using the missing value filter in R. In addition, some of the data sets contain tied observations; we dealt with these by adding a small amount to the tied observations. We also tested our method without breaking the tied observations and observed that the results were close. Furthermore, for the NPI-M and the IDM1 algorithms, as they can only handle categorical attributes, therefore, we discretised the continuous variables presented in Table 3.7 using the `mdlp` package in R and the 'discretization' function. This function converts a continuous variable into a categorical variable using the Fayyad and Irani method [43, 44], which find a threshold value using only the avergae class entropy to evaluate the partitions created by each candidate threshold. These pre-processing steps are essential, and they were carried out at the beginning of the analysis for all algorithms to ensure a fair comparison for all algorithms. After that, we applied all classification algorithms to all data sets

Data set	a	b	NPI_2-Tree	C4.5	CART	NPI-M	IDM1
Breast Cancer	0.57	0.73	87.10	86.89	86.90	87.65	**87.86**
Blood Transfusion	0.79	0.64	**89.48**	75.43	74.76	79.56	79.56
Liver Patients	0.32	0.65	**80.70**	77.25	76.43	80.28	80.28
Haberman's Survival	0.84	0.42	75.19	75.62	73.18	**76.39**	**76.39**
Cryotherapy	0.56	0.50	79.38	80.11	**83.48**	81.45	80.18
QSAR Biodeg	0.65	0.82	**91.22**	73.13	77.86	82.16	82.16
Average	-	-	**83.84**	78.07	78.72	81.24	81.07

Table 3.8: Average result of classification accuracy of different classification algorithms and the optimal values of a and b for the NPI_2-Tree algorithm.

and the results were compared in several ways.

Five classification algorithms, which are the NPI_2-Tree, the C4.5, the CART, the NPI-M and the IDM1 algorithms, have been used in this experiment. The following measures were used to evaluate the performance of the classification algorithms: classification accuracy, in-sample accuracy and tree size. The classification accuracy is the ratio of the number of correctly classified observations to the total number of observations in the test data set, while the in-sample accuracy is the classification accuracy for the training set. The tree size is the number of leaf nodes in the tree [22]. All results given in this experiment were obtained using the average of a 10-fold cross-validation scheme, as presented in Section 2.5. With respect to the NPI_2-Tree algorithm, we first found the optimal values of a and b using the method presented in Section 3.3.1, and then we used the 10-fold cross-validation scheme to report the final result. The results of this experiment are presented in Tables 3.8 - 3.10. Table 3.8 presents the results of classification accuracies of the proposed NPI_2-Tree algorithm and other algorithms for each data set, including the optimal values of a and b used by the NPI_2-Tree algorithm. Table 3.9 shows the results of in-sample accuracy for all classification algorithms. The tree sizes of classification algorithms are presented in Table 3.10. In all tables, the best results are presented in bold font.

As shown in Table 3.8, the classification accuracies indicate that the NPI_2-Tree

Data set	a	b	NPI_2-Tree	C4.5	CART	NPI-M	IDM1
Breast Cancer	0.57	0.73	88.12	**88.22**	**88.22**	**88.22**	**88.22**
Blood Transfusion	0.79	0.64	82.78	**84.58**	**84.58**	84.40	84.40
Liver Patients	0.32	0.65	89.80	**83.93**	83.20	81.96	81.62
Haberman's Survival	0.84	0.42	75.40	**76.99**	**76.99**	76.23	76.23
Cryotherapy	0.56	0.50	**82.87**	82.64	80.34	82.16	82.28
QSAR Biodeg	0.65	0.82	87.39	87.50	**88.86**	85.96	85.96
Average	-	-	82.72	**83.97**	83.71	83.16	83.11

Table 3.9: Average result of the in-sample accuracy of the different classification algorithms and the optimal values of a and b for the NPI_2-Tree algorithm.

algorithm outperforms other classification algorithms for most data sets, and it has the highest average classification accuracy, followed by the NPI-M, the IDM1, the CART and the C4.5. For the Breast Cancer data set, all classification algorithms obtained similar results, with the NPI-M algorithm performing slightly better than the other algorithms. For the Blood Transfusion and QSAR Biodeg data sets, there is a noticeable difference in classification accuracies between the classification algorithms, with the NPI_2-Tree algorithm clearly outperforming the other classification algorithms. We have investigated the characteristics of these data sets in order to gain insight into reasons that may cause this clear difference in the performance of the NPI_2-Tree algorithm, and we found that these data sets are large compared to other data sets and they have less overlap between their data classes, which could be a reason why the NPI_2-Tree algorithm is superior to the other algorithms on these data sets. The worst performing algorithms are the CART and C4.5 algorithms. The NPI-M and IDM1 algorithms obtained very similar classification accuracies. For the Liver Patients data set, we can see that the NPI_2-Tree, the NPI-M and the IDM1 algorithms perform better than the classical algorithms. For the Haberman's Survival data set, the NPI-M and the IDM1 perform slightly better than the other algorithms, obtaining the same classification accuracy of 76.39%. Finally, for the Cryotherapy data set, the NPI_2-Tree algorithm obtained the worst result, which is 79.38%. For this data set, the NPI_2-Tree algorithm built smaller trees than the ones built by the other algorithms, which might be the reason why it performs a little

Algorithm	NPI_2-Tree	C4.5	CART	NPI-M	IDM1
Average	**4.20**	4.72	4.46	4.33	4.39

Table 3.10: Average result of the tree size of the different classification algorithms.

less than the other algorithms.

Table 3.9 presents the results of the in-sample accuracy of all classification algorithms. The in-sample accuracy, which is the classification accuracy on the training set, is not widely used to evaluate classification algorithms, but it gives insight into how the classification algorithm works on the training set. It is well known that if the classification algorithm works very well on the training set but does not work very well on the testing set, it likely indicates overfitting. Therefore, it is useful to show the classification algorithms' performance on both training and testing sets, and hence, to check overfitting as well, as was done in [22, 64]. As we can see from Table 3.9, the C4.5 algorithm performs slightly better than the other classification algorithms, followed by the CART, the NPI-M, the IDM1 and the NPI_2-Tree. Note that the C4.5, the CART, the NPI-M and the IDM1 algorithms have better in-sample accuracy than classification accuracy, but, the performance of the NPI_2-Tree algorithm on the training sets is slightly worse than its performance on the testing sets. It is possible that these results are due to the fact that the NPI_2-Tree algorithm selects the optimal thresholds by focusing on prediction, unlike the other algorithms which focus on maximising the correct classification on the training sets. We see from the results that the C4.5 and the CART algorithms clearly perform better on the training sets than on the testing sets, which indicates that these algorithms may be suffering from overfitting. According to the average results of classification accuracy and in-sample accuracy, we can state that the NPI_2-Tree algorithm works well on both data sets, training and testing sets, which might indicate that the NPI_2-Tree algorithm does not suffer from overfitting.

Finally, Table 3.10 shows the average results of the tree sizes for all the classification algorithms. Note that it is referred here to the tree size as the total number

of leaf nodes in the tree, as was done by Bertsimas and Dunn [22], and by Murthy and Salzberg [64]. However, some researchers may refer to the tree size as the total number of all nodes in the tree. Of course, one can use any method to refer to the size of the tree. Table 3.10 shows that the NPI$_2$-Tree algorithm creates slightly smaller trees than the other algorithms, followed by the NPI-M algorithm. The C4.5 algorithm creates the largest trees with average tree size of 4.72, which could be the reason for its good performance on the training sets. The CART algorithm mostly creates similar trees to the C4.5 algorithm, but in Liver Patients and Cryotherapy data sets, the CART algorithm has the smallest tree size compared to the C4.5 algorithm, and hence, it gives a smaller average tree size than the C4.5 algorithm. This study shows that the NPI$_2$-Tree algorithm tends to create smaller trees than the C4.5, the CART, the NPI-M and the IDM1 algorithms.

Overall, the results of the experimental analysis shows that the NPI$_2$-Tree algorithm performs well compared to the other classification algorithms. According to the classification accuracy results, the NPI$_2$-Tree algorithm is performing better than other classification algorithms, followed by the NPI-M, the IDM1, the CART and the C4.5. Regarding the in-sample accuracy results, the other algorithms are performing slightly better than the NPI$_2$-Tree algorithm. This is because of the method of selecting the threshold values, in which the NPI$_2$-Tree algorithm selects the optimal threshold by focusing on prediction rather than maximising the correct classification of the training data sets. Finally, the NPI$_2$-Tree algorithm obtained the smallest average tree size, while the C4.5 has the highest average tree size.

3.5 NPI$_2$-Tree with imprecise split criterion

In this section we aim to evaluate the performance of the NPI$_2$-Tree algorithm using a different split criterion based on imprecise probability, and compare the results with the classical split criterion. We use the imprecise split criterion, Imprecise Information Gain, introduced in Section 2.3, to build the NPI$_2$-Tree classification trees

3.5.1 Imprecise Information Gain (IIG)

The Imprecise Information Gain (IIG) is a split criterion, which was introduced by Abellán and Moral [6], to build classification trees from an imprecise probability perspective. It is an extension of the Information Gain (IG) split criterion introduced by Quinlan [66] as the basis of the ID3 algorithm, replacing precise probabilities and entropy with imprecise probabilities and the maximum entropy over credal sets. As explained in Section 2.3, one can build different classification trees using the IIG split criterion. For example, one can build a classification tree using the maximum entropy distributions from the credal sets associated with the Imprecise Dirichlet Model (IDM) or with the Nonparametric Predictive Inference for multinomial data (NPI-M), which are presented in Sections 2.6 and 2.7.4, respectively.

However, as we are working with a binary class variable, it is useful to obtain the credal sets from the NPI for Bernoulli data (NPI-Bern) [27], introduced in Section 2.7.3. This is useful because the NPI-Bern method leads to slightly less imprecision than the IDM and the NPI-M methods. In the following, we explain how we obtain the credal sets from the NPI-Bern approach, and how we maximise the entropy on the credal sets, for a class variable C. Assume that we have a data set with n observations and a binary class variable, $C \in \{C_1, C_2\}$. In the classical split criterion IGR, the probability distribution p_i, for class i (for $i = 1, 2$), is the proportion of the data belonging to class i, that is,

$$p_i = \frac{n_i}{n}, \text{for } i = 1, 2 \tag{3.9}$$

where n is the total number of observations, and n_i is the number of observations that belong to class C_i. However, when we use the imprecise split criterion, IIG, the probability p_i is given by an interval of probabilities, i.e. $\underline{P}$ and $\overline{P}$. So, using the NPI-Bern method, but considering only a single future observation, i.e. $m = 1$, the probability p_i is obtained using an interval of probabilities

$$p_i \in \left[\frac{n_i}{n+1}, \frac{n_i+1}{n+1}\right], i = 1, 2 \tag{3.10}$$

This representation provides imprecise probabilities that lead the following closed

convex set of probability distributions for the class variable C,

$$L(C) = \left\{ p|\ p_i \in \left[\frac{n_i}{n+1}, \frac{n_i+1}{n+1} \right], i = 1, 2, \sum_{i=1}^{2} p_i = 1 \right\} \tag{3.11}$$

On this type of credal set, the classification trees are built using the maximum entropy function. This function is denoted by H^* and defined as

$$H^*(L(C)) = \max \{H(p), p \in L(C)\} \tag{3.12}$$

where H is the classical entropy function (Formula 2.1). The following example illustrates how we use the maximum entropy on credal sets.

Example 3.5.1 Assume that we have a training data set S consisting of an attribute variable X and a binary class variable C. Suppose the data set has 36 observations belonging to class C_1 and 40 observations belong to class C_2, so $n_1 = 36, n_2 = 40$ and $n = 76$. The credal set of probability intervals on the class variable C is

$$L(C) = \left\{ p|\ p_1 \in \left[\frac{36}{77}, \frac{37}{77} \right], p_2 \in \left[\frac{40}{77}, \frac{41}{77} \right], p_1 + p_2 = 1 \right\}$$

We choose the probability distribution p_i, for $i = 1, 2$, from $L(C)$ that maximises the entropy, Equation (3.12).

Similarly, we can compute the imprecise split criterion IIG for the attribute variable X and the class variable C, IIG(C, X), as presented in Section 2.3 (Formula (2.8)). It should be noted that the value of the imprecise split criterion can be negative, unlike the classical split criterion which cannot be negative. This property allows the IIG split criterion to detect and discard variables that worsen the information on the class variable [3]. Therefore, we consider this property to be useful as an additional stopping criterion to prevent the tree from branching, as was done by Abellán and Castellano [3], and by Mantas et al. [60].

3.5.2 Performance of the NPI$_2$-Tree with the IIG

This section shows the performance of the NPI$_2$-Tree algorithm using the imprecise split criterion, the IIG split criterion with NPI-Bern, and compares its results with

Data set	a	b	IGR	IIG
Breast Cancer	0.57	0.73	**87.10**	**87.10**
Blood Transfusion	0.79	0.64	**89.48**	88.24
Liver Patients	0.32	0.65	**80.70**	80.12
Haberman's Survival	0.84	0.42	**75.19**	**75.19**
Cryotherapy	0.56	0.50	79.38	**79.94**
QSAR Biodeg	0.65	0.82	**91.22**	90.83
Average	-	-	**83.84**	83.57

Table 3.11: Average result of the classification accuracy of the NPI$_2$-Tree algorithm using different split criteria.

Data set	a	b	IGR	IIG
Breast Cancer	0.57	0.73	**88.12**	87.17
Blood Transfusion	0.79	0.64	**82.78**	81.96
Liver Patients	0.32	0.65	**79.80**	78.65
Haberman's Survival	0.84	0.42	75.40	**78.12**
Cryotherapy	0.56	0.50	**82.87**	**82.87**
QSAR Biodeg	0.65	0.82	**87.39**	86.22
Average	-	-	**82.72**	82.48

Table 3.12: Average result of the in-sample accuracy of the NPI$_2$-Tree algorithm using different split criteria.

the NPI$_2$-Tree algorithm using the classical split criterion, the IGR split criterion, to determine which of these split criteria is more appropriate for the NPI$_2$-Tree algorithm. The tree building process by the NPI$_2$-Tree algorithm using the imprecise split criterion is very similar to the tree building process by the NPI$_2$-Tree algorithm using the classical split criterion presented in Agorithm 1 (see Section 3.3.2), but the IGR replaced by the IIG.

In order to evaluate the NPI$_2$-Tree algorithm using the IIG imprecise split criterion, we have used all tools that have been used in Section 3.4. The same six data sets presented in Table 3.7, and the same performance measures, namely classification accuracy, in-sample accuracy and tree size, are used. We have used the 10-fold

Split criteria	IGR	IIG
Average	**4.20**	4.35

Table 3.13: Average result of the tree size of the NPI$_2$-Tree algorithm using different split criteria.

cross-validation procedure, introduced in Section 2.5. Furthermore, the same values of a and b as presented in Table 3.8 are used to build classification trees. Table 3.11 presents the average results of classification accuracy of the NPI$_2$-Tree using the two split criteria, the IGR and the IIG. Table 3.11 shows that the NPI$_2$-Tree algorithm obtained very similar results with both split criteria. For the Breast Cancer and Haberman's Survival data sets, the NPI$_2$-Tree algorithm obtained the same accuracy of 87.10% and 75.19%, respectively, for both criteria. Table 3.12 shows the average results of the in-sample accuracy of the NPI$_2$-Tree algorithm for both split criteria. Both split criteria have again obtained very similar results, performing slightly better with the NPI$_2$-Tree using the IGR split criterion. For the Haberman's Survival data set, the NPI$_2$-Tree using the IIG criterion performs slightly better than the NPI$_2$-Tree using the IGR criterion with in-sample accuracy of 78.12. For this data set, it is noticed that the NPI$_2$-Tree algorithm using the IIG imprecise split criterion creates larger trees, i.e. number of leaves, than the IGR split criterion, which could be the reason for this result. Finally, tree sizes for the two split criteria are presented in Table 3.13. The NPI$_2$-Tree algorithm with the IGR has the smallest average tree size, while the NPI$_2$-Tree algorithm with the IIG has the largest average tree size. We have noticed during the experimental analysis that when the value of the IIG for some attributes in data sets, e.g. for the Breast cancer data set, is negative, the NPI$_2$-Tree algorithm with the IIG creates smaller trees compared to the NPI$_2$-Tree algorithm with the IGR. This is because we stop the branching of trees when the value of IIG is negative. Overall, according to the analysis results presented in Tables 3.11-3.13, the NPI$_2$-Tree algorithm has achieved good results for both split criteria. The performance of the NPI$_2$-Tree algorithm using the IGR split criterion has performed slightly better than the NPI$_2$-Tree algorithm using the IIG split criterion.

3.6 Concluding remarks

In this chapter, we have presented a new classification algorithm for building classification trees with data consisting of continuous-valued attributes and a binary class variable based on NPI, which we called the NPI_2-Tree algorithm. The NPI_2-Tree algorithm uses the NPI approach for selecting the optimal thresholds, where the threshold values are selected using predictive inference considering a given number of future observations and target proportions a and b. We have introduced a new procedure for selecting the values of target proportions by choosing that to maximise classification performance, using a two-level k-fold cross-validation procedure to define these values and to validate their performance in classification trees.

We have carried out an experimental analysis on several data sets in order to evaluate the performance of the NPI_2-Tree classification algorithm, using different evaluation metrics. We have also compared the results with four different classification algorithms, which are the C4.5 and the CART, NPI-M and IDM1 algorithms. The results of our experimental analysis have indicated that the NPI_2-Tree algorithm performs well and performs slightly better than the other classification algorithms in terms of both classification accuracy and tree sizes. The results have also suggested that all classification algorithms have obtained similar results in terms of in-sample accuracy. In addition, we have evaluated the performance of the NPI_2-Tree algorithm using the imprecise split criterion, and the results were compared with the classical split criterion. The results have shown that the performance of the NPI_2-Tree algorithm is quite similar for both split criteria, but slightly better with the classical split criterion.

In the experimental analysis, we have restricted attention to using only continuous attribute variables because the aim of this work is about using the thresholds for continuous attributes and our classification algorithms apply only to continuous attributes. However, this does not mean that all the attribute variables have to be continuous attributes. It is possible that, once the optimal desirable proportions have been computed, the thresholds are computed and the selection of a variable,

categorical or continuous, can be done with the information gain ratio; we leave this opportunity topic for future research. In our work in Chapter 3, we extend the use of the NPI_2-Tree algorithm from building classification trees with two classes to building classification trees with three classes, however, it is easy to generalise this method to more than three classes using the same approach. This can be achieved by generalising first the NPI method for selecting the optimal threshold to include more than three classes, as this approach is immediately generalisable in the same way to any number of classes as long as they are ordered. We then can use the same approach for choosing the target proportions and building the classification trees. We leave this as a topic for future research. Another idea for future research is to explore the use of the NPI_2-Tree classification method considering the misclassification cost. In many practical applications, classification aims to minimize misclassification costs instead of maximising the total classification accuracy. In this thesis, we have chosen the values of target proportions that maximise the total classification accuracy, however, it would be useful to develop the process of choosing these target proportions to consider the misclassification cost. It may also be useful to determine the kind of data sets for which the NPI_2-Tree algorithm works well. For example, one could analyse the characteristics of the data sets on which NPI_2-Tree performs well. Finally, it will be of interest to investigate the use of the NPI_2-Tree classification algorithm in random forests. The random forest is an ensemble learning method that make prediction by aggregating majority vote of many classification trees [24].

Chapter 4

NPI-based classification trees with three classes

4.1 Introduction

In this chapter, we extend the method for building binary classification trees, presented in Chapter 3, to build classification trees with three classes. Throughout this chapter, we assume that there is a natural ordering of the three classes, where observations from class C_1 tend to be smaller than observations from class C_2, which in turn tend to be smaller than observations from class C_3. This type of data exists in many real-world applications; for example, in medical applications, there are three stages (or classes) in Alzheimer's disease: early stage, middle stage and late stage [15]. In the middle stage of Alzheimer's disease progress, symptoms become more noticeable, and it is crucial to detect as it is a transition stage to the late stage in which no medical treatments are efficient [15]. Therefore, in order to determine the classes and split the data, there is a need to select two optimal thresholds, $t_1 < t_2$, for continuous-valued data. The classical methods usually similarly select these two thresholds as in the two classes setting, focusing on maximising the probability of correct classification in the training data set rather than prediction.

