ป่าสุ่มด้วยบูทสแทรปรูปแบบควอไทล์สำหรับปัญหาคลาสไม่ดุล

นายวรวิทย์ จิตภักดีบดินทร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2565

RANDOM FOREST USING QUARTILE-PATTERN BOOTSTRAPPING FOR CLASS IMBALANCED PROBLEMS

Mr. Worawit Jitpakdeebodin

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Applied Mathematics and

Computational Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2022

Thesis Title	RANDOM FOREST USING QUARTILE-PATTERN BOOTSTRAPPING FOR CLASS IMBALANCED PROBLEMS
By	Mr. Worawit Jitpakdeebodin
Field of Study	Applied Mathematics and Computational Science
Thesis Advisor	Associate Professor Krung Sinapiromsaran, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

[illegible]
.... Dean of the Faculty of Science
(Professor Pranut Potiyaraj, Ph.D.)

THESIS COMMITTEE

K. Plaimas
.... Chairman
(Associate Professor Kitiporn Plaimas, Ph.D.)

[illegible]
.... Thesis Advisor
(Associate Professor Krung Sinapiromsaran