In this chapter, we present a new classification algorithm, which we call the NPI_3-Tree algorithm, for building classification trees for data with continuous-valued at-

tributes and three ordered classes. The NPI_3-Tree algorithm uses the NPI approach for selecting the optimal thresholds for the three ordered classes scenario, where the inferences are in terms of a given number of future observations and the values of target proportions. We extend the method of choosing the target proportions for cases of two-class setting, presented in Chapter 3, to cases of three-class setting. We conduct an experimental analysis using different measures on several data sets to measure the performance of the NPI_3-Tree algorithm and compare its results with other classification algorithms. Furthermore, the NPI_3-Tree algorithm is also evaluated using an imprecise split criterion, and the results are compared with the classical split criterion.

This chapter is organised as follows: In Section 4.2, we present the NPI method for selecting the optimal thresholds for data with three ordered classes, and we provide an example to illustrate this method. Section 4.3 presents the method for building the classification trees using the NPI approach for selecting the optimal thresholds for three ordered classes, with illustrative examples. In Section 4.4, we evaluate the performance of the NPI_3-Tree algorithm using the classical split criterion and compare its performance with the C4.5, CART, NPI-M and IDM1 algorithms. In Section 4.5, we also assess the performance of the NPI_3-Tree algorithm using a split criterion based on imprecise probability. Finally, some concluding remarks are given in Section 4.6.

4.2 NPI-based thresholds for three classes

In Section 3.2, we have presented the results of the NPI method for selecting the optimal threshold for two classes. In this section, we present the results of the NPI method for selecting the optimal thresholds for three ordered classes, introduced in [11, 37], with an illustrative example. As explained in [11, 37], one could naively use the NPI method, presented in Section 3.2, twice, to find the optimal thresholds t_1 and t_2 for the three classes, i.e. one can find t_1 using C_1 and C_2 and sequence find t_2

using C_2 and C_3. However, because of the assumed ordering of the three classes, selecting the two thresholds in this method may not satisfy the condition that $t_1 < t_2$. In this chapter, we will not use the NPI method for two classes, presented in Section 3.2, twice, but we will use the NPI method for selecting the optimal thresholds for three ordered classes, which finds two combined optimal thresholds t_1 and t_2. First, we summarise the results of [11, 37] using the same notation as presented in Section 3.2, but with additional notation for class C_3.

Suppose there are three classes C_1, C_2 and C_3, which have a natural ordering in the sense that observations from class C_1 tend to be smaller than observations from class C_2, which in turn tend to be smaller than observations from C_3. Let n_3 denote the number of observations in class C_3, the ordered data from this class are denoted by $x_1^3 < x_2^3 < \cdots < x_{n_3}^3$. For ease of notation, we define $x_0^3 = -\infty$ and $x_{n_3+1}^3 = \infty$. Again, the n_3 observations divide the real-line into $n_3 + 1$ intervals $I_l^3 = (x_{l-1}^3, x_l^3)$, for $l = 1, 2, \ldots, n_3 + 1$. Let m_3 denote the number of future observations in class C_3, with random variable $X_{n_3+d}^3$, for $d = 1, \ldots, m_3$. Let the m_3 ordered future observations from class C_3 be denoted by $X_{(1)}^3 < X_{(2)}^3 < \cdots < X_{(m_3)}^3$. To classify observations into one of the classes, C_1, C_2 or C_3, we want to find the two optimal thresholds t_1 and t_2, where $t_1 < t_2$, such that observations less than or equal to t_1 are classified as belonging to C_1, observations greater than t_1 and less than or equal to t_2 are classified as belonging to C_2 and observations greater than t_2 are classified as belonging to C_3. For particular values of t_1 and t_2, we denote the number of correctly classified future observations from class C_1, C_2 and C_3 by $L_{t_1}^1$, $L_{(t_1,t_2)}^2$ and $L_{t_2}^3$, respectively. Let denote the target proportions chosen to reflect the desired importance of the three classes by a, b and c, respectively. Selecting these values will depend on a person's beliefs of which class is more important to be correctly classified than others. There is no constraint on these values except to be in $(0, 1]$. One can choose a, b and c to be equal if one gives the same importance of correct classification to all three classes. In Section 4.3.1, we present a strategy to optimise these values through some automated algorithm. The general event of interest which we consider for the three classes C_1, C_2 and C_3 is that the number of

correctly classified future observations from class C_1 is at least am_1, the number of correctly classified future observations from class C_2 is at least bm_2, and the number of correctly classified future observations from class C_3 is at least cm_3, that is $L^1_{t_1} \geq am_1$, $L^2_{(t_1,t_2)} \geq bm_2$ and $L^3_{t_2} \geq cm_3$.

Using the assumption of independence between the three classes, the NPI lower probability for the event of interest is

$$\begin{aligned} &\underline{P}(L^1_{t_1} \geq am_1, L^2_{(t_1,t_2)} \geq bm_2, L^3_{t_2} \geq cm_3) = \\ &\underline{P}(L^1_{t_1} \geq am_1) \times \underline{P}(L^2_{(t_1,t_2)} \geq bm_2) \times \underline{P}(L^3_{t_2} \geq cm_3) \end{aligned} \tag{4.1}$$

and the corresponding NPI upper probability is

$$\begin{aligned} &\overline{P}(L^1_{t_1} \geq am_1, L^2_{(t_1,t_2)} \geq bm_2, L^3_{t_2} \geq cm_3) = \\ &\overline{P}(L^1_{t_1} \geq am_1) \times \overline{P}(L^2_{(t_1,t_2)} \geq bm_2) \times \overline{P}(L^3_{t_2} \geq cm_3) \end{aligned} \tag{4.2}$$

For $I^1_i = (x^1_{i-1}, x^1_i), i = 1, \ldots, n_1 + 1$, and $t_1 \in I^1_{i_{t_1}} = (x^1_{i_{t_1}-1}, x^1_{i_{t_1}})$, where $i_{t_1} \in \{2, 3, \ldots, n_1\}$ is defined as that interval $I^1_{i_{t_1}}$ which contains t_1, the NPI lower and upper probabilities for the event $L^1_{t_1} \geq am_1$ are [11, 37]

$$\underline{P}(L^1_{t_1} \geq am_1) = \underline{P}(X^1_{\lceil am_1 \rceil} \leq t_1) = \sum_{i=1}^{i_{t_1}-1} P(X^1_{\lceil am_1 \rceil} \in I^1_i) \tag{4.3}$$

$$\overline{P}(L^1_{t_1} \geq am_1) = \overline{P}(X^1_{\lceil am_1 \rceil} \leq t_1) = \sum_{i=1}^{i_{t_1}} P(X^1_{\lceil am_1 \rceil} \in I^1_i) \tag{4.4}$$

For $i_{t_1} = 1$, these NPI lower and upper probabilities are $\underline{P}(L^1_{t_1} \geq am_1) = 0$ and $\overline{P}(L^1_{t_1} \geq am_1) = P(X^1_{\lceil am_1 \rceil} \in I^1_1)$, and for $i_{t_1} = n_1 + 1$, they are $\underline{P}(L^1_{t_1} \geq am_1) = 1 - P(X^1_{\lceil am_1 \rceil} \in I^1_{n_1+1})$ and $\overline{P}(L^1_{t_1} \geq am_1) = 1$.

For $I^2_j = (x^2_{j-1}, x^2_j)$ with $j = 1, \ldots, n_2 + 1$ and $t_1 \in I^2_{j_{t_1}} = (x^2_{j_{t_1}-1}, x^2_{j_{t_1}})$, and $t_2 \in I^2_{j_{t_2}} = (x^2_{j_{t_2}-1}, x^2_{j_{t_2}})$, with $j_{t_1} \in \{1, 2, \ldots, n_1 + 1\}$ and $j_{t_2} \in \{1, 2, \ldots, n_2 + 1\}$, with $t_2 \geq t_1$ so $j_{t_2} \geq j_{t_1}$, the NPI lower and upper probabilities for the event $L^2_{(t_1,t_2)} \geq bm_2$ are [11, 37]

$$\underline{P}(L^2_{(t_1,t_2)} \geq bm_2) = P(L^2_{(x^2_{j_{t_1}}, x^2_{j_{t_2}-1})} \geq bm_2) \tag{4.5}$$

$$\overline{P}(L^2_{(t_1,t_2)} \geq bm_2) = P(L^2_{(x^2_{j_{t_1}-1}, x^2_{j_{t_2}})} \geq bm_2) \tag{4.6}$$

For $j_{t_1} = 1$ and $j_{t_2} = 2$, these NPI lower and upper probabiliites are $\underline{P}(L^2_{(t_1,t_2)} \geq bm_2) = 0$ and $\overline{P}(L^2_{(t_1,t_2)} \geq bm_2) = P(L^2_{(-\infty, x^2_{j_{t_2}})} \geq bm_2)$.

For $I^3_l = (x^3_{l-1}, x^3_l), l = 1, \ldots, n_3 + 1$, and $t_2 \in I^3_{l_{t_2}} = (x^3_{l_{t_2}-1}, x^3_{l_{t_2}})$ where $l_{t_2} \in \{1, 2, \ldots, n_3\}$, the NPI lower and upper probabilities for the event $L^3_{t_2} \geq cm_3$ are [11, 37]

$$\underline{P}(L^3_{t_2} \geq cm_3) = \underline{P}(X^3_{(m_3 - \lceil cm_3 \rceil + 1)} > t_2) = \sum_{l=l_{t_2}+1}^{n_3+1} P(X^3_{(m_3-\lceil cm_3 \rceil+1)} \in I^3_l) \tag{4.7}$$

$$\overline{P}(L^3_{t_2} \geq cm_3) = \overline{P}(X^3_{(m_3 - \lceil cm_3 \rceil + 1)} > t_2) = \sum_{l=l_{t_2}}^{n_3+1} P(X^3_{(m_3-\lceil cm_3 \rceil+1)} \in I^3_l) \tag{4.8}$$

where $\lceil cm_3 \rceil$ is the smallest integer greater than or equal cm_3. For $l_{t_2} = 1$, these NPI lower and upper probabilites are $\underline{P}(L^3_{t_2} \geq cm_3) = 1 - P(X^3_{(m_3-\lceil cm_3 \rceil+1)} \in I^3_1)$ and $\overline{P}(L^3_{t_2} \geq cm_3) = 1$, and for $l_{t_2} = n_3 + 1$, they are $\underline{P}\left(L^3_{t_2} \geq cm_3\right) = 0$ and $\overline{P}\left(L^3_{t_2} \geq cm_3\right) = P\left(X^3_{(m_3-\lceil cm_3 \rceil+1)} \in I^3_{n_3+1}\right)$.

We obtain the two optimal thresholds, t_1 and t_2, for the three ordered classes C_1, C_2 and C_3 by maximising either the NPI lower probability, Equation (4.1), or the NPI upper probability, Equation (4.2). One needs to search for the values t_1 and t_2 that maximise the lower or the upper probability within each of the $n_1 + n_2 + n_3 + 1$ intervals created by the data observations. However, as shown in [11, 37], the optimal threshold t_1 can only be in intervals where the left-end value of the interval is an observation from class C_1 and the right-end value is an observation from class C_2, that is $t_1 \in (x^1, x^2)$. On the other hand, the optimal threshold t_2 can only be in intervals where the left-end value of the interval is an observation from class C_2 and the right-end value is an observation from class C_3, that is $t_2 \in (x^2, x^3)$. In the following, we provide an example to illustrate the NPI method for selecting the optimal thresholds for three classes, as presented above. For further explanations, examples and discussions of this method, we refer to the PhD thesis of Alabdulhadi [11], and the paper published by Coolen-Maturi et al. [37]. Further fundamental

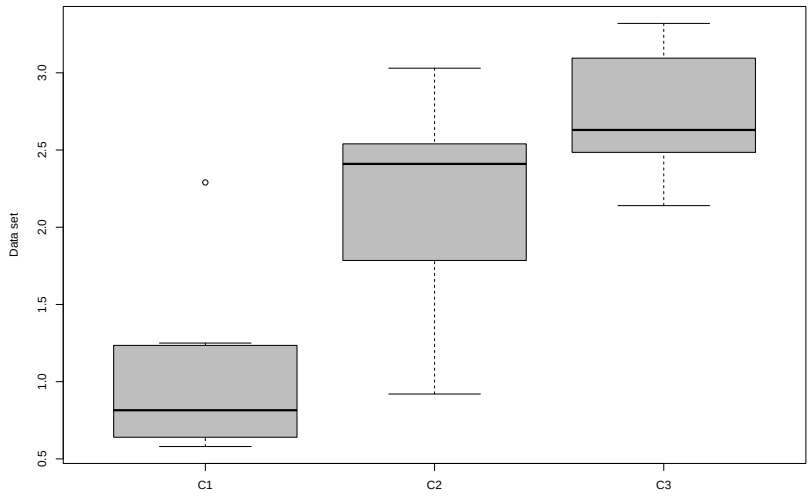


Figure 4.1: Box-plots of data set for the three classes.

properties of this method are also presented in [11, 37].

Example 4.2.1 Assume that we have a real-valued data set from three ordered classes C_1, C_2 and C_3, with 22 observations, where $n_1 = 8$ and $n_2 = n_3 = 7$, consisting of the data $\{0.58, 0.59, 0.69, 0.80, 0.83, 1.22, 1.25, 2.29\}$ for C_1, $\{0.92, 1.46, 2.11, 2.41, 2.43, 2.65, 3.03\}$ for C_2 and $\{2.14,2.45, 2.52, 2.63, 3.04, 3.15, 3.32\}$ for C_3. The box plots in Figure 4.1 show the overlap between the data for the three classes. There is a slight overlap between the three classes, particularly between classes C_2 and C_3.

Table 4.1 presents the results of the optimal threshold values t_1 and t_2 obtained from the NPI method along with their corresponding lower and upper probabilities, for $m_1 = 8, m_2 = 7$ and $m_3 = 6$. To illustrate the NPI method, we have considered four different scenarios for the target proportions a, b and c. For the first scenario, we have chosen small values for the target proportions, $a = b = c = 0.30$, leading to optimal threshold values $t_1 = 0.83$ and $t_2 = 2.43$. As the required target proportions are quite easy to achieve, the corresponding NPI lower and upper probabilities are high. The second scenario has $a = b = c = 0.7$, which are quite high,

Scenario	Target proportions			NPI lower method			NPI upper method		
#	a	b	c	t_1	t_2	value	t_1	t_2	value
1	0.30	0.30	0.30	0.83	2.43	0.53	0.83	2.43	0.84
2	0.70	0.70	0.70	1.25	2.43	0.07	1.25	2.43	0.38
3	0.70	0.70	0.20	1.25	3.03	0.24	1.25	3.03	0.65
4	0.50	0.75	0.50	0.83	2.43	0.21	0.83	2.43	0.59

Table 4.1: Optimal thresholds (t_1, t_2) using NPI-based method, and corresponding values of the NPI lower and upper probabilities, for $m_1 = 8, m_2 = 7$ and $m_3 = 6$

Scenario	Target proportions			NPI lower mathod			NPI upper method		
#	a	b	c	t_1	t_2	value	t_1	t_2	value
1	0.30	0.30	0.30	0.83	2.43	0.89	0.83	2.43	0.98
2	0.70	0.70	0.70	0.83	2.43	0.06	0.83	2.43	0.41
3	0.70	0.70	0.20	1.25	3.03	0.31	1.25	3.03	0.79
4	0.50	0.75	0.50	0.83	3.03	0.65	0.83	3.03	0.93

Table 4.2: Optimal thresholds (t_1, t_2) using NPI-based method, and corresponding values of the NPI lower and upper probabilities, for $m_1 = m_2 = m_3 = 10$

so the NPI method attempts to make a balance between the three classes to meet the required target proportions and to find the optimal thresholds that maximise the NPI lower probability, given in Equation (4.1), and the NPI upper probability, given in Equation (4.2). The corresponding NPI lower and upper probabilities are smaller than for the first scenario, which is due to the fact that the required target proportions are larger. The third scenario has $a = b = 0.70, c = 0.20$, so there is emphasis on the number of correctly classified future observations from class C_1 and class C_2 than from class C_3. This leads to t_2 being larger than in the other scenarios. The final scenario has $a = 0.50, b = 0.75, c = 0.50$, so it puts more emphasis on the number of correctly classified future observations from class C_2 than that from C_1 and C_3. It is noticed that the optimal threshold values and the NPI lower and upper probabilities changed again in order to meet the target proportions.

To further illustrate the results of the NPI method for selecting the optimal threshold values, we present in Table 4.2 the optimal threshold values along with

the corresponding NPI lower and upper probabilities for $m_1 = m_2 = m_3 = 10$, using the same target proportions presented in Table 4.1. The results are similar to those with $m_1 = 8, m_2 = 7$ and $m_3 = 6$ in Table 4.1. For $a = b = c = 0.30$ and $a = b = 0.70, c = 0.20$, the optimal thresholds are the same in both tables, but the corresponding NPI lower and upper probabilities are greater than those in Table 4.1. It should be clarified that the change in the NPI lower and upper probabilities here is due to the discrete nature of the event we consider, but it is not a direct effect of larger m, as larger m still needs the same proportion to be achieved. For example, for $m_1 = 8$, $m_1 = 10$ and $a = 0.3$, we need at least 3 good classifications in both cases, which is easier for $m_1 = 10$ than for $m_1 = 8$, so then for the latter the NPI lower and upper probabilities are probably smaller. For $a = b = c = 0.70$, the NPI method provides the same optimal threshold for t_2, but the optimal threshold for t_1 is different, while the values of the NPI lower and upper probabilities for both tables are quite similar. Finally, for $a = 0.50, b = 0.75$ and $c = 0.50$, the optimal threshold t_1 is the same in both tables, but the optimal threshold t_2 is different and the corresponding NPI lower and upper probabilities in Table 4.2 are high compared with those for the same scenario in Table 4.1. This section has presented the NPI method for selecting the optimal thresholds for data with three classes, giving illustrative examples. For more discussion about this method, we refer to [11, 37].

4.3 NPI-based classification trees with three classes

In this section, we extend the method presented in Section 3.3, for building classification trees with two classes, to build classification trees with three classes, using the NPI method for selecting the optimal thresholds t_1 and t_2 presented in Section 4.2. Note that maximising the NPI lower probability, Equation (4.1), or the NPI upper probability, Equation (4.2), are different criteria which may give different optimal thresholds. However, as in Chapter 3, we only consider the optimal threshold corresponding to the NPI lower probability, given in Equation (4.1).

Following the same notation as presented in Section 3.3, we need to add further notation. Assume that there is a data set, $\mathcal{D}$, with a set of continuous attribute variables $\{X_1, \ldots, X_f\}$ and a class variable C, where $C \in \{C_1, C_2, C_3\}$. The three classes are assumed to be fully independent and have a natural ordering, indicated by $C_1 < C_2 < C_3$. These classes could be e.g. low, medium, large or healthy, mild disease, disease. We first divide the data set $\mathcal{D}$ into two subsets: training data set S and testing data set T. For the training data set S, let n_1, n_2 and n_3 denote number of observations that belong to class C_1, C_2 and C_3, respectively, where $\sum_{i=1}^{3} n_i = n$. We set the values of m_1, m_2 and m_3 the number of future observations in C_1, C_2 and C_3, respectively, based on the distribution of the training data set S, that is $m_1 = n_1, m_2 = n_2$ and $m_3 = n_3$. This is because we do not actually know the number of future observations and the classes to which future observations belong.

To select the two optimal thresholds, t_1 and t_2, that maximise the NPI lower probability, Equation (4.1), we first need to choose the values of a, b and c for C_1, C_2 and C_3, respectively. The process of choosing particular values of a, b and c depends on a person's opinion of the relative importance of correct classification for the different classes. However, predefining these values may not enhance classification performance, i.e. the NPI method may perform poorly when the target proportions are too small or too large. For small values of a, b and c, the NPI method may achieve the target proportions for each class, but it may not maximise the total classification accuracy. On the other hand, for large values of a, b and c, the NPI method may not achieve the target proportions for each class, but it may tend to achieve the target proportions for one or two classes. The following example illustrates this issue further, where the goal is to show the total classification accuracy using different scenarios of the values of a, b and c.

Example 4.3.1 Assume that we have a data set consisting of 21 observations and three classes, C_1, C_2 and C_3. Suppose that the training data set, with $n_1 = 4, n_2 = 5$ and $n_3 = 4$, consisting of the data {1,2,3,5}, {4,6,7,8,10} and {9,11,12,13} for C_1, C_2 and C_3, respectively, and that the testing data set consists of the data {1,3,5},

Scenario	Target proportions			Optimal thresholds		Accuracy
#	a	b	c	t_1	t_2	(%)
1	0.25	0.25	0.50	3	8	75.00
2	0.75	0.75	0.75	3	10	87.50
3	0.60	0.33	0.70	5	8	87.50

Table 4.3: Comparisons of different scenarios of predefined a, b and c values

{7,8,10} and {11,12} for C_1, C_2 and C_3, respectively. So, we have $m_1 = 4, m_2 = 5$ and $m_3 = 4$. We consider three different scenarios of values of a, b and c. Table 4.3 presents the results of each scenario and the optimal thresholds along with their corresponding classification accuracy. The first scenario has small values for the target proportions, $a = b = 0.25$ and $c = 0.50$, leading to optimal thresholds $t_1 = 3$ and $t_2 = 8$, and the classification accuracy is 75.00%. The second scenario has $a = b = c = 0.75$. The optimal threshold t_2 is now large because we have $c = b$ which is different from the first scenario that has $c > b$. This change leads to a change in the classification accuracy which increased to 87.50%. Finally, the third scenario has $a = 0.60, b = 0.33$ and $c = 0.70$, as it puts more emphasis on the number of correctly classified future observations from class C_1 and C_3 than from class C_2, the optimal threshold value increased to $t_1 = 5$ because $a > b$, and the target proportions have been achieved for each class. The total classification accuracy is the same as in the second scenario.

As shown in Example 4.3.1, the predefined choice of the a, b and c values does not enhance classification accuracy. Therefore, we will do the same as in Chapter 2, in which we choose the values of a, b and c that maximise the total classification accuracy.

4.3.1 Selecting the target proportions

Following the proposed method of choosing the values of a and b, presented in Section 3.3, we focus on choosing the values of a, b and c based on the data set used in classification tasks, not setting them in advance. We will choose the values that improve classification performance. Consider the NPI method for selecting the

optimal thresholds, t_1 and t_2, which is based on the NPI lower probability, Equation (4.1),

$$\underline{P}(L^1_{t_1} \geq am_1, L^2_{(t_1,t_2)} \geq bm_2, L^3_{t_2} \geq cm_3) =$$
$$\underline{P}(L^1_{t_1} \geq am_1) \times \underline{P}(L^2_{(t_1,t_2)} \geq bm_2) \times \underline{P}(L^3_{t_2} \geq cm_3)$$

where $P(L^1_{t_1} \geq am_1)$, $P(L^2_{(t_1,t_2)} \geq bm_2)$ and $P(L^3_{t_2} \geq cm_3)$ are given in Equations (4.3), (4.5) and (4.7), respectively, and a, b, c are any values in $(0, 1]$. Note that we now consider a, b and c as parameters instead of achieved target proportions. As these parameters play an important role in the total classification accuracy, we propose to choose these parameters that maximise the classification accuracy on the testing sets. This raises the question of how one can choose the target proportions that maximise the classification accuracy and how to validate their performance in classification trees. Similar to Chapter 3, we suggest to use double cross-validation procedure in order to train our classification algorithm and tune the parameters a, b and c.

The process of choosing the values of a, b and c using the two levels of the k-fold cross-validation procedure, where $k = 5$, is presented in Figure 4.2. As we see from the figure, the data set is divided into two levels of the 5-fold cross-validation procedure. In the first level, which is the outer level, we validate our method with optimal parameters of a, b and c, while in the second level, which is the inner fold, we use a search function, the Genetic Algorithm [47, 48], to tune the parameters a, b and c. In other word, the inner folds will return only the model with the best values for a and b to the outer folds, while the outer folds will to validate the model's quality. In the outer folds, we will get five different performance with different values of the parameters a, b and c. However, as we still want for a real data a best value of a, b and c, therefore we will take those results from the outer folds, and extracting the best a, b and c. Then we use them as the target proportions for the data set, as in D.

The GA is a search-based optimisation technique based on the rules of genetics and natural selection to provide solutions to problems [47, 48]. We provide in the appendix more details about how the GA method works to tune the values of a, b

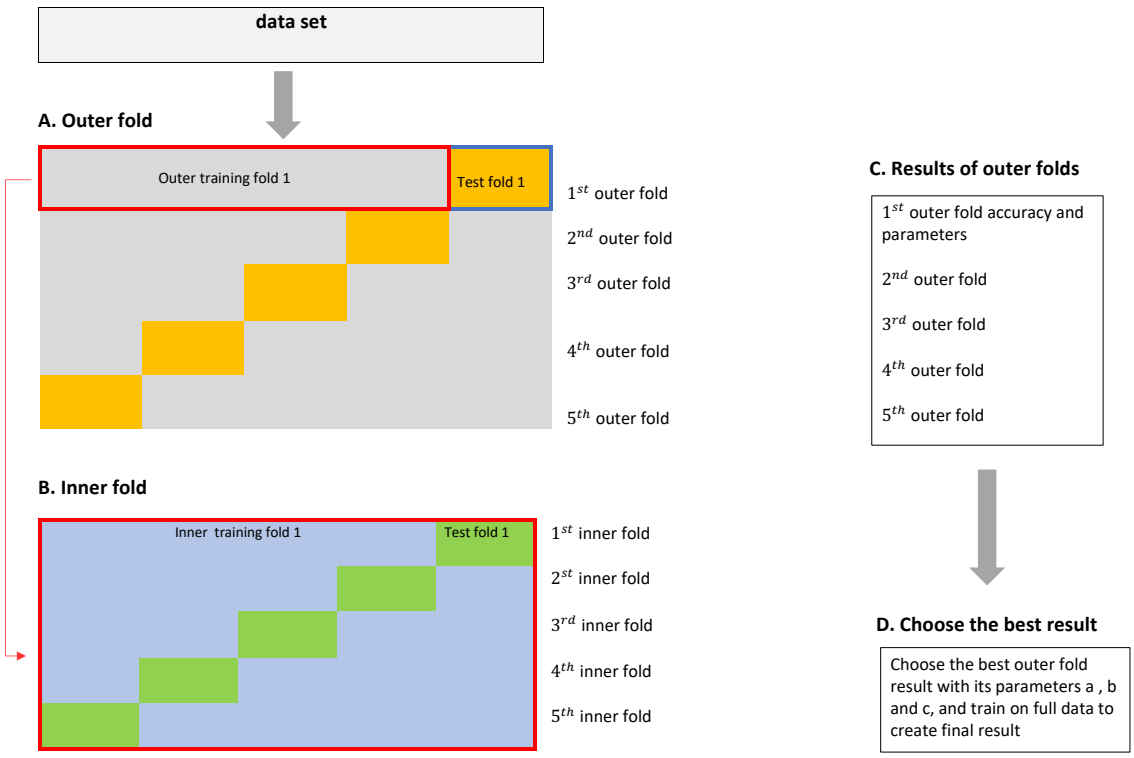


Figure 4.2: A diagram of a two-stage 5-fold cross-validation procedure to find the optimal values of the target proportions a, b and c.

and c in the inner level. Also, we provide examples to show the optimal values of a, b and c that maximise the classification accuracy using the GA method. After we have explained the process of choosing the parameters a, b and c based on a given data set, we present in Section 4.3.2 a new algorithm for building classification trees using the NPI approach for selecting the threshold values for three classes and our proposed method of selecting the values of a, b and c.

4.3.2 NPI$_3$-Tree algorithm

In this section, we present a new classification algorithm for building classification trees with data containing three classes, which we call Nonparametric Predictive Inference for building classification trees with three classes (NPI$_3$-Tree) algorithm. It is similar to the NPI$_2$-Tree algorithm, presented in Section 3.3.2, for building

classification trees with data containing binary classes, but the main difference is that the NPI$_3$-Tree algorithm works with data that contains three classes. In the NPI$_3$-Tree algorithm, we use the NPI method for selecting the optimal thresholds, t_1 and t_2, presented in Section 4.2, to find the optimal threshold values, and the method of choosing the values of a, b and c, presented in Section 4.3.1. The procedure of building classification trees by the NPI$_3$-Tree algorithm is similar to the well-known C4.5 algorithm, given in Section 2.3, but we use the NPI-based thresholds method, presented in Section 4.2, and the method of choosing the values of a, b and c, presented in Section 4.3.1. The following explains the procedure to build a classification tree using the NPI$_3$-Tree algorithm.

Assume that we have training data set, S, with n observations, which has continuous attribute variables, $\{X_1, \ldots, X_f\}$, and a class variable with three states, $C \in \{C_1, C_2, C_3\}$. The three classes are assumed to have a natural ordering, indicated by $C_1 < C_2 < C_3$. Let n_1, n_2 and n_3 denote the total number of observations from class C_1, C_2 and C_3, respectively. So we set the values of number future observations based on the size of the training sets, that is $m_1 = n_1, m_2 = m_2$ and $m_3 = m_3$. As a starting point for building a classification tree, we set the initial a, b and c values equal to the training set proportions S belonging to C_1, C_2 and C_3, respectively. For each continuous attribute variables, X_i, for $i = 1, \ldots, f$, in the training data set S, we find optimal threshold values, t_1 and t_2, by maximising the NPI lower probability, given in Equation (4.3.1). To search for the two optimal threshold values, t_1, and t_2, rather than searching for each value that maximising Equation (4.1) within each of the $n_1 + n_2 + n_3 + 1$ intervals produced by the data observations, we only consider the intervals as discussed in [11, 37], which is that the optimal threshold t_1 can only be in intervals where the left-end value of the interval is an observations from class C_1 and the right-end value is an observations from class C_2, that is $t_1 \in (x^1, x^2)$. On the other hand, the optimal threshold t_2 can only be in intervals where the left-end value of the interval is an observation from class C_2 and the right-end value is an observation from class C_3, that is $t_2 \in (x^2, x^3)$. Our classification algorithm uses this property in order to reduce the required computa-

tion to search the optimal thresholds.

After selecting the optimal thresholds for each attribute variable, X_i, we compute the value of the spliting criterion, IGR, (Equation (2.4)), for each attribute variable to find the most informative one. Once the IGR values are computed for each attribute variable, we choose the attribute variable with the highest value of the IGR for the root node, we then split the training data set S into three subsets as S_1, S_2 and S_3, where $S_1 \bigcup S_1 \bigcup S_3 = S$ and $S_1 \bigcap S_1 \bigcap S_3 = \emptyset$. Here S_1, S_2 and S_3 are subsets of S with $X \leq t_1, t_1 < X \leq t_2$ and $X > t_2$, respectively. For each subset, S_i for $i = 1, 2, 3$, we find the optimal thresholds again and compute the IGR for the existing attribute variables. The NPI$_3$-Tree algorithm continues recursively by splitting the training data set further and creating a new subtree for each branch not yet ending in a node. The tree branching stops when the observations in the subset all belong to a single class or the minimum split value is reached. The procedure for building the NPI$_3$-Tree classification tree is described in Algorithm 2, which is similar to Algorithm 1 used in Chapter 2, but the NPI$_3$-Tree algorithm works with data sets that contains three classes. In this chapter, we set the value of the minimum split number equal to five in order to prevent our classification algorithms from building large trees that may overfit the data and reduce classification performance.

4.3.3 Examples

In this section we provide two examples of the NPI$_3$-Tree algorithm. In the first example, Example 4.3.2, we illustrate how the NPI$_3$-Tree algorithm can be used to build a classification tree with the proposed method of choosing the values a, b and c presented in Section 4.3.1. In the second example, which is Example 4.3.3, we illustrate two different classification trees, one based on our method of choosing the values of a, b and c, and one based on the predefined method of the values of a, b and c. Note that the performance of the NPI$_3$-Tree algorithm is evaluated and compared with other classification algorithms in Section 4.4.

Algorithm 2 Pseudocode NPI$_3$-Tree algorithm

1. **Input**:($\mathcal{S}$, C, Ω)
2. $\mathcal{S}$: Training data set
3. C: A class variable $C = \{C_1, C_2, C_3\}$
4. Ω: Set of continuous attributes $\Omega = \{X_1, \ldots, X_f\}$
5. **Procedure** NPI$_3$-Tree($\mathcal{S}$, C, Ω)
6. Create a Root node for the tree
7. **if** all observations in $\mathcal{S}$ have the same class C, **then**
8. Return the single-node tree with class C
9. **if** Ω is empty (i.e. there are no attributes available), **then**
10. Return the single-node tree with most common class C in $\mathcal{S}$
11. Otherwise
12. Select the values of a, b, c and m_i for $i = 1, 2, 3$
13. Make the initial values of a, b and c equal to the class proportion in S,
14. i.e. make $a = \frac{n_1}{n}$, $b = \frac{n_2}{n}$ and $c = \frac{n_3}{n}$
15. Make the values of m_i equal to the number of observations in class C_i in S,
16. i.e. make $m_1 = n_1,\ m_2 = n_2$ and $m_3 = n_3$
17. **for** each attribute, X_i in Ω, **do**
18. Find the threshold values t_1 and t_2 that maximise the NPI lower probability, given in Equation (4.1)
19. Compute the IGR value using Equation (2.4)
20. Choose attribute variable X from Ω, with the highest IGR value
21. Assign the attribute X for the Root node
22. Add a branch below Root, corresponding to $X \leq t_1$, $t_1 < X \leq t_2$ and $X > t_2$,
23. Let S_i, for $i = 1, 2, 3$, be the subset of S that has $X \leq t_1$, $t_1 < X \leq t_2$ and $X > t_2$, respectively
24. **if** S_i is not empty, **then**
25. Add the subset created by NPI$_3$-Tree (S_i, C, $\Omega - \{X\}$)
26. **return** Root

Attribute	t_1	t_2	IGR value
X_1	5.5	6.1	0.41
X_2	2.9	3.3	0.41
X_3	1.9	4.7	0.84
X_4	0.6	1.7	0.86

Table 4.4: The optimal thresholds and the IGR values for all attributes variables

Example 4.3.2 This example illustrates the process of building classification trees using the NPI$_3$-Tree algorithm with the method of choosing the a, b and c values. We have used the Iris data set obtained from the UCI repository of machine learning databases [41]. The Iris data set has 150 observations and four attribute variables (sepals length, sepals width, petals length and petals width), denoted here as X_1, X_2, X_3 and X_4, respectively. The class variable has three classes, each contains 50 observations, which refers to the type of the Iris plant, denoted here as C_1, C_2 and C_3. First, we use the method proposed in Section 4.3.1 to find the optimal values for the target proportions a, b and c. Using this method, we have found that the optimal values that improve classification performance are $a = 0.77, b = 0.75$ and $c = 0.80$.

After selecting the values of a, b and c for the Iris data set, we divide the data into two subsets: a training data set, about 80% of the data, with $n_1 = 40, n_2 = 41$ and $n_3 = 39$, and a testing data set, about 20% of the data. We set the number of m_1, m_2 and m_3 future observations based on n_1, n_2 and n_3 values, so we have $n_1 = m_1 = 40, n_2 = m_2 = 41$ and $n_3 = m_3 = 39$. This binary split is only applied here to illustrate how we build the classification tree using the NPI$_3$-Tree algorithm, however, in Section 4.4, all classification algorithms are evaluated using the 10-fold cross-validation procedure, given in Section 2.5. Now, for the training set, we select the threshold values t_1 and t_2 that maximise the NPI lower probability, given in Equation (4.1), for all attribute variables, and we then compute the IGR values for all attributes using Formula (2.4). The optimal thresholds and the IGR values are presented in Table 4.4.

Next, to find the best attribute variable for splitting the data upon it, we choose the attribute variable that has the highest IGR value. We can see from Table 4.4 that the fourth attribute, X_4, has the highest IGR value. Thus, we assign X_4 as the root node in the tree, and we then split the training data set according to its optimal threshold values. So, we split the training data set into three subsets, with $X_4 \leq 0.6, 0.6 < X_4 \leq 1.7$ and $X_4 > 1.7$, respectively. Note that we do not use again X_4 for further splitting in the next levels of the classification tree. As the training data set has been split into three subsets, we again find the optimal thresholds and compute the IGR values for each subset. In the subsets of the training data set with $X_4 \leq 0.6$ and $X_4 > 1.7$, we get pure subsets, i.e. all observations in each subset belonging to a single class. Hence, we fix a leaf node for these subsets with the most common class, which are class C_1 and C_3, respectively. In the subset of the training data set with $0.6 < X_4 \leq 1.7$, we do not get a pure subset; in this case, we again calculate the optimal thresholds and the IGR values for this subset. Similarly, we select X_3 as a second split below the branch of $0.6 < X_4 \leq 1.7$ because it has the highest IGR value among the other attribute variables. We stop here the tree branching and terminate the tree because all observations in each subtree belong to the same class. The result of the complete classification tree is presented in Figure 4.3. Note that as we have only observations from two classes in X_3, in Figure 4.3, we deal here with this case as in Chapter 3, in which we use the NPI-based threshold selection method for two classes instead of three classes.

Example 4.3.3 This example illustrates two methods for choosing the target proportions for building a classification tree using the NPI_3-Tree algorithm. We first use the predefined choice for the a, b and c values to determine the thresholds and build the classification tree, and then we use our optimal way, presented in Section 4.3.1, in which we choose these values that improve classification performance. The data set used in this example consists of 125 observations, two attribute variables X_1 and X_2, and three ordered classes C_1, C_2 and C_3, which is obtained from the UCI repository of machine learning databases [41]. We first divide the data set into two subsets: training data set (80%), with $n_1 = 25, n_2 = 32$ and $n_3 = 43$, and testing data (20%) set. So, we have $m_1 = 25, m_2 = 32$ and $m_3 = 43$. Note that our aim

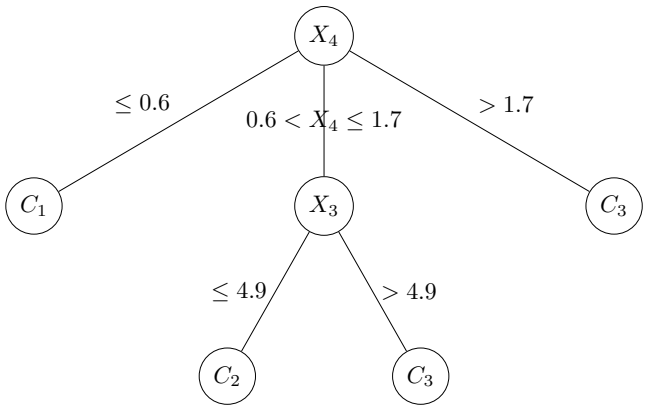


Figure 4.3: Classification tree of Iris data set created by the NPI_3-Tree algorithm, where $a = 0.77, b = 0.75$ and $c = 0.80$.

of this example is not to evaluate our method of choosing the values of a, b and c with the predefined choice for these values, but we aim to illustrate the resulting classification trees of these two ways. In the next section, we carry out experiments to examine the performance of the NPI_3-Tree algorithm on several real data sets, using the proposed method of choosing the a, b and c values, and compare its results with other classification algorithms.

For the first method, we have predefined the values for the target proportions equal to $a = b = c = 0.80$. The classification tree created by these proportions is given in Figure 4.4. We have found that the tree size qual to 4, and the classification accuracy, The classification accuracy, which is the ratio of the number of correctly classified observations to the total number of observations in the test data set, is 84%. For the second method, we have determined these target proportions based on our method, the optimisation technique, we found that the optimal target proportions are $a = 0.43, b = 0.87, c = 0.65$. As shown in Figure 4.5, the result of the classification tree is same as the first tree in terms of the tree size, but the classification accuracy increased to 92% compared to the first method.

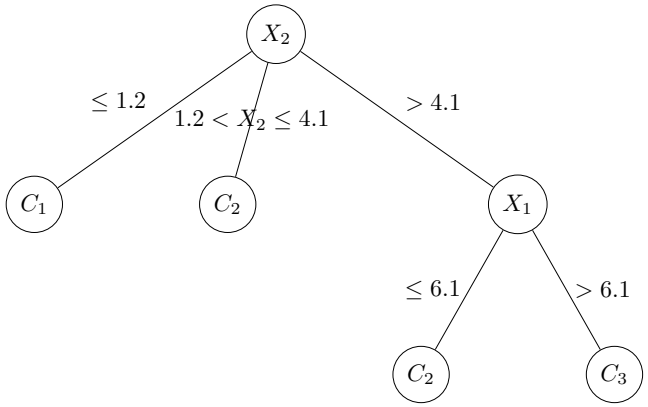


Figure 4.4: Classification trees created with the predefined choice of the a, b and c values, where $a = b = c = 0.80$, and the classification accuracy is 84%.

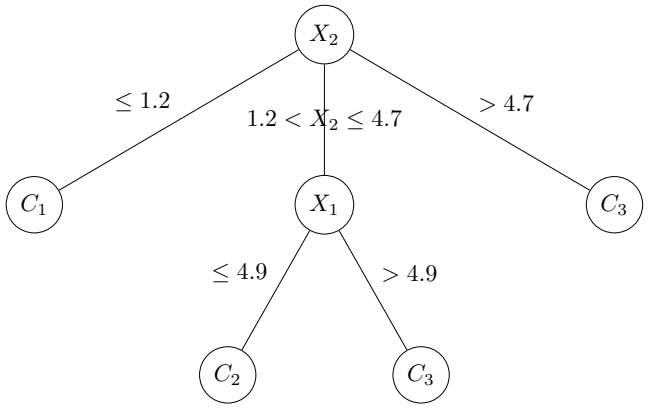


Figure 4.5: A classification tree created with the proposed method of choosing the a, b and c, where $a = 0.43, b = 0.87, c = 0.65$, and the classification accuracy is 92%.

4.4 Experiment

In this section, we carry out an experimental analysis to examine the performance of the NPI_3-Tree algorithm and to compare its performance with other classification algorithms on five data sets obtained from the UCI Machine Learning Repository [41]. We compare its performance with the most commonly used classical algorithms, the C4.5 and the CART algorithms, and with some classification algorithms based on imprecise probabilities, the NPI-M and the IDM1 algorithms. More information about these algorithms has been given in Section 2.2. The five data sets used in this experiment, namely Iris, Seeds, Wine, Contraceptive Method Choice (CMC) and Fitness data sets, are diverse in terms of their size and the number of attribute variables. Table 3.7 briefly summarises the main characteristics of these data sets, where Column 'N' gives the number of observations in the data set, column 'Attr' gives the number of continuous attribute variables, column 'Pro of class 1' gives the proportion of the data belonging to class C_1, column 'Pro of class 2' gives the proportion of the data belonging to class C_2 and column 'Pro of class 3' gives the proportion of the data belonging to class C_3. For further information and more details about these data sets, we refer to [41]. We only used five data sets because the NPI_3-Tree algorithm are only functions for continuous-valued attributes with three ordered classes, which are not commonly available in the public database. It would be interesting to fully automate the NPI_3-Tree algorithm to enable us to analyse more data sets including categorical attributes.

The statistical R software [69] has been used to carry out this experiment. All missing values were replaced with mean values using the missing value filter in R. All tied observations were dealt with by adding a small amount to the tied observations. For the NPI-M and the IDM1 algorithms, as they only handle categorical attributes, we discretised the continuous variables presented in Table 3.7 using the `mdlp` package in R and the 'discretization' function. This function converts a continuous variable into a categorical variable using the Fayyad and Irani method [43, 44], which discretize the continuous variables using a threshold value that minimises Equation (2.9) among all candidate thresholds. These pre-processing steps are essential and

Data set	N	Attr	Pro of class 1	Pro of class 2	Pro of class 3
Iris	150	4	0.33	0.33	0.33
Seeds	210	7	0.33	0.33	0.33
Wine	178	3	0.34	0.39	0.27
CMC	1473	2	0.42	0.24	0.34
Fitness	8020	10	0.36	0.41	0.23

Table 4.5: A brief description of the data sets.

were carried out at the beginning of the analysis for all algorithms to ensure a fair comparison. After that, we applied all classification algorithms to all data sets, and the results were compared in several ways.

The Fitness data set has originally four ordered classes and 13374 observations. However, as the NPI_3-Tree algorithm works only with three classes, we removed one class from this data, where the new data became 8020 observations and three ordered classes. Combining two neighbouring classes may be possible, but we leave this as future work. In addition, as the Fitness data set is large, we fix a minimum split number of 100 observations that must exist before splitting any node further. Otherwise, the tree is terminated, and the most frequent class in that node is fixed as a leaf node. A minimum split value has been fixed to reduce the calculation needed and avoid overfitting, as was done by Berry and Linoff [21], and by Bertsimas and Dunn [22]. However, all other classification algorithms will be applied to the Fitness data set using this minimum split value; hence the comparison will be fair for all algorithms. A minimum split value for all other data sets was fixed to 5, as was done in Chapter 2.

Five classification algorithms, which are the NPI_3-Tree, the C4.5, the CART, the NPI-M and the IDM1, have been used in this experiment. We used the `RWeka` package [51, 79] to build the C4.5 algorithm, the `rpart` package [75] to build the CART algorithm and the `imptree` package [45] to build both the NPI-M and IDM1 algorithms. The NPI_3-Tree classification algorithm has been built using the procedure presented in Section 4.3.2. In order to evaluate the performance of the NPI_3-Tree

Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.77	0.75	0.80	94.61	94.22	94.38	**94.69**	94.52
Seeds	0.81	0.79	0.78	**93.43**	89.72	90.42	92.63	92.38
Wine	0.94	0.68	0.87	**96.54**	93.12	91.14	95.19	94.64
CMC	0.43	0.36	0.52	49.96	**50.10**	48.40	49.81	49.81
Fitness	0.53	0.64	0.49	**79.81**	77.31	72.19	77.60	77.82
Average	-	-	-	**82.87**	80.89	79.30	81.98	81.83

Table 4.6: Average result of the classification accuracy of all algorithms and the optimal values of a, b and c for the NPI$_3$-Tree algorithm.

algorithm and all other algorithms, we have used three metrics. First, we used the classification accuracy, which is the ratio of the number of correctly classified observations to the total number of observations in the test data set. Further information about the classification accuracy has been given in Section 2.5. Secondly, we use in-sample accuracy, which is the classification accuracy on the training set. For more details about this metric, see Section 2.5. Finally, we used a tree size metric for each classification algorithm. Note that we refer here to the tree size as the total number of leaf nodes in the tree, as was done by Bertsimas and Dunn [22], and by Murthy and Salzberg [64]. A 10-fold cross-validation technique, as described in Section 2.5, has been used to obtain the final results from each of the three metrics.

Tables 4.6 present the results of the classification accuracies of the NPI$_3$-Tree algorithm and all the other classification algorithms for each data set. It also presents the target proportions a, b and c that correspond to the NPI$_3$-Tree algorithms. As shown in Table 4.6, the NPI$_3$-Tree algorithm performs better than the other algorithms for most data sets, and it achieves the highest average classification accuracy, followed by NPI-M, IDM1, C4.5 and CART, respectively. For the Iris data set, all classification algorithms including the NPI$_3$-Tree have similar results. This similarity between all algorithms' performance could be because the Iris data set has 150 observations, distributed equally across the three classes, and it has less overlap between their data classes. For this data set, the values of a, b and c are also similar. For the Seeds and Wine data sets, the NPI$_3$-Tree algorithm clearly outperforms the

classical algorithms, the C4.5 and the CART algorithms, and slightly performs better than the NPI-M and the IDM1 algorithms. For these data sets, the NPI_3-Tree classification algorithm built larger trees than the other classification algorithms which might be the reason why the NPI_3-Tree algorithm performs better than the other algorithms. For the CMC data set, all the classification algorithms including the NPI_3-Tree algorithm have not achieved good results, with the C4.5 algorithm slightly performing better than the other algorithms, which obtained classification accuracy of 50.10%. For this data set, the values of a, b and c are also low. We have analysed this data in-depth in order to give an insight into the characteristics of this data set that could be causing these results. The CMC data set has 1473 observations and two continuous attribute variables. However, this data has overlaps between their data observations' classes of more than half of the data. This could be the reason for these results which also match those observed in earlier studies, e.g. [8, 60]. Finally, for the Fitness data set, the NPI_3-Tree algorithm obtained the highest classification accuracy of 79.81%, where the C4.5, the NPI-M and the IDM1 algorithms have similar results. On the other hand, the CART obtained the worst result. Overall, according to the average classification accuracy, we can say that the NPI_3-Tree algorithm tends to perform better than the C4.5 and the CART algorithms, and it tends to perform slightly better than the NPI-M and the IDM1 algorithms.

Following [22, 64], we have evaluated the performance of the NPI_3-Tree algorithm on the training data set and compared it with the other classification algorithms. The in-sample accuracy measure, which is the performance of algorithms on the training set, is not commonly used to indicate classification accuracy, but it would be useful to give insight into how the classification algorithm works on the training set. It is well known that if the classification algorithm works very well on the training set but does not work very well on the testing set, it likely indicates overfitting. Therefore, it is useful to show the classification algorithms' performance on both training and testing sets, and hence, to check overfitting as well. Table 4.7 presents the results of the in-sample accuracy of the NPI_3-Tree algorithm and the

Data set	a	b	c	NPI_3-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.77	0.75	0.80	**94.82**	**94.82**	**94.82**	**94.82**	**94.82**
Seeds	0.81	0.79	0.78	90.66	**92.53**	92.11	91.16	91.98
Wine	0.94	0.68	0.87	93.54	94.56	**94.93**	92.40	92.40
CMC	0.43	0.36	0.52	51.68	52.39	51.68	**52.78**	**52.78**
Fitness	0.53	0.64	0.49	79.96	**80.77**	79.61	80.22	80.28
Average	-	-	-	82.13	**83.04**	82.63	82.27	82.45

Table 4.7: Average result of the in-sample accuracy for all classificaton algorithms and the optimal values for a, b and c to the NPI_3-Tree algorithm.

Algorithm	NPI_3-Tree	C4.5	CART	NPI-M	IDM1
Average	**3.66**	5.80	4.48	5.92	5.94

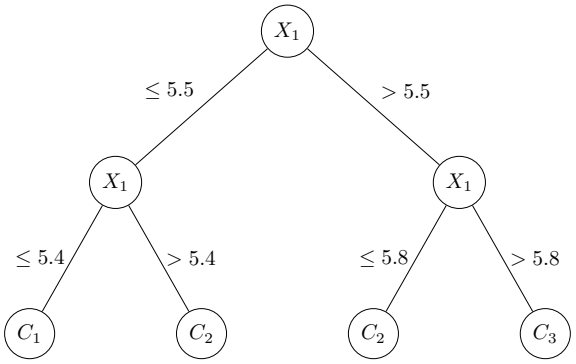
Table 4.8: Average result of the tree size of all classification algorithms.

other classification algorithms. All classification algorithms perform better on the training data sets than on the testing data sets, except the NPI_3-Tree algorithm whose performance on the train sets is slightly less than its performance on the test sets. This is due to the fact that the NPI_3-Tree algorithm selects the optimal thresholds by focusing on prediction, unlike the other classification algorithms which focus on maximising the correct classification of the training sets. The C4.5 algorithm has the highest average result of in-sample accuracy, followed by the CART, the IDM1, the NPI-M and the NPI_3-Tree algorithms. For the Iris data set, although all classification algorithms do not give the same trees, they have obtained the same result which is 94.82%. In this experimental analysis, it is noticed that the classical algorithm, the C4.5 and the CART, clearly perform better on the training data sets compared to their performance on the testing sets, which could be an indication they suffer from overfitting. From the results presented in Table 4.7, we can explain that the NPI_3-Tree has good results on both the training and testing data sets, which may indicate that the NPI_3-Tree algorithm does not overfit the data sets.

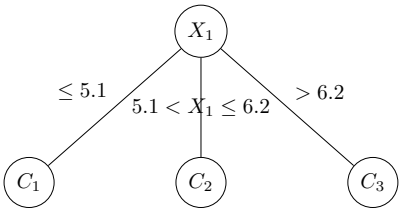
Finally, the average tree size for all the classification algorithms is given in Table 4.8. As we can see from the table, the NPI_3-Tree algorithm creates the smallest

trees, which has obtained the smallest result of the average tree size, followed by the CART, the C4.5, the NPI-M and the IDM1 algorithms. The use of a multiple split on continuous attributes which is used by the NPI_3-Tree algorithm could be one of the main reasons for creating the smallest trees by the NPI_3-Tree algorithm. The use of a multiple split on an attribute variable usually makes a classification tree smaller, easier to be understood and faster to build. Unlike the binary split which makes the trees larger because it allows an attribute variable to appear many times in the paths from the root of the tree to its leaf. For more clarification, we illustrate in Figure 4.6 two classification trees created by the C4.5 and the NPI_3-Tree algorithms for one attribute variable, X_1, of the Iris data. Figure 4.6(a) presents the classification tree created by the C4.5 which uses a binary split, e.g. two branches, while Figure 4.6(b) presents the classification tree created by the NPI_3-Tree algorithm, which uses a multiple split, e.g. three branches. As we can see from the two trees, the classification tree built by the NPI_3-Tree algorithm is smaller than the classification tree built by the C4.5 algorithm for the same data set.

To summarise, from the classification accuracy results presented in Table 4.6, we can conclude the following regarding the NPI_3-Tree algorithm's performance. The NPI_3-Tree algorithm performs well and slightly better than other classification algorithms. The NPI-M and IDM1 algorithms have quite similar results on all the data sets. With respect to the in-sample accuracy, the NPI_3-Tree algorithm is slightly performing less than other its performance on the testing data sets, which is due to the fact that the NPI_3-Tree algorithm selects the optimal thresholds by focusing on prediction rather than maximising the correct classification of the training sets. As the NPI_3-Tree algorithm has good results on both the classification accuracy and in-sample accuracy, this could indicate that the NPI_3-Tree algorithm does not suffer from overfitting. Finally, the NPI_3-Tree algorithm has the lowest average tree size, while the IDM1 algorithm has the highest average tree size.



(a) Tree with a binary split



(b) Tree with a multiple split

Figure 4.6: Classification trees with binary and multiple split.

4.5 Performance of NPI$_3$-Tree with the IIG

So far, the NPI$_3$-Tree has been tested using the classic split criterion, the IGR split criterion, which is based on precise probability theory. This section aims to get insight into the performance of the NPI$_3$-Tree algorithm using different split criteria based on imprecise probability, which is the Imprecise Information Gain Ratio (IIG) split criterion, introduced by Abellán and Moral [6]. The IIG is used as a split criterion for the NPI$_3$-Tree algorithm to build classification trees from an imprecise probability approach, using the maximum entropy distributions on credal sets obtained from the Imprecise Dirichlet Model (IDM) [1] or the Nonparametric Predictive Inference for multinomial data (NPI-M) [7]. For simplicity, and to avoid confusion, we will refer to the NPI$_3$-Tree algorithm using the IIG based on the NPI-M model as IIG$_{NPIM}$, and the NPI$_3$-Tree algorithm using the IIG based on the IDM1 model as IIG$_{IDM1}$. Further details about this split criterion have been presented in Section 2.3.

The procedure of building the NPI$_3$-Tree classification trees using the IIG split criterion is similar to the procedure of building the NPI$_3$-Tree classification trees using the IGR split criterion presented in Algorithm 2 (see Section 4.3.2), but the IGR in Algorithm 2 replaced by either the IIG_{NPIM} or the IIG_{IDM1}. To measure the performance of the NPI$_3$-Tree using the IIG_{NPIM} and the IIG_{IDM1} split criteria, the same five data sets presented in Table 4.5 are used. All tools used with the experimental analysis given in Section 4.4 are also used in this section. For example, we have applied the following pre-processing method: missing values were replaced with mean values using the missing value filter in R. Further, the tied observations were broken by adding a small amount to the tied observations. In addition, we have used the same metrics, which are the classification accuracy, in-sample accuracy and tree sizes, to measure the NPI$_3$-Tree algorithm's performance with the imprecise split criteria.

Table 4.9 presents the average result of the classification accuracy of the NPI$_3$-Tree algorithm using the imprecise split criteria, the IIG_{NPIM} and the IIG_{IDM1} split criteria. We have also provided the average classification accuracy of the NPI$_3$-Tree algorithm with the classical split criterion, the IIG split criterion, and the values of a, b and c. The NPI$_3$-Tree algorithm has a similar performance on all split criteria, where the NPI$_3$-Tree algorithm using the IIG_{NPIM} is performing slightly better than the other split criteria. For the Wine data set, the NPI$_3$-Tree algorithm has the same classification accuracy of 96.67% with both split criteria, the IIG_{NPIM} and the IIG_{IDM1}. For this data set, the NPI$_3$-Tree algorithm also built the same trees with these split criteria. For the CMC data set, it is noticed that the NPI$_3$-Tree algorithm with all split criteria again performs poorly. Also, all classification trees created for this data set were very large. What is interesting about the results presented in Table 4.9 compared to the results presented in Table 4.6 is that the NPI$_3$-Tree algorithm performs well with the three split criteria and outperforms the other classification algorithms presented in Table 4.6.

To investigate the performance of the NPI$_3$-Tree algorithm using the imprecise

Data set	a	b	c	IGR	IIG_{NPIM}	IIG_{IDM1}
Iris	0.77	0.75	0.80	**94.61**	94.58	94.43
Seeds	0.81	0.79	0.78	**93.43**	92.81	92.81
Wine	0.94	0.68	0.87	96.54	**96.67**	**96.67**
CMC	0.43	0.36	0.52	49.96	**50.26**	50.24
Fitness	0.53	0.64	0.49	79.81	**81.22**	80.66
Average	-	-	-	82.87	**83.10**	82.96

Table 4.9: Average result of the classification accuracy of the NPI_3-Tree with three split criteria, and the values of a, b and c.

split criteria and compare them with the classical split criterion, we have calculated in-sample accuracy for the NPI_3-Tree algorithm with all split criteria as presented in Table 4.10. The results of in-sample accuracy, which is the classification accuracy on the training set, are calculated to show how the NPI_3-Tree algorithm with the imprecise split criteria performs on the training set, and hence, to check on overfitting as well. As shown in Table 4.10, the average results of in-sample accuracy are similar for all split criteria and for all data sets. According to the average results on in-sample accuracy and classification accuracy, we can say that the NPI_3-Tree algorithm works well on all the split criteria, which might indicate that it does not suffer from overfitting. Finally, to compare different trees built by the different split criteria, the results of tree size (number of leaf nodes) for each split criterion are presented in Table 4.11. The NPI_3-Tree algorithm using the IGR split criterion creates the smallest trees, followed by the IIG_{NPIM} and the IIG_{IDM1} split criteria. It should be clarified that the NPI_3-Tree algorithm creates smaller trees than the other classification algorithms presented in Table 4.8, with all split criteria.

4.6 Concluding remarks

In this chapter, we have presented a new classification algorithm, which we have called the NPI_3-Tree algorithm, to build classification trees based on the NPI approach. This algorithm is an extension of the NPI_2-Tree algorithm, presented in

Data set	a	b	c	IGR	IIG_{NPIM}	IIG_{IDM1}
Iris	0.77	0.75	0.80	**94.82**	93.90	94.78
Seeds	0.81	0.79	0.78	90.66	**91.66**	**91.66**
Wine	0.94	0.68	0.87	**93.54**	92.40	92.40
CMC	0.43	0.36	0.52	51.68	51.17	**52.16**
Fitness	0.53	0.64	0.49	79.96	76.63	**78.19**
Average	-	-	-	**82.13**	81.15	81.83

Table 4.10: Average result of the in-sample accuracy of the NPI_3-Tree with three split criteria, and the optimal values of a, b and c.

Split criterion	IGR	IIG_{NPIM}	IIG_{IDM1}
Tree size	**3.66**	3.76	3.94

Table 4.11: Average result of the tree size of the NPI_3-Tree using different split criteria.

Chapter 3, but it has been developed for data with continuous-valued attributes and three ordered classes. The NPI_3-Tree algorithm uses the NPI approach for selecting optimal thresholds, where the inferences are explicitly in terms of a given number of future observations and target proportions. First, we have presented the NPI method for selecting the optimal thresholds for three ordered classes, presented by Alabdulhadi [11] and Coolen-Maturi et al. [37], with illustrative examples to show the method. We have then extended the procedure for selecting the optimal values of target proportions, introduced in Chapter 3, to data with three classes by choosing that to maximise classification performance on testing data sets. Finally, the NPI_3-Tree algorithm has been presented with an example showing how to build classification trees using this algorithm.

We have carried out an experimental analysis of five data sets taken from the UCI repository of machine learning databases [41]. We have evaluated the performance of the NPI_3-Tree algorithm on these data sets and compared its performance with other classification algorithms. We have used the classification accuracy, in-sample accuracy and tree size to measure the performance of all classification algorithms.

According to the results obtained from the experimental analysis, the NPI_3-Tree algorithm performs slightly better than other classification algorithms for most data sets in terms of classification accuracy. The results have also shown that all classification algorithms have similar results in terms of in-sample accuracy, where the other algorithms have slightly higher accuracies than the NPI_3-Tree algorithm. In terms of tree sizes, the results have indicated that the NPI_3-Tree algorithm tends to create smaller trees than other classification algorithms.

This work is explicitly on choosing the threshold, which requireds that there is a natural ordering of the attribute values with their classes: lowest values for the first class and greatest values for the last one. We have considered this type of data because it is important and exists in many practical situations. For non-ordered classes, we have not considered it yet, but it is probably a quite simple way, for example, we can consider a pre-processing step in which if the order is not the appropriate the variable is transformed in some way, e.g. taking the opposite value. This opportunity is left as a topic for future work. For non-ordered classes, we can refer to Abdulmajeed's thesis [12], but this work does not require threshold. One important topic for future work, which is a similar to one discussed at the end of Chapter 3, is to develop the NPI_3-Tree algorithm considering the misclassification cost. For example, classification aims to minimize misclassification costs in many practical applications instead of maximising the total classification accuracy. So, it would be of interest to develop the process of choosing the target proportions taking the misclassification cost into account. Another interesting topic to investigate is to compare the NPI_3-Tree classification algorithm with other classification methods that do not use classification trees. Although the NPI_3-Tree algorithm provides different inferences than other classification methods, such comparisons can provide valuable information and conclusion. Finally, it would be interesting to extend this research to apply the NPI_3-Tree algorithm to imprecise classification. In imprecise classification, trees might return a set of classes in leaf nodes rather than one class. More details about imprecise classification can be found in [63].

Chapter 5

NPI-based classification trees for noisy data

5.1 Introduction

In real-world applications of classification techniques, the data sets used to learn the classification algorithms are not always reliable and could suffer from several issues, the presence of noise is one of these issues. Noise refers to situations that occur when the data sets used for classification tasks have incorrect values in the attribute variables or the class variable. If the noise appears in attribute variables, it is referred to as attribute noise. If the noise appears in the class variable, it is referred to as class noise or label noise. There are several possible sources that could cause the class noise, e.g. the class noise may come from poor quality information or human errors in assigning the class label [4, 81]. The presence of noise can have substantial consequences on the classification algorithms' performance. It has been proved in previous studies that class noise has a more negative impact on classification algorithms' performance than attribute noise [14, 81]. This might due to the fact that each observation in the data set typically has several attributes, but usually only has one class variable, which might have significant importance for training purposes. One of the most commonly used methods for handling noisy data is to use a classification algorithm that is robust to noisy data [14]. It has been shown that some classification algorithms are more robust than others.

In this chapter, we consider applications of the NPI_2-Tree algorithm, presented in Chapter 3, and the NPI_3-Tree algorithm, presented in Chapter 4, on class noise. We evaluate the performance of the NPI_2-Tree and the NPI_3-Tree algorithms in terms of their classification accuracy using different levels of random noise added to the class variables. We then compare the performance of these algorithms with the C4.5, the CART, the NPI-M, and IDM1 algorithms, introduced in Section 2.2, using the same different levels of noise. We also investigate the performance of the NPI_3-Tree algorithm using different scenarios of adding the noise to the three ordered classes.

This chapter is organised as follows: Section 5.2 briefly reviews some of the literature on classification data noise. Section 5.3 presents some methods for adding noise to data sets. In Section 5.4, we present the results of the experimental analysis carried out to evaluate the performance of the NPI_2-Tree and the NPI_3-Tree algorithms on data sets with several levels of added random noise, and compare the results with other classification algorithms. In Section 5.5, we also present the results of the NPI_3-Tree algorithm's performance with different scenarios of adding the noise to the three ordered classes. Some concluding remarks and suggestions for future research are given in Section 5.6.

5.2 Noise in classification

In data classification problems, noise refers to situations that occur when the data sets used for classification task have incorrect values in attributes or incorrect class labels. If the noise appears in attribute variables, it is referred to as attribute noise. If the noise appears in the class variable, it is referred to as class noise or label noise. Several possible reasons may cause noise in data sets. For example, the noisy data may come from poor quality information, the data might have been measured incorrectly, or experts may wrongly assign the class label [4, 81]. In addition, according to [59], data encoding or network connectivity problems are major reasons

for data noise. It has been estimated that real-world data sets contain at least five percent of encoding errors [81]. It has been shown in previous studies that class noise has a more negative impact on classification performance than attribute noise [14, 81]. This effect is because each observation in the data set often tends to have several attributes, and the value of each attribute may have different importance for training purposes [14, 81]. On the other hand, for each observation in the data, there is usually only one class, so noise in the class variable can have a big effect on the performance of the classification [14, 81]. Of course, the level of impact depends on the level of class noise and the characteristic of the data set. In this chapter, we consider only noise on a class variable as it was done in [2, 4, 5, 14, 60, 61].

It is usually difficult to accurately detect observations that have a class noise because this detection usually needs an expert in the application field, however, due to a lack of such expertise or the difficulty of manually examining the information provided in data sets, different solutions have been proposed for handling class noise [14]. One method for handling class noise is using a noise filter technique. In this method, possibly noisy observations are identified and removed from the training data sets before constructing the classification trees. Another method is to use a noise correction technique. This method is similar to the previous method and first attempts to identify noisy observations, but instead of deleting them completely from the data, it changes their current class labels with the probably right class label. The main challenge is, of course, to decide which observations are noisy. Usually, this challenge can be solved using classification algorithms themselves. An observation is considered to be noisy if a classification algorithm is wrongly classifying it, and can then be dealt with by either removing it from the data or changing its class label to the possible correct class label. However, this could lead to 100% in-sample accuracy and hence likely poor performance on the data set in reality. Another way to handle class noise is to use an algorithm that is robust for noisy data. The robustness of algorithms with regard to data noise would be attractive for machine learning methods to include such robust algorithms. It should be noted that the goal of this chapter is neither to detect which observation is noisy, nor to

remove or correct class noise from training data. Our aim is to examine how the use of noisy data sets affects the performances of the classification algorithms. We study the performance of the NPI_2-Tree and the NPI_3-Tree algorithms when they are used with noisy data sets, and compare their performance to other classification algorithms used on the same noisy data sets. The classification accuracy measure is used to examine the performances of the classification algorithms on noisy data sets. We compare the performance of the algorithms using the classification accuracy results obtained from the original (noise-free) data set with those obtained from the same data set with some added noise. The classification algorithm is considered robust if the classification accuracy results for noisy data are close to those for the original data set.

The performance of classification algorithms on noisy data sets is expected to be less accurate than their performance on noise-free data sets. It is rare to find classification algorithms that are not sensitive to noisy data; however, some algorithms are more robust to noise than others. Mantas and Abellán [61] have carried out several experimental analyses to check the performance of the Credal-C4.5 algorithm, which uses a new split criterion based on imprecise probability, and compared its performance with the C4.5 and the ID3 algorithms. The comparison was carried out by adding different levels of noise to the class variable. The results showed that on the original data sets, all classification algorithms achieved similar results, while on the noisy data, the Credal-C4.5 algorithm performs better than other classification algorithms. In the work of Abellán and Masegosa [4, 5], they have shown an application of bagging credal classification tree on data sets with class noise. The idea of bagging, also known as Bootstrap aggregating, is to generate multiple versions of a model, and then use these models to get an aggregated model [5]. Abellán and Masegosa [4, 5] examined their method and compared it with similar methods using the C4.5 classification algorithm, and they found that their method is better than other bagging methods on data sets with a class noise. It is interesting to compare the NPI_2-Tree and the NPI_3-Tree algorithms using bagging approaches, however, we leave such comparisons for future study. For more studies about the

performance of classification algorithms on data with noisy class variables, we refer to [2, 4, 5, 14, 60, 61].

5.3 Adding noise to data sets

In order to study the performance and robustness of the classification algorithms on noisy data, we need to add noise to the data set because many data sets may not contain much noise or it is difficult to know which observations are noisy or not. Adding noise to data sets allows us to investigate the impact of noisy data on classification algorithms' performance and hence we can determine which algorithms are robust in terms of noisy data, and we can also look for potential methods to improve the performance of classification algorithms on noisy data sets. Several methods have been introduced in the literature for adding noise to data sets, in this section, we briefly present some of these methods. In this thesis, we have restricted attention to class noise because this type of noise appears in most real-world data sets and it can affect the performance of classification algorithms more than attribute noise. It is also interesting to study and investigate the impact of attribute noise on the performance of the classification algorithms, as this type of noise appears in most real-world data sets, but we leave this for future work.

One of the most commonly used methods for adding noise to a class variable is presented by Abellán and Masegosa [4, 5], Mantas and Abellán [61] and Mantas et al. [60]. In this method, a specific percentage $x\%$ of random noise is added to the class variable in the training set, while the testing set stays unchanged for evaluation purposes. The process to add the noise into the class variable is as follows: first, they randomly choose a subset of the observations, based on a given percentage, and then, they randomly replace the classes of these selected observations by other possible classes with equal probability. In this chapter, this method is used to add noise to the class variable in the data set. We randomly choose a given percentage of observations in the training set and then we flip their current classes to other possible

classes. Note that in our experimental analysis in this chapter, the real percentage will exactly match the theoretical percentage. This means that we exactly corrupt $x\%$ of observations since there is no chance for their original classes to be chosen. Different levels of added noise have been considered in the literature, ranging from 5% to 50% [2]. In this chapter, we consider four levels of noise: 10%, 15%, 30% and 50%, which are added only to the training set. In Section 5.4, we present more details about using this method in our work. In Section 5.5, we also consider a different method of adding noise to the data that has three ordered classes. In such a method, we randomly choose a given percentage of observations, and then we change their classes to other possible classes based on predefined probabilities. More details about this method are given in Section 5.5. Zhu and Wu [81] introduced another method of adding noise to class variables. They add noise to the data using a pair of classes. For given a noise percentage $x\%$ and a pair of classes, observations with the first class have an $x\%$ chance of being changed to the second class, and observations with the second class have an $x\%$ chance of being changed to the first class. For example, for given classes (C_1, C_2), and noise 30%, observations with class C_1 has 30% chance to be changed to class C_2, so do observations in the second class. This method is used because in some situations only specific types of classes are likely to be mislabeled. Another method for adding noise to a class variable was introduced by Sáez et al. [72]. They add noise to a class variable using a *uniform class noise method*, which replaces the class labels of the observations by randomly changing a class by another one from the existing classes using uniform distribution, and a *pairwise class noise method*, which the majority class observations are changed to the second majority class. For more explanations about the uniform class noise and pairwise class noise methods, we refer to [81].

5.4 Experimental analysis

This section studies the performance of the NPI_2-Tree algorithm, presented in Chapter 3, and the NPI_3-Tree algorithm, presented in Chapter 4, when they are used to build classification trees involving noisy data. We also compare the performance of

these algorithms with four classification algorithms: the C4.5, CART, NPI-M and IDM1 algorithms. These algorithms have been given in Section 2.2, and they were used in Chapter 3 and Chapter 4 for our experimental analyses. We use different levels of noise to measure and evaluate the performance and the robustness of these classification algorithms with noisy data. We begin this section by presenting the way that the experiments have been carried out, and then we explain how we add the noise to the class variable. After that, we present and discuss the experimental results of the performance of the NPI_2-Tree and the NPI_3-Tree algorithms and other classification algorithms.

5.4.1 Experimental setup

In these experiments, we have used the same six data sets, presented in Chapter 3, and the same five data sets, presented in Chapter 4. The NPI_2-Tree algorithm and other classification algorithms have been applied to the six data sets since the NPI_2-Tree algorithm can work only with two classes, while the NPI_3-Tree algorithm and other classification algorithms have been applied to the five data sets since the NPI_3-Tree algorithm can work only with three classes. The characteristic of the six and five data sets have been given in Tables 3.7 and 4.5, respectively. The statistical R software [69] has been used to carry out these experiments. All R packages and the pre-processing steps used in Chapters 3 and 4 have been also used to conduct our experiments.

In order to check the performance of these classification algorithms on noisy data, we introduce some noise to the class variables of these data sets. We add noise to the data because we do not know whether they contain noise or not, and if they contain noise we do not know how much noise they contain. Therefore, we do not assume any percentage of noise in the data sets, hence, we use these data sets as containing no noise. Therefore, the following percentages of noise are added to the class variable: 10%, 15%, 30%, and 50%. These noise levels are only added to the training data sets, and the testing data sets are not changed. Of course, it is enough to add noise to the class variable up to 30%, as was done in many studies in the

literature such as [4, 5, 14, 60, 61], but we add noise levels up to 50% because we want to evaluate these algorithms also on a high level of noise as it was done by Abellán [2]. The procedure of adding the noise into a class variable is as follows: we first randomly select $x\%$ of the observations in the training set, where x refers to the needed level of the noise. Then, we replace their class labels with a different class from the existing classes with equal probability, except the actual class label. For example, assume that we have a data set with three classes, C_1, C_2 and C_3. If we want to add noise to observation with class C_1, then this observation has a equal chance to go either to class C_2 or C_3, but no chance for class C_1. We add the different noise levels to the training sets only, while the testing sets stay unchanged in order to be a fair comparison when we assess the performance of the classification algorithms.

In our experiments, we only use the most important evaluation measure which is the classification accuracy. The classification accuracy, which is the ratio of the number of correctly classified observations to the total number of observations in the test data set, is used to evaluate the performance of the classification algorithms on noisy data sets. It would be also useful to consider other evaluation metrics to evaluate the performance of these algorithms on noisy data. A 10-fold cross-validation technique, as presented in Section 2.5, is used to obtain the final results of the classification accuracy. With respect to the NPI_2-Tree and NPI_3-Tree algorithms, we first use the double-5-fold cross-validation procedure introduced earlier to find the optimal values of the target proportions with noisy data. The classification accuracy results of the classification algorithms on the original training sets (noise-free, denoted as Noise level 0%) are used as a reference to measure the robustness of these algorithms with the different noisy levels of data sets. A classification algorithm is considered to be robust if it achieves similar accuracy results with both the noisy data sets and noise-free data sets. This method of comparing and measuring the performance of the classification algorithms on the noisy data sets has been also used by Sáez et al. [72].

The results of our experiments are presented in two phases. First, in Section 5.4.2, we present the results of the performance of the NPI_2-Tree, C4.5, CART,

NPI-M and IDM1 algorithms on the six data sets that have two classes. After that, in Section 5.4.3, we present the results of the NPI_3-Tree, C4.5, CART, NPI-M and IDM1 algorithms on the five data sets that have three classes.

5.4.2 Results for the NPI_2-Tree algorithm

In this section, we study and present the performance of the NPI_2-Tree algorithm with noisy data, and we compare its performance to the performance of the C4.5, CART, NPI-M and IDM1 algorithms. We have carried out our experiments using the same six data sets presented in Table 3.7, where we add different levels of noise to the class variable only in training data sets in the 10-fold cross-validation procedure. Table 5.1 presents the classification accuracy results of the NPI_2-Tree algorithm and the other classification algorithms for each data set and for each level of noise, 10%, 15%, 30%, and 50%. We have also provided in this table the results of the classification accuracy of these algorithms based on the original data sets, denoted as Noise level 0%, and the optimal values of the target proportions a and b, corresponding to the NPI_2-Tree algorithm. Figure 5.1 shows a summary of the classification accuracy results for all the classification algorithms on all the levels of noise. All results given in this section were obtained using the 10-fold cross-validation procedure. The best results are highlighted in bold font.

We see from the results in Table 5.1 that all classification algorithms' performances decrease when the level of noise increases. However, the results show that the NPI_2-Tree, the NPI-M and the IDM1 algorithms are more robust to noise than the C4.5 and the CART algorithms for the most noise levels. As shown in Figure 5.1, the almost parallel lines indicate similar robustness, particularly for the three algorithms, the NPI_2-Tree, the NPI-M and the IDM1 algorithms. For low levels of noise (10% and 15%), the classification accuracy results are similar, so we comment on these results together. At these levels of noise, the NPI_2-Tree algorithm achieves the best average classification accuracy followed by the NPI-M, the IDM1, the C4.5 and the CART algorithms. For these levels of noise, the performance of the NPI_2-Tree, NPI-M and IDM1 algorithms are quite similar to their performance on the

Data set	a	b	NPI_2-Tree	C4.5	CART	NPI-M	IDM1
Noise level 0%							
Breast Cancer	0.57	0.73	87.10	86.89	86.90	87.65	**87.86**
Cryotherapy	0.56	0.50	79.38	80.11	**83.48**	81.45	80.18
Blood Transfusion	0.79	0.64	**89.48**	75.43	74.76	79.56	79.56
Haberman's Survival	0.84	0.42	75.19	75.62	73.18	**76.39**	**76.39**
Liver Patients	0.32	0.65	**80.70**	77.25	76.43	80.28	80.28
QSAR Biodeg	0.65	0.82	**91.22**	73.13	77.86	82.16	82.16
Average	-	-	**83.84**	78.07	78.72	81.24	81.08
Noise level 10%							
Breast Cancer	0.82	0.48	85.13	83.32	70.35	**86.80**	84.12
Cryotherapy	0.52	0.65	80.10	79.56	78.38	80.23	**80.73**
Blood Transfusion	0.61	0.48	**87.32**	73.51	71.90	76.11	77.49
Haberman's Survival	0.87	0.61	74.37	**75.76**	73.16	75.32	73.81
Liver Patients	0.52	0.79	77.56	75.82	69.41	**79.12**	78.39
QSAR Biodeg	0.79	0.86	**90.14**	72.13	75.38	83.12	84.19
Average	-	-	**82.43**	76.68	73.09	80.11	79.80
Noise level 15%							
Breast Cancer	0.54	0.61	84.19	81.94	70.10	**85.91**	84.12
Cryotherapy	0.48	0.57	**80.54**	76.93	76.14	79.30	79.65
Blood Transfusion	0.44	0.49	**85.29**	72.19	68.81	78.15	75.11
Haberman's Survival	0.71	0.69	74.14	73.38	71.84	**74.59**	72.95
Liver Patients	0.43	0.18	75.33	73.16	66.25	**79.63**	78.06
QSAR Biodeg	0.82	0.54	**88.75**	71.13	73.68	81.93	82.27
Average	-	-	**81.37**	74.78	71.13	79.81	78.69
Noise level 30%							
Breast Cancer	0.43	0.27	70.28	60.74	65.82	**73.15**	70.53
Cryotherapy	0.31	0.46	70.39	65.57	71.28	72.93	**74.33**
Blood Transfusion	0.52	0.39	**71.81**	64.32	65.16	65.60	67.81
Haberman's Survival	0.62	0.46	55.22	63.69	58.96	**67.15**	64.55
Liver Patients	0.31	0.19	57.11	54.82	56.13	70.13	**74.18**
QSAR Biodeg	0.58	0.63	**73.35**	60.41	64.10	67.41	67.12
Average	-	-	66.36	61.56	63.57	69.39	**69.75**
Noise level 50%							
Breast Cancer	0.19	0.24	51.21	47.15	49.26	54.54	**56.30**
Cryotherapy	0.24	0.17	45.14	46.68	53.28	48.68	**51.98**
Blood Transfusion	0.43	0.49	58.91	52.35	48.40	**61.39**	53.71
Haberman's Survival	0.53	0.36	47.12	**58.16**	42.24	46.14	46.59
Liver Patients	0.30	0.72	43.80	41.33	40.71	55.43	**57.42**
QSAR Biodeg	0.46	0.61	53.22	55.18	51.97	**57.38**	56.43
Average	-	-	49.90	50.14	47.64	**53.92**	53.73

Table 5.1: Classification accuracy results obtained by the NPI_2-Tree, C4.5, CART, NPI-M and IDM1 algorithms, with different levels of added noise.

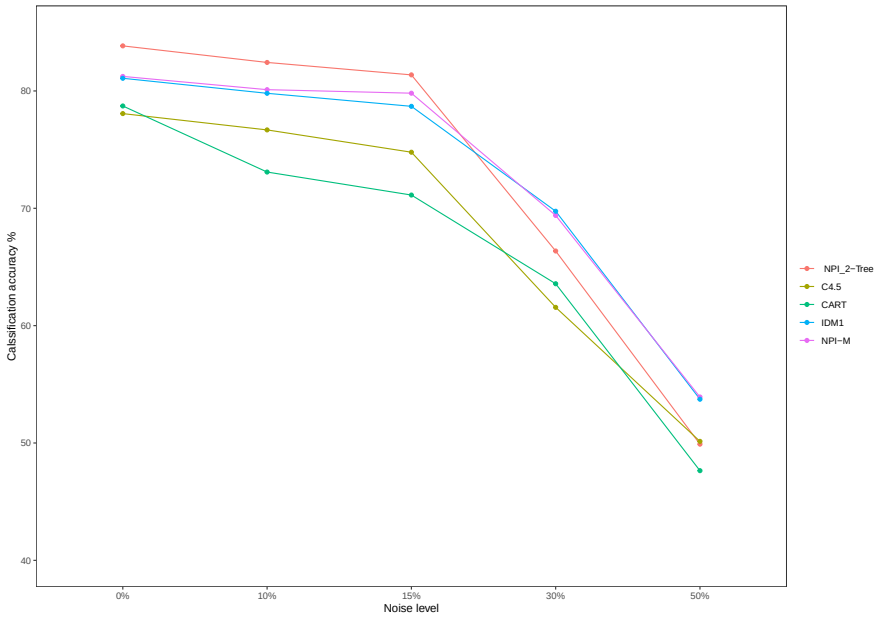


Figure 5.1: Classification algorithms performance with different levels of added noise, two classes case.

original data sets (Noise level 0%). For the Cryotherapy data set, it is noticed that the performance of the NPI_2-Tree algorithm has increased with 10% and 15% noise levels, but decreased after that with 30% and 50%. This increase in the classification results would be due to the randomness in the data and is more likely to occur for small data sets and low levels of noise. For this data set, we have also noticed that the IDM1 algorithm has increased with 10% noise level. In general, one would expect more noise to lead to worse results, but this can vary from one application to another. It is possible that such a difference in accuracy results might occur in this data set because it is relatively small with only 90 observations and 4 attributes. The unexpected increase was as noticed in some previous studies such as [5, 60, 61], when the performance of some classification algorithms increased with some noise levels.

For medium level of noise (Noise 30%), as can be seen from Table 5.1, the NPI_2-Tree algorithm achieves the best classification accuracy on two data sets as it attains a higher average classification accuracy than the classical algorithms, the C4.5 and the CART. For the Breast Cancer and the Haberman's Survival data sets, the IDM1

algorithm performs slightly better than all other classification algorithms, and for the Cryotherapy and Liver Patients data sets, the NPI-M algorithm performs slightly better than other classification algorithms. However, for the Blood Transfusion and QSAR Biodeg data sets, the NPI_2-Tree algorithm achieves the best classification accuracy when the noise level is 30%. For these two data sets, it is noticed that the NPI_2-Tree algorithm performs clearly better than other algorithms for all the noise levels considered, excluding noise level 50%, which may be due to the fact that these data sets are large and they have little overlap between their data classes. This better performance of the NPI_2-Tree algorithm on these two data sets suggests that the NPI_2-Tree algorithm is more robust than other algorithms to noise levels up to 30% when the data sets are large and have less overlap. For high level of noise (Noise 50%), all classification algorithms perform less well than their performance with the other levels of noise. It should be clarified that adding noise to the class variable up to 30% seems to be sufficient to evaluate the robustness of the classifiers, as was done in many studies in the literature, such as [4, 5, 14, 60, 61], however, we add noise levels up to 50% to give insight into how the classification algorithms perform of this level of noise, as also done by Abellán [2]. Note that in binary classification with 50% of noise, the data might not contain useful information, so any difference between classifiers may be due to randomness. For 50% level of noise, the NPI-M and IDM1 algorithms have obtained the highest classification accuracy, whereas the CART algorithm is the worst.

In this section, we have presented the performance of the NPI_2-Tree with noisy data sets, and we have compared its performance to other classification algorithms. From the results given in Table 5.1, we can make the following conclusions about the performance of the NPI_2-Tree algorithm on data sets with different levels of class noise. The NPI_2-Tree algorithm performs quite well and can be considered a robust algorithm for most levels of noisy data. For data sets with noise levels equal to 0% (the original data sets), 10% and 15%, the NPI_2-Tree algorithm has the best classification accuracy results. However, when the class noise level is 30%, the NPI-M and IDM1 algorithms are performing better than the other algorithms, but

with a performance very close to the NPI_2-Tree algorithm for the same class noise level. For 50% noise level, the NPI-M and IDM1 algorithms are performing better than other classification algorithms, where the CART algorithm is the worst. In the following section, we present the performance of the NPI_3-Tree algorithm when it is applied to data sets with class noise, and we compare its performance to the performance of classification algorithms.

5.4.3 Results for the NPI_3-Tree algorithm

In this section, we study the effect of class noise on the performance of the NPI_3-Tree algorithm when applied to data sets that have three classes. We have carried out experiments on the same five data sets presented in Table 4.5, and also the same noise levels, 0% (the original data sets), 10%, 15%, 30%, and 50%, as used in Section 5.4.2. To add noise to a class variable with three states, we first randomly select x% observations in the training set, where x is the level of noise required. After that, we replace their class labels with different classes from the existing classes, excluding the actual class label. The performance of the NPI_3-Tree algorithm is also compared to the performances of the C4.5, CART, NPI-M and IDM1 algorithms, in a similar approach taken in Chapter 4.

Table 5.2 presents the classification accuracy results obtained from all classification algorithms based on noisy data sets with different levels of class noise equal to 0%, 10%, 15%, 30%, and 50%. The classification accuracy results given in this table are obtained using the 10-fold cross-validation procedure, and the best results are highlighted in bold font. The optimal values of target proportions a, b and c, corresponding to the NPI_3-Tree algorithm, are also given in the table. Figure 5.2 shows the average classification accuracy for each classification algorithm and for each level of noise. Table 5.2 shows that the NPI_3-Tree algorithm performs well and achieves the highest average accuracy rate for most class noise levels, compared to the other classification algorithms. It performs better than the C4.5 and the CART algorithms at all levels of noise. For the original data sets (0%) and low levels of noise (10% and 15%), the NPI_3-Tree algorithm performs slightly better than the

other classification algorithms, followed by the NPI-M, IDM1, C4.5 and CART algorithms, respectively. Note that the optimal values of target proportions a, b and c were not much affected by these low levels of noise, compared to the corresponding values without added noise. For the CMC data set, it is noticed that all classification algorithms perform less on all levels of noise including the noise-free data set. This could be because it is a large data set with 1474 observations and two attribute variables, but more than half of their data classes overlap, hence, the classification algorithms are likely to perform poorly. Such less performance on this data set was also noticed by other studies, e.g. by Abellán et al. [8] and by Manats et al. [60]. For this data set, it is also noticed that the optimal values of target proportions a, b and c are low.

At the medium level of noise (Noise 30%), the IDM1 algorithm performs slightly better than the other classification algorithms, but its performance is very close to the NPI_3-Tree and the NPI-M algorithms. This result is consistent with that of Mantas and Abellán [61], who found that the IDM1 achieved the best average classification accuracy for many different data sets when the level of random noise is 30%. At this level of noise, the NPI_3-Tree algorithm achieves the best accuracy result on the Fitness data set, the largest data set among these data sets. This finding matches the result of the earlier experiment, presented in Section 5.4.2, where the NPI_2-Tree algorithm also performed better than the other classification algorithms for large data sets, and noise level 30%. These results mostly indicate that the NPI_3-Tree algorithm performs slightly better for 30% class noise than other algorithms in large data sets. It would be interesting to analyse more large data sets in order to give an insight into the performance of the NPI_3-Tree algorithm with large data sets and 30% class noise. One possible investigation for this case, which is left for future research, is to study the impact of adding class noise on the process of selecting the optimal thresholds for the NPI_3-Tree algorithm. For example, one can study the effect of each class noise separately on the process of selecting the optimal thresholds, by adding the noise to only one class and then examine the performance results of the NPI_3-Tree algorithm, and so on for the other classes.

Data set	a	b	c	NPI_3-Tree	C4.5	CART	NPI-M	IDM1
Noise level 0%								
Iris	0.77	0.75	0.80	94.61	94.22	94.38	**94.69**	94.52
Seeds	0.81	0.79	0.78	**93.43**	89.72	90.42	92.63	92.38
Wine	0.94	0.68	0.87	**96.54**	93.12	91.14	95.19	94.64
CMC	0.43	0.36	0.52	49.96	**50.10**	48.40	49.81	49.81
Fitness	0.53	0.64	0.49	**79.81**	77.31	72.19	77.60	77.82
Average	-	-	-	**82.87**	80.89	79.30	81.98	81.83
Noise level 10%								
Iris	0.81	0.75	0.68	93.60	91.70	92.16	93.33	**93.72**
Seeds	0.93	0.63	0.72	**92.13**	85.20	87.40	90.60	90.23
Wine	0.91	0.87	0.83	**94.28**	92.68	91.78	92.70	91.11
CMC	0.55	0.32	0.62	48.12	46.92	**48.30**	47.45	47.83
Fitness	0.68	0.45	0.47	**79.22**	76.12	69.94	77.10	77.31
Average	-	-	-	**81.47**	78.52	77.91	80.23	80.04
Noise level 15%								
Iris	0.79	0.66	0.72	**93.49**	86.35	85.29	91.64	91.87
Seeds	0.88	0.70	0.63	**92.22**	83.49	82.75	89.10	86.76
Wine	0.92	0.81	0.74	**92.10**	90.28	89.44	90.28	88.40
CMC	0.43	0.51	0.48	46.83	44.19	47.98	47.70	**48.08**
Fitness	0.59	0.43	0.55	78.43	75.91	74.28	77.12	76.94
Average	-	-	-	**80.61**	76.04	75.95	79.16	78.41
Noise level 30%								
Iris	0.74	0.59	0.60	88.19	84.14	83.14	89.93	**90.85**
Seeds	0.81	0.58	0.74	83.36	76.80	77.78	**82.42**	82.19
Wine	0.72	0.69	0.43	**91.13**	89.23	85.84	87.18	89.16
CMC	0.21	0.28	0.40	41.20	45.43	44.31	47.67	**49.55**
Fitness	0.37	0.56	0.31	**75.88**	70.34	71.67	75.82	75.11
Average	-	-	-	75.95	73.18	72.54	76.60	**77.37**
Noise level 50%								
Data set	a	b	c	NPI_3-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.63	0.48	0.81	**74.13**	61.40	59.31	69.87	69.13
Seeds	0.59	0.36	0.68	71.23	59.32	57.80	70.15	**72.67**
Wine	0.48	0.45	0.27	58.48	67.16	49.43	**73.84**	73.34
CMC	0.42	0.31	0.23	**53.12**	33.52	30.92	52.13	50.17
Fitness	0.35	0.45	0.29	55.36	58.40	57.91	**59.67**	59.24
Average	-	-	-	62.46	55.86	51.07	**65.13**	64.91

Table 5.2: Classification accuracy results obtained by the NPI_3-Tree, C4.5, CART, NPI-M and IDM1 algorithms, with different levels of added noise.

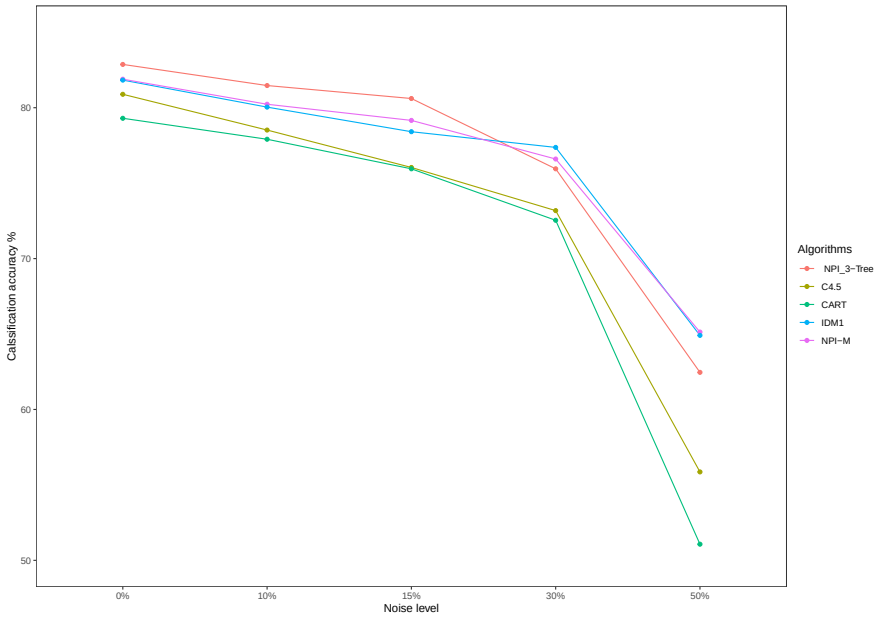


Figure 5.2: Classification algorithms performance with different levels of added noise, three classes case.

Finally, for the high level of noise (Noise 50%), the NPI-M and the IDM1 perform better than the other classification algorithms, where the NPI_3-Tree algorithm has a similar result to these algorithms. There is a noticeable difference in the performances of the C4.5 and CART algorithms at this level of noise compared to their performance at other noise levels. For example, for 30% class noise, their average accuracy results are 73.18% and 72.54%, respectively, however, for 50% class noise, their average accuracy results have dropped to 55.86% and 51.07%, respectively. To summarise, the NPI_3-Tree algorithm performs well and can be considered to be a quite robust algorithm for building classification trees with class noise. It has performed best at levels of noise 0% (the original data sets), 10% and 15%, and it has performed better than the classical algorithms on all noise levels. The next section of this chapter presents further analysis of the NPI_3-Tree algorithm using different scenarios of adding the noise to a class variable.

5.5 NPI$_3$-Tree with noisy neighbour labels

In many real situations, the observations are more likely to be wrongly classified to a neighbouring class than to a class further away. For example, in medical diagnostics, it may occur more often to wrongly classify values that are close to a threshold than if such a value is further away from the threshold. If there are three ordered classes of temperature and two thresholds, below 37.5 indicates healthy, 37.5 to 38.5 mild disease, and above 38.5 serious disease, then someone with an actual temperature 37.4 is more likely to be wrongly classified as mild diseased than as diseased. From this perspective, it would be useful in this section to evaluate the performance of the NPI$_3$-Tree and the other algorithms in such situations. We aim to consider different scenarios with different probabilities for adding noise, higher to a neighbouring class and lower to a class further away. This method can only be performed on the data sets with three ordered classes, as for data with two classes there is only one possible wrongly classification. Therefore, we only examine the NPI$_3$-Tree in this section.

To evaluate the performance of the NPI$_3$-Tree algorithm in such cases, we have used three different scenarios with different probabilities for adding the noise to a class variable, higher to neighbouring classes and lower to classes further away. These three different scenarios are given in Table 5.3, where p_{ij} is the probability of replacing observations' class labels from a class i (for $i = 1, 3$) to a class j (for $j = 1, 2, 3$), excluding the actual class label, i.e. $i \neq j$. Note that, since the middle class has two neighbouring classes, it is misclassified to the other class with equal probability, i.e. with probability 0.5. These scenarios are used to investigate how the NPI$_3$-Tree algorithm performs on neighbouring noisy classes. However, it would be interesting to consider more different scenarios for the probabilities of adding noise to neighbouring classes, but we leave this for future research. The method of adding noise into a class variable with these three scenarios is given as follows: we first randomly select $x\%$ of the observations in the training set, where x is the needed level of the noise. We then replace their class labels with new classes from the available classes based on the probabilities of the three scenarios presented in Table 5.3.

# Scenario	p_{ij} if $\lvert i-j\rvert = 1$	p_{ij} if $\lvert i-j\rvert = 2$
1	0.6	0.4
2	0.8	0.2
3	1	0.0

Table 5.3: Different scenarios of adding noise to the class variable.

Using these three scenarios for adding noise to the class variable, we have tested the NPI$_3$-Tree algorithm on the five data sets presented in Table 4.5, using the following noise levels added to the training sets only: $10\%, 15\%, 30\%$ and 50%. We compared the performance of the NPI$_3$-Tree algorithm with the C4.5, the CART, the NPI-M and the IDM1 algorithms for all three scenarios, similar to the method taken in Section 5.4.3. Tables 5.4, 5.5 and 5.6 present the average results of the classification accuracy of all classification algorithms for each data set, each scenario and each level of noise. Figure 5.3 provides graphs showing the accuracy results of all algorithms for each scenario and each level of noise. The results presented in the tables were obtained using a 10-fold cross-validation procedure. Bold font indicates the best results.

For Scenario 1, the probability of replacing observations' classes with neighbouring ones is 0.60, and the probability of replacing observations' classes with ones further away is 0.40. The results in Table 5.4 show that all classification algorithms have very similar results to their performances in the case of random class noise presented in Section 5.4.3. For Scenario 2, we have increased the chance of replacing observations' classes with neighbouring classes, i.e. the probability of replacing observations' classes with neighbouring classes increased to 0.80, while the probability of replacing observations' classes with ones further away decreased to 0.20. From Table 5.5, the results of the average accuracy show that, when the levels of noise are 10% and 15%, the NPI$_3$-Tree algorithm performs slightly better than the other classification algorithms. However, for 30% of noise, the results are clearly different compared to the same level for the first scenario. Here, the NPI$_3$-Tree algorithm

achieves the highest classification accuracy in four out of five data sets and performs similarly to the imprecise algorithms in the CMC data set. At this level of noise, it has obtained the highest average accuracy 75.64% among the other algorithms, followed by the NPI-M, IDM1, CART and C4.5 algorithms, respectively. It is noticed that all other classification algorithms' performances decreased for 30% of noise compared to their performances for the same level in the first scenario. For example, the average classification accuracy of the C4.5 algorithm in the first scenario is 72.34%, but in the second scenario, it dropped to 69.10%. For the high level of noise, 50%, the NPI-M and the IDM1 algorithms perform clearly better than the C4.5 and CART algorithms, but with very close results to the NPI$_3$-Tree algorithm. For Scenario 3, we have again increased the chance of replacing observations' classes with neighbouring classes, i.e. the probability is 1, while the probability of replacing observations' classes with ones further away is now 0. The results in Table 5.6 show that, for 10% and 15% noise levels, the NPI$_3$-Tree performs slightly better than other algorithms, although the performances of all algorithms are quite similar. For 30% noise, the NPI$_3$-Tree algorithm again obtains the highest average result of classification accuracy compared to the other classification algorithm. Finally, with 50% noise level, all algorithms achieve poor results, although the IDM1 performs slightly better than the other algorithms.

In this section we have given an insight into how the position of noisy data can affect the performance of the NPI$_3$-Tree algorithm, and compared its performance to other classification algorithms. We have used three scenarios with different probabilities for adding the noise to the class variable, higher to neighbouring classes and lower to classes further away. According to the results presented in Tables 5.4 to 5.6, we can say that the NPI$_3$-Tree algorithm is performing well for these three scenarios, and mostly better than the other algorithms for most noise levels. For levels of noise equal to 10% and 15%, the NPI$_3$-Tree algorithm performs better than the other classification algorithms in all scenarios. When the level of the noise is increased to 30%, the NPI$_3$-Tree algorithm performs also better than the other classification algorithms in all scenarios, except the first scenario, which performs quite

similarly to the NPI-M and IDM1 algorithms. Finally, the NPI$_3$-Tree algorithm performs better than the C4.5 and CART algorithms when the level of noise equal is 50%, but the IDM1 and NPI-M algorithms perform better in this case.

Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Noise level 10%								
Iris	0.92	0.82	0.64	93.13	93.98	92.75	**93.52**	**93.52**
Seeds	0.87	0.55	0.74	**92.95**	89.63	87.42	91.32	91.58
Wine	0.74	0.63	0.96	**92.87**	91.50	91.18	91.64	91.43
CMC	0.52	0.31	0.64	49.30	48.63	**49.50**	48.71	48.19
Fitness	0.58	0.42	0.60	**79.38**	77.32	70.82	78.21	78.10
Average	-	-	-	**81.52**	80.21	78.33	80.68	80.56
Noise level 15%								
Iris	0.80	0.78	0.71	**93.24**	89.74	87.22	91.32	91.15
Seeds	0.83	0.76	0.71	**92.28**	83.51	82.29	90.17	87.16
Wine	0.87	0.84	0.76	92.20	91.57	91.61	**92.84**	92.76
CMC	0.49	0.50	0.55	50.41	48.19	50.98	50.70	**51.18**
Fitness	0.50	0.45	0.66	**78.21**	74.63	73.90	77.19	77.23
Average	-	-	-	**81.26**	77.47	77.20	80.44	79.89
Noise level 30%								
Iris	0.70	0.72	0.69	85.21	83.10	84.80	**89.87**	89.12
Seeds	0.81	0.58	0.74	**84.36**	80.80	79.21	83.50	83.85
Wine	0.72	0.69	0.43	**86.48**	83.23	85.84	85.54	84.46
CMC	0.21	0.28	0.40	42.63	44.21	44.61	**47.67**	47.53
Fitness	0.37	0.56	0.31	73.14	70.34	71.67	**73.38**	73.17
Average	-	-	-	74.36	72.34	73.22	**75.99**	75.62
Noise level 50%								
Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.45	0.30	0.61	**77.62**	72.19	57.50	75.87	74.13
Seeds	0.59	0.36	0.68	**75.63**	67.18	69.98	73.65	73.11
Wine	0.58	0.49	0.39	61.12	64.53	56.36	69.21	**69.84**
CMC	0.46	0.51	0.33	39.75	47.52	45.92	50.16	**51.98**
Fitness	0.38	0.65	0.26	54.65	53.12	51.98	**58.17**	59.14
Average	-	-	-	62.09	60.90	56.34	65.41	**65.64**

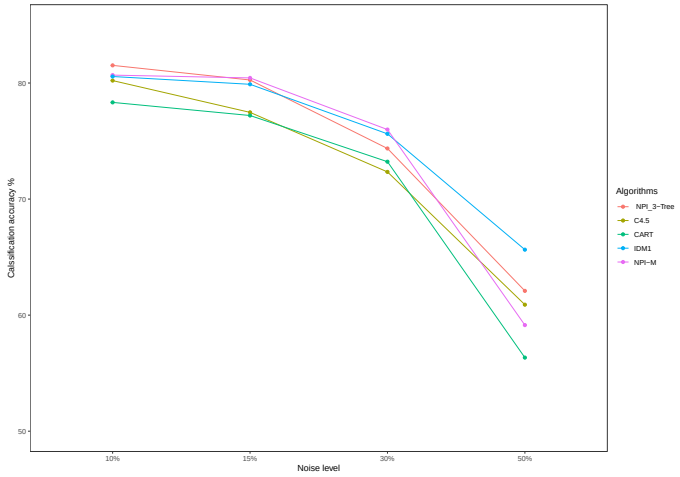
Table 5.4: Accuracy results of classification algorithms, first scenario.

Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Noise level 10%								
Iris	0.88	0.76	0.72	93.26	91.84	90.37	**93.58**	93.19
Seeds	0.93	0.61	0.80	**93.46**	85.29	87.40	91.12	90.23
Wine	0.85	0.67	0.92	92.16	92.75	91.88	92.37	**92.81**
CMC	0.52	0.30	0.64	49.34	49.34	48.50	**50.21**	**50.21**
Fitness	0.68	0.45	0.47	**79.76**	76.32	67.12	78.55	79.43
Average	-	-	-	**81.59**	79.10	77.05	81.16	81.17
Noise level 15%								
Iris	0.80	0.78	0.71	93.26	89.55	90.29	92.54	**92.87**
Seeds	0.88	0.70	0.63	**92.56**	86.49	86.75	89.10	90.76
Wine	0.92	0.81	0.74	**92.64**	90.28	89.44	92.81	92.81
CMC	0.43	0.51	0.48	50.19	**51.39**	47.98	51.33	51.21
Fitness	0.59	0.43	0.58	78.43	74.91	69.28	77.12	**79.14**
Average	-	-	-	**81.88**	78.10	76.74	80.58	81.35
Noise level 30%								
Iris	0.83	0.79	0.74	**86.58**	71.90	76.23	86.34	88.24
Seeds	0.85	0.53	0.78	**83.42**	78.85	82.21	82.93	83.16
Wine	0.70	0.60	0.79	**89.38**	80.23	82.84	82.63	**83.22**
CMC	0.38	0.45	0.44	45.51	44.21	41.61	**46.67**	46.55
Fitness	0.57	0.49	0.37	**73.33**	70.34	70.67	71.32	69.11
Average	-	-	-	**75.64**	69.10	70.71	73.97	73.85
Noise level 50%								
Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.45	0.30	0.61	59.50	49.88	52.27	64.21	**65.47**
Seeds	0.59	0.36	0.68	70.98	65.18	64.98	72.65	**73.14**
Wine	0.58	0.49	0.39	63.79	65.73	59.12	**73.81**	69.34
CMC	0.46	0.51	0.33	47.98	40.19	46.92	50.76	**51.98**
Fitness	0.38	0.65	0.26	50.40	60.86	61.49	**63.17**	58.32
Average	-	-	-	62.40	56.36	56.95	**64.92**	63.65

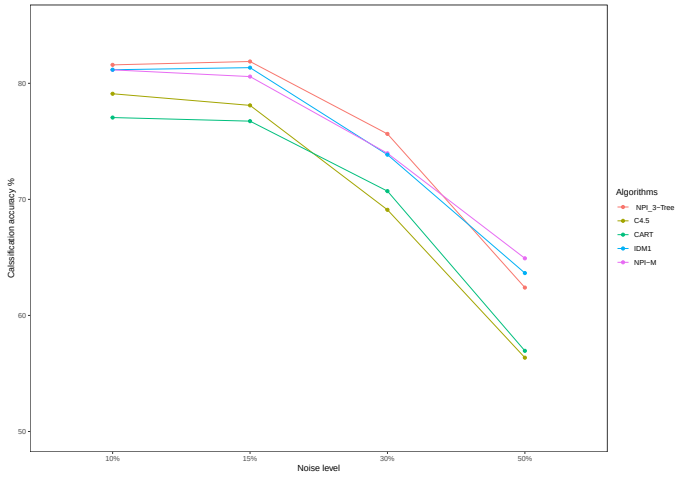
Table 5.5: Accuracy results of classification algorithms, second scenario.

Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Noise level 10%								
Iris	0.91	0.72	0.68	92.19	91.80	90.73	93.22	**93.34**
Seeds	0.86	0.64	0.84	**92.18**	86.11	89.23	91.18	90.17
Wine	0.85	0.71	0.84	**91.50**	90.78	90.92	91.33	89.41
CMC	0.56	0.32	0.69	50.19	49.63	48.50	48.36	**50.23**
Fitness	0.63	0.47	0.38	**79.64**	76.39	67.32	77.54	78.30
Average	-	-	-	**80.86**	78.94	77.34	80.32	80.29
Noise level 15%								
Iris	0.76	0.68	0.75	91.90	85.32	84.29	92.99	**93.10**
Seeds	0.81	0.56	0.60	**89.37**	84.49	83.75	87.12	86.77
Wine	0.90	0.69	0.73	**90.43**	88.46	89.61	90.32	89.70
CMC	0.61	0.39	0.42	50.77	48.92	46.67	51.39	**52.68**
Fitness	0.58	0.37	0.65	**80.10**	75.88	74.97	79.73	78.94
Average	-	-	-	**80.51**	76.61	75.85	80.31	80.23
Noise level 30%								
Iris	0.69	0.43	0.59	80.24	78.96	78.21	80.13	**80.67**
Seeds	0.83	0.51	0.63	**79.32**	76.19	76.27	78.43	78.98
Wine	0.66	0.49	0.50	78.62	76.25	76.12	79.28	**79.72**
CMC	0.42	0.29	0.61	**49.43**	47.21	48.69	46.14	47.50
Fitness	0.40	0.37	0.63	55.41	**55.54**	53.67	54.87	52.15
Average	-	-	-	**68.60**	66.83	66.59	67.77	67.80
Noise level 50%								
Data set	a	b	c	NPI$_3$-Tree	C4.5	CART	NPI-M	IDM1
Iris	0.32	0.58	0.43	**63.19**	62.40	61.31	62.87	62.14
Seeds	0.49	0.18	0.37	65.25	65.46	65.98	66.65	**66.77**
Wine	0.33	0.23	0.59	55.32	56.52	56.89	61.16	**58.24**
CMC	0.21	0.09	0.56	43.21	45.13	47.91	45.68	**48.33**
Fitness	0.32	0.28	0.41	49.15	45.17	47.68	**53.39**	52.58
Average	-	-	-	56.89	54.93	55.75	57.95	**57.61**

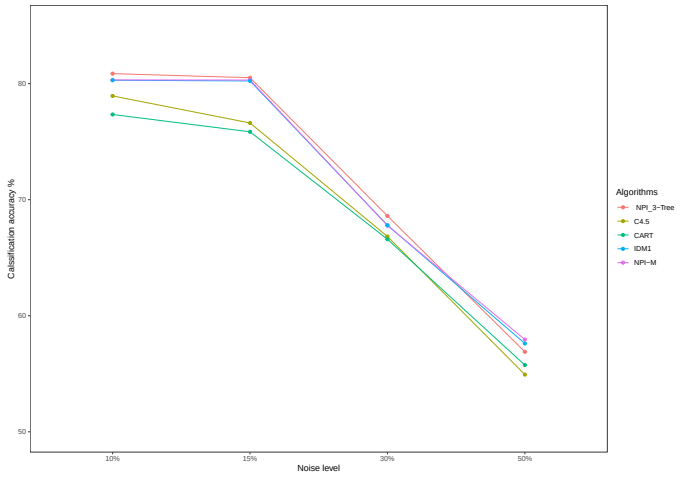
Table 5.6: Accuracy results of classification algorithms, third scenario.



(a) First scenario



(b) Second scenario



(c) Third scenario

Figure 5.3: Performance of the classification algorithms for different scenarios of the noise adding process.

5.6 Concluding remarks

In this chapter, we have presented applications of the NPI_2-Tree and NPI_3-Tree algorithms to the problem of classification with noisy data. We briefly reviewed some of the literature on classification data noise. We restricted attention to class noise because this type of noise appears in most real-world data sets and it can affect the performance of classification algorithms more than attribute noise. However, it is also interesting to study and investigate the impact of attribute noise on the performance of the NPI_2-Tree and NPI_3-Tree algorithms, as this type of noise appears in most real-world data sets, and it also can affect classification performance, but we leave this topic as future research. Also, it may be of interest to investigate the impact of adding noise to both attribute variables and class variables at the same time. Such studies and investigations might give more insight into the performance of these algorithms on noisy data sets.

We have carried out an experimental analysis on different data sets and different levels of random noise in order to evaluate the performance of the NPI_2-Tree and NPI_3-Tree algorithms with noisy data. We have also compared their results with the C4.5, CART, NPI-M and IDM1 algorithms. All classification algorithms have been measured and compared using the classification accuracy, i.e. the ratio of correctly classified observations to the total number of observations. The results obtained from these experiments have shown that the NPI_2-Tree and NPI_3-Tree algorithms perform well and provide some robustness to data sets with class noise. Their performance was slightly better than the other classification algorithms for most levels of random noise added to the class variable. Furthermore, we examined the performance of the NPI_3-Tree algorithm with a different method to add noise to the three ordered classes, and we also compared its performance with the other classification algorithms. We considered three different scenarios giving three different probabilities for adding the noise to a neighbouring class. The results also have shown that the NPI_3-Tree algorithm performed better than the other classification algorithms considered, for most scenarios and for most different noise levels.

In this chapter, we have provided an insight into the performance of the NPI_2-Tree and NPI_3-Tree algorithms with noisy data sets, however, some other interesting topics still need to be studied. In this chapter we have examined the performance of the NPI_2-Tree and NPI_3-Tree algorithms on noisy data sets using the classical split criterion, the IGR split criterion, but it will be also interesting to extend this research to consider different split criteria e.g. imprecise split criteria and compare the results with the classical split criterion. In this chapter we have added noise levels to training data sets, but it may also be possible to consider other kinds of adding noise to the data sets. It may be interesting to study the performance of classification algorithms by adding noise to both training and testing data sets. Another possible future research is to investigate the performance of the NPI_2-Tree and NPI_3-Tree algorithms with more data sets and more levels of noise. Such investigation could give more conclusions about the performance of these algorithms with noisy data. Focusing more on the characteristics of the data sets on which application of the NPI_2-Tree and NPI_3-Tree algorithms perform well, or perform poorly, could enable us to determine which characteristics of data sets are appropriate for these classification algorithms.

Chapter 6

Conclusions

In this thesis, we introduced a new method to build classification trees based on Nonparametric Predictive Inference (NPI). We built classification trees using the NPI approach for selecting optimal thresholds for data sets that involve continuous-valued attributes. This approach selects the optimal threshold values using predictive inference that considers specific numbers of future observations and the target proportions. As a first step, we presented our method for binary classification trees, where the attribute variables are continuous and class variable is binary. A new classification algorithm, which we called the NPI_2-Tree algorithm, was presented for building binary classification trees. We then extended our method to classification trees with three classes, where the attribute variables are continuous and a class variable has three ordered classes. A new classification algorithm for building classification trees with three classes was presented, which we called the NPI_3-Tree algorithm. To build classification trees based on the NPI_2-Tree and NPI_3-Tree algorithms, which are based on NPI approach for selecting the optimal thresholds, we introduced a new procedure for selecting the target proportions by choosing that to maximise classification performance on the testing datasets. We used a two-level 5-fold cross-validation procedure to define these values and validate their performance in classification trees.

We carried out an experimental analysis on several data sets to evaluate the performance of the NPI_2-Tree and NPI_3-Tree classification algorithms and compare

their performance with other classification algorithms from the literature. The classification accuracy, in-sample accuracy and tree size have been used to measure the performance of all the classification algorithms. The results of the experimental analysis have indicated that the NPI_2-Tree and NPI_3-Tree classification methods perform well, slightly better than the other classification algorithms. We presented applications of the use of our classification algorithms for cases when there is noise in the data set. The performances of these algorithms in cases of noisy data were examined using different levels of noise added to the class variable, and the results were compared with the other classification algorithms. The results have shown that our classification algorithms perform well in the case of noise and tend to be quite robust for most noise levels compared to the others.

The work presented in this thesis provides many possible topics for future research. As a first step to build classification trees based on our methods, we started with two classes; we then extended our method to build classification trees with three classes. Now, it would be of interest to extend this method further to involve more than three classes. This can be achieved by first developing the NPI method for selecting the optimal threshold to include more than three classes. In this work, we have restricted attention to using only continuous-valued attributes, it is of interest to investigate the performance of the NPI_2-Tree and the NPI_3-Tree algorithms on data sets that include categorical attributes. Future research may consider different classification methods; for example, it would be interesting to explore the use of the NPI_2-Tree and NPI_3-Tree algorithms with random forests.

One important topic for future work is to develop the NPI_2-Tree and the NPI_3-Tree algorithms with consideration for the misclassification cost. In many practical applications, classification aims to minimize misclassification costs instead of maximising the total classification accuracy. In this thesis, we have chosen the values of target proportions that maximise the total classification accuracy. However, it would be useful to develop the process of choosing these target proportions while also considering the misclassification cost. In our work in Chapter 5, we restricted

attention to studying the effect of class noise on the performance of the NPI_2-Tree and NPI_3-Tree algorithms; however, it would be interesting to study and investigate the effect of attribute noise on these algorithms, as this type of noise also appears in most real-world data sets and can affect classification performance. Such studies and investigations might provide more insight into the performance of these algorithms on noisy data sets.

Another possible future research is to investigate the performance of the NPI_2-Tree and NPI_3-Tree algorithms on a larger number of data sets with more detailed analysis. Focusing more on the characteristics of the data sets on which these classification algorithms perform well, or perform poorly, could enable us to determine which data set characteristics are appropriate for them. It will be interesting to consider other evaluation metrics, such as nonparametric tests [40], to assess the performance of our classification algorithms and compare them with other classification algorithms. Finally, developing the NPI_2-Tree and NPI_3-Tree algorithms with imprecise classification would be interesting. In imprecise classification, trees may sometimes return a set of classes in their leaves rather than a single class.

Appendix A

Optimisation technique using Genetic Algorithm (GA)

In this appendix, we explain how the optimisation method Genetic Algorithm (GA) works to find the optimal values for the target proportions. We first explain this method for the two classes, as used in Chapter 3, and we then explain it for the three classes, as used in Chapter 4. For more details about this method, we refer to [47, 48].

A.1 The GA for two classes

Consider the NPI lower method for selecting the optimal threshold for two classes, as given in Section 3.2 (Equation (3.1)). In order to formulate the main optimization problem, we need to rewrite this equation as follows

$$f(a,b) = \underline{P}\left(L_t^1 \geq am_1, L_t^2 \geq bm_2\right) = \underline{P}\left(L_t^1 \geq am_1\right) \times \underline{P}\left(L_t^2 \geq bm_2\right) \qquad \text{(A.1.1)}$$

where $\underline{P}(L_t^1 \geq am_1)$ and $\underline{P}(L_t^2 \geq bm_2)$ are given in Equations (3.3) and (3.6), respectively. Where a and b are values in (0,1], i.e.

$$(a,b) \in (0,1]^2 \qquad \text{(A.1.2)}$$

Using Equation (A.1.1) and condition (A.1.2), we can formulate the main optimisation problem as follows

$$F(a,b) = f(a,b) \rightarrow Max\ Accuracy \qquad \text{(A.1.3)}$$

where *Max Accuracy* refers to the maximum classification accuracy on the testing set. To find the solution of problem (A.1.2) and (A.1.3), we use a Genetic Algorithm (GA) [47, 48]. The GA is a a search-based optimization technique based on the rules of genetics and natural selection. For construction of the GA, we need to define two main functions: crossover function between points from $(0,1]^2$ and mutation function form point from $(0,1]^2$. In the GA, crossover, also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring, while mutation is the process of altering the value of gene i.e to replace the value 1 with 0 and vice-versa. In the following we explain how these two main functions are calculated.

For crossover function, consider two points $(a_1, b_1) \in (0,1]^2$ and $(a_2, b_2) \in (0,1]^2$, with values of the optimisation function $F(a_1, b_1)$ and $F(a_2, b_2)$. Then crossover between points (a_1, b_1) and (a_2, b_2) are defined by point (a_{cros}, b_{cros}):

$$\begin{cases} a_{cros} = \frac{1}{F(a_1,b_1)+F(a_2,b_2)}(F(a_1,b_1)a_1 + F(a_2,b_2)a_2) \\ b_{cros} = \frac{1}{F(a_1,b_1)+F(a_2,b_2)}(F(a_1,b_1)b_1 + F(a_2,b_2)b_2). \end{cases} \tag{A.1.4}$$

For mutation function, consider point $(a_1, b_1) \in (0,1]^2$. The mutation function of the point $(a_1, b_1) \in (0,1]$ to point(a_{mut}, b_{mut}) :

$$(a_{mut}, b_{mut}) = (a_1, b_1) + r * (u_1, u_2), \tag{A.1.5}$$

where u_i are independent random variables with uniform distribution in $[-1, 1]$, and r defined as

$$r = min\{a_1, 1 - a_1, b_1, 1 - b_1\}. \tag{A.1.6}$$

After explaining the two main functions of the GA, we can now define the main algorithm, the GA, for solving the problem (A.1.2) and (A.1.3):

1. Choose number of points N , number of mutations in each iteration $M < 0.9N$, number of iterations $Iter$;
2. $iter = 1$;
3. Define initials points (a_i, b_i), $i = 1, \ldots, N$, which are in $(0, 1]^2$;
4. Calculate $F(a_i, b_i)$, $i = 1, \ldots, N$;
5. Ordering points (a_i, b_i), $i = 1, \ldots, N$ by the values of $F(a_i, b_i)$;
6. Choose 10% of the best points (a_i, b_i), $i = 1, \ldots, N$ with the highest values of $F(a_i, b_i)$;
7. Create crossover points $(a_i^{new}, b_i^{new}), i = 1, \ldots, M$ as random crossover of points from step 5 with 10% points from 90% of the worst points;
8. Mutate all points, which does not used in step 6;
9. $iter = iter + 1$;
10. Return to 2 until $iter \leq Iter$.

A.2 The GA for three classes

The GA for the three classes is very similar as for the two classes. Consider the NPI lower method for selecting the optimal threshold for three classes, as given in Section 4.2 (Equation (4.1)). we rewrite this equation as follows

$$\begin{aligned} f(a, b, c) =& \underline{P}(L^1_{t_1} \geq am_1, L^2_{(t_1,t_2)} \geq bm_2, L^3_{t_2} \geq cm_3) = \\ & \underline{P}(L^1_{t_1} \geq am_1) \times \underline{P}(L^2_{(t_1,t_2)} \geq bm_2) \times \underline{P}(L^3_{t_2} \geq cm_3) \end{aligned} \tag{A.2.7}$$

where $P(L^1_{t_1} \geq am_1)$, $P(L^2_{(t_1,t_2)} \geq bm_2)$ and $P(L^3_{t_2} \geq cm_3)$ are given in Equations (4.3), (4.5) and (4.7), respectively, and a, b, c are any values in (0,1]. So, the main optimisation problem is formulated as follows

$$F(a, b, c) = f(a, b, c) \rightarrow Max\ Accuracy \tag{A.2.8}$$

where $Max\ Accuracy$ is the maximum classification accuracy on the testing set. To use the GA, we also need to define the two main functions, crossover function and mutation function. Consider two points (a_1, b_1, c_1) and (a_2, b_2, c_2), each value must be in $(0, 1]$, with values of the optimization function $F(a_1, b_1, c_1)$ and

$F(a_2, b_2, c_2)$. Then crossover between points (a_1, b_1, c_1) and (a_2, b_2, c_2) are defined by point $(a_{cros}, b_{cros}, c_{cros})$:

$$\begin{cases} a_{cros} = \frac{1}{F(a_1,b_1,c_1)+F(a_2,b_2,c_2)}(F(a_1, b_1, c_1)a_1 + F(a_2, b_2, c_2)a_2) \\ b_{cros} = \frac{1}{F(a_1,b_1,c_1)+F(a_2,b_2,c_2)}(F(a_1, b_1, c_1)b_1 + F(a_2, b_2, c_2)b_2) \\ c_{cros} = \frac{1}{F(a_1,b_1,c_1)+F(a_2,b_2,c_2)}(F(a_1, b_1, c_1)c_1 + F(a_2, b_2, c_2)c_2) \end{cases} \tag{A.2.9}$$

For mutation function, consider point (a_1, b_1, c_1), where each value must be in $(0, 1]$. The mutation function of the point (a_1, b_1, c_1) to point $(a_{mut}, b_{mut}, c_{mut})$:

$$(a_{mut}, b_{mut}, c_{mut}) = (a_1, b_1, c_1) + r * (u_1, u_2, u_3), \tag{A.2.10}$$

where u_i are independent random variables with uniform distribution in $[-1, 1]$, and r defined as

$$r = min\{a_1, 1 - a_1, b_1, 1 - b_1, c_1, 1 - c_1\}. \tag{A.2.11}$$

After defining main function of the GA, we can now use the main algorithm, the GA, to solve the problem (A.2.8):

1. Choose number of points N , number of mutations in each iteration $M < 0.9N$, number of iterations $Iter$;
2. $iter = 1$;
3. Define initials points (a_i, b_i, c_i), $i = 1, \ldots, N$, which are in $(0, 1]$;
4. Calculate $F(a_i, b_i, c_i)$, $i = 1, \ldots, N$;
5. Ordering points (a_i, b_i, c_i), $i = 1, \ldots, N$ by the values of $F(a_i, b_i, c_i)$;
6. Choose 10% of the best points (a_i, b_i, c_i), $i = 1, \ldots, N$ with the highest values of $F(a_i, b_i, c_i)$;
7. Create crossover points $(a_i^{new}, b_i^{new}, c_i^{new}), i = 1, \ldots, M$ as random crossover of points from step 5 with 10% points from 90% of the worst points;
8. Mutate all points, which does not used in step 6;
9. $iter = iter + 1$;

10. Return to 2 until $iter \leq Iter$.

The following small examples illustrate the optimal values of a, b and c that maximise the classification accuracy using the GA optimisation method.

Example A.2.1 Assume that we have a data set of 33 observations belonging to three classes, C_1, C_2 and C_3. Suppose that the training data set with $n_1 = n_2 = n_3 = 7$, consisting of the data {1,2,3,4,5,6,7} for C_1, {8,9,10,11,12,13,14} for C_2 and {15,16,17,18,19,20,21} for C_3, and the testing data set with $m_1 = m_2 = m_3 = 4$, consisting of the data {4,5,6,7} for C_1, {10,11,12,13} for C_2 and {17,18,19,20} for C_3. Using the GA optimization technique, we found that the values that maximise the classification accuracy are $a = b = c = 1$, leading to the optimal thresholds $t_1 = 7$ and $t_2 = 14$, and the classification accuracy 100%.

Example A.2.2 Consider the same data set described in Example 4.3.1, where $n_1 = 4, n_2 = 5$, $n_3 = 4$, $m_1 = m_2 = 3$ and $m_3 = 2$. Using the GA optimization technique, we found the values of a, b and c that maximise the classification accuracy are $a = 0.66, b = 0.33$ and $c = 0.50$, leading to the optimal thresholds $t_1 = 5$ and $t_2 = 10$, and the total classification accuracy is 100%.

Bibliography

[1] Abellán, J. (2006). Uncertainty measures on probability intervals from the imprecise Dirichlet model. *International Journal of General Systems*, 35, 509–528.

[2] Abellán, J. (2013). An application of non-parametric predictive inference on multi-class classification high-level-noise problems. *Expert Systems with Applications*, 40, 4585–4592.

[3] Abellán, J. and Castellano, J. (2017). A comparative study on base classifiers in ensemble methods for credit scoring. *Expert Systems with Applications*, 73, 1–10.

[4] Abellán, J. and Masegosa, A. (2010). Bagging decision trees on data sets with classification noise. In *International Symposium on Foundations of Information and Knowledge Systems*, pp. 248–265. Springer.

[5] Abellán, J. and Masegosa, A. (2012). Bagging schemes on the presence of class noise in classification. *Expert Systems with Applications*, 39, 6827–6837.

[6] Abellán, J. and Moral, S. (2003). Building classification trees using the total uncertainty criterion. *International Journal of Intelligent Systems*, 18, 1215–1225.

[7] Abellán, J., Baker, R. and Coolen, F.P.A. (2011). Maximising entropy on the nonparametric predictive inference model for multinomial data. *European Journal of Operational Research*, 212, 112–122.

[8] Abellán, J., Baker, R., Coolen, F.P.A., Crossman, R. and Masegosa, R. (2014). Classification with decision trees from a nonparametric predictive inference perspective. *Computational Statistics and Data Analysis*, 71, 789–802.

[9] Aboalkhair, A. and Coolen, F.P.A., and MacPhee, I. (2014). Nonparametric predictive inference for reliability of a k-out-of-m: G system with multiple component types. *Reliability Engineering & System Safety*, 131, 298–304.

[10] Ahmad, A. (2009). *Data transformation for decision tree ensembles*. Ph.D. thesis, Manchester University.

[11] Alabdulhadi, M. (2018). *Nonparametric predictive inference for diagnostic test thresholds*. Ph.D. thesis, Durham University.

[12] Alharbi, A.A. (2022). *Direct nonparametric predictive inference classification trees*. Ph.D. thesis, Durham University.

[13] Alqifari, H. (2017). *Nonparametric predictive inference for future order statistics*. Ph.D. thesis, Durham University.

[14] Amri, N. (2009). *Classification techniques for noisy and imbalanced data*. Ph.D. thesis, Florida Atlantic University.

[15] Attwood, K., Tian, L. and Xiong, C. (2014). Diagnostic thresholds with three ordinal groups. *Journal of Biopharmaceutical Statistics*, 24, 608–633.

[16] Augustin, T. and Coolen, F.P.A. (2004). Nonparametric predictive inference and interval probability. *Journal of Statistical Planning and Inference*, 124, 251–272.

[17] Augustin, T., Coolen, F.P.A., De Cooman, G. and Troffaes, M. (editors) (2014). *Introduction to Imprecise Probabilities*. Wiley, Chichester.

[18] Baker, R. (2010). *Multinomial nonparametric predictive inference: selection, classification and subcategory data*. Ph.D. thesis, Durham University.

[19] Battineni, G., Sagaro, G., Nalini, C., Amenta, F. and Tayebati, S. (2019). Comparative machine-learning approach: a follow-up study on type 2 diabetes predictions by cross-validation methods. *Machines*, 7, 74.

[20] Bernard, J. (2005). An introduction to the imprecise Dirichlet model for multinomial data. *International Journal of Approximate Reasoning*, 39, 123–150.

[21] Berry, M. and Linoff, G. (2000). *Mastering Data Mining: The Art and Science of Customer Relationship Management*. Wiley, New York.

[22] Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106, 1039–1082.

[23] Boole, G. (1854). *An Investigation of the Laws of Thought: on which are founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberly, London.

[24] Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.

[25] Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont.

[26] Coffin, M. and Sukhatme, S. (1997). Receiver operating characteristic studies and measurement errors. *Biometrics*, 53, 823–837.

[27] Coolen, F.P.A. (1998). Low structure imprecise predictive inference for Bayes' problem. *Statistics and Probability Letters*, 36, 349–357.

[28] Coolen, F.P.A. (2006). On nonparametric predictive inference and objective Bayesianism. *Journal of Logic, Language and Information*, 15, 21–47.

[29] Coolen, F.P.A. (2011). Nonparametric predictive inference. In *International Encyclopedia of Statistical Sciences*, pp. 968–970. Springer, Berlin.

[30] Coolen, F.P.A. and Augustin, T. (2005). Learning from multinomial data: a nonparametric predictive alternative to the Imprecise Dirichlet Model. *In the Fourth International Symposium on Imprecise Probabilities: Theories and Applications*, 5, 125–134.

[31] Coolen, F.P.A. and Augustin, T. (2009). A nonparametric predictive alternative to the Imprecise Dirichlet Model: the case of a known number of categories. *International Journal of Approximate Reasoning*, 50, 217–230.

[32] Coolen, F.P.A. and Maturi, T. (2010). Nonparametric predictive inference for order statistics of future observations. *In: Combining Soft Computing and Statistical Methods in Data Analysis*, pp. 97–104.

[33] Coolen, F.P.A. and Yan, K. (2003). Nonparametric predictive comparison of two groups of lifetime data. In *the Third International Symposium on Imprecise Probabilities and Their Applications*, pp. 148–161.

[34] Coolen, F.P.A. and Yan, K. (2004). Nonparametric predictive inference with right-censored data. *Journal of Statistical Planning and Inference*, 126, 25–54.

[35] Coolen, F.P.A., Coolen-Maturi, T. and Alqifari, H. (2018). Nonparametric predictive inference for future order statistics. *Communications in Statistics-Theory and Methods*, 47, 2527–2548.

[36] Coolen, F.P.A., Coolen-Schrijner, P. and Yan, K. (2002). Nonparametric predictive inference in reliability. *Reliability Engineering & System Safety*, 78, 185–193.

[37] Coolen-Maturi, T., Coolen, F.P.A. and Alabdulhadi, M. (2020). Nonparametric predictive inference for diagnostic test thresholds. *Communications in Statistics-Theory and Methods*, 49, 697–725.

[38] Coolen-Schrijner, P., Coolen, F.P.A., and MacPhee, I. (2008). Nonparametric predictive inference for system reliability with redundancy allocation. *Journal of Risk and Reliability*, 222, 463–476.

[39] De Finetti, B. (1974). *Theory of Probability.* Wiley, London.

[40] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7, 1–30.

[41] Dua, D. and Graff, C. (2019). UCI machine learning repository. *University of California, Irvine, School of Information and Computer Science.* `http://archive.ics.uci.edu/ml`.

[42] Elkhafifi, F. and Coolen, F.P.A. (2012). Nonparametric predictive inference for accuracy of ordinal diagnostic tests. *Journal of Statistical Theory and Practice*, 6, 681–697.

[43] Fayyad, U. and Irani, K. (1992). On the handling in decision tree of continuous-valued attributes generation. *Machine Learning*, 8, 87–102.

[44] Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learing. *In: Proceeding of the 13th International Joint Conference on Artificial Inteligence*, pp. 1022–1027.

[45] Fink, P. (2018). *imptree: Classification Trees with Imprecise Probabilities.* R package version 0.5.1.

[46] Fluss, R., Faraggi, D. and Reiser, B. (2005). Estimation of the Youden index and its associated cut-off point. *Biometrical Journal*, 47, 458–472.

[47] Genlin, J. (2004). Survey on genetic algorithm. *Computer Applications and Software*, 2, 69–73.

[48] Haldurai, L., Madhubala, T. and Rajalakshmi, R. (2016). A study on genetic algorithm and its applications. *International Journal of Computer Sciences and Engineering*, 4, 139.

[49] Hill, M. (1968). Posterior distribution of percentiles: Bayes' theorem for sampling from a population. *Journal of the American Statistical Association*, 63, 677–691.

[50] Hill, M. (1988). De Finetti's theorem, induction, and $A_{(n)}$ or Bayesian nonparametric predictive inference (with discussion). *Bayesian Statistics*, 3, 211–241.

[51] Hornik, K. and Buchta, C. and Zeileis, A. (2009). Open-source machine learning: R meets Weka. *Computational Statistics*, 24, 225–232.

[52] Idri, A., Kadi, I. and Benjelloun, H. (2015). Heart disease diagnosis using C4.5 algorithm - A Case Study. In *Proceedings of the International Conference on Health Informatics*, pp. 397–404. SciTePress, Lisbon (Portugal).

[53] James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer, New York.

[54] Khozeimeh, F., Alizadehsani, R., Roshanzamir, M., Khosravi, A., Layegh, P. and Nahavandi, S. (2017). An expert system for selecting wart treatment method. *Computers in Biology and Medicine*, 81, 167–175.

[55] Khozeimeh, F., Jabbari, F., Mahboubi, Y., Jafari, M., Tehranian, S., Alizadehsani, R., and Layegh, P. (2017). Intralesional immunotherapy compared to cryotherapy in the treatment of warts. *International Journal of Dermatology*, 56, 474–478.

[56] Kohavi, R. and Sahami, M. (1996). Error-based and entropy-based discretization of continuous features. *Proceedings of the Second International Cnference on Knowledge Discovery and Data Mining*, pp. 114–119.

[57] Kubat, M. (2017). *An Introduction to Machine Learning*. Springer, Switzerland.

[58] Lee, C. (2007). A Hellinger-based discretization method for numeric attributes in classification learning. *Knowledge-Based Systems*, 20, 419–425.

[59] Maletic, J. and Marcus, A. (2000). Data cleansing: beyond integrity analysis. *In International Conference on Information Quality*, pp. 200–209.

[60] Mantas, C., Abellán, J. and Castellano, J. (2016). Analysis of Credal-C4. 5 for classification in noisy domains. *Expert Systems with Applications*, 61, 314–326.

[61] Mantas, C. and Abellán, J. (2014). Credal-C4. 5: Decision tree based on imprecise probabilities to classify noisy data. *Expert Systems with Applications*, 41, 4625–4637.

[62] Mašetic, Z. and Subasi, A. (2013). Detection of congestive heart failures using C4. 5 decision tree. *Southeast Europe Journal of Soft Computing*, 2, 74–77.

[63] Moral, S., Mantas, C., Castellano, J. and Abellán, J. (2020). Imprecise classification with nonparametric predictive inference. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 53–66. Springer.

[64] Murthy, S. and Salzberg, S. (1995). Decision Tree Induction: How Effective Is the Greedy Heuristic? *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 222–227.

[65] Piatti, A., Zaffalon, M. and Trojani, F. (2005). Limits of Learning from Imperfect Observations under Prior Ignorance: the Case of the Imprecise Dirichlet Model. In *International Symposium on Imprecise Probability: Theories and Applications*, volume 5, pp. 276–286.

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

[67] Quinlan, J. (1993). C4.5: Program for machine learning. *Morgan Kaufmann*.

[68] Quinlan, J. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4, 77–90.

[69] R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[70] Rabbi, M., Hasan, S., Champa, A., AsifZaman, M. and Hasan, Md. (2020). Prediction of liver disorders using machine learning algorithms: a comparative study. In *2020 2nd International Conference on Advanced Information and Communication Technology*, pp. 111–116. IEEE, Dhaka (Bangladesh).

[71] Rokach, L. and Maimon, O. (2008). *Data Mining with Decision Trees: Theory and Applications*. World Scientific.

[72] Sáez, J., Galar, M., Luengo, J. and Herrera, F. (2013). Tackling the problem of classification with noisy data using multiple classifier systems: analysis of the performance and robustness. *Information Sciences*, 247, 1–20.

[73] Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27, 379–423.

[74] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B*, 36, 111–133.

[75] Therneau, T., Atkinson, B. and Ripley, B. (2015). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1.16.

[76] Unal, I. (2017). Defining an optimal cut-point value in ROC analysis: an alternative approach. *Computational and Mathematical Methods in Medicine*, 2017, 3762651–3762651.

[77] Walley, P. (1996). Inferences from multinomial data: learning about a bag of marbles. *Journal of the Royal Statistical Society: Series B*, 58, 3–34.

[78] Weichselberger, K. (2000). The theory of interval-probability as a unifying concept for uncertainty. *International Journal of Approximate Reasoning*, 24, 149–170.

[79] Witten, I. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco.

[80] Zhang, X., Wu, J., Lu, T. and Jiang, Y. (2007). A discretization algorithm based on Gini criterion. *International Conference on Machine Learning and Cybernetics*, 5, 2557–2561.

[81] Zhu, X. and Wu, X. (2004). Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22, 177–210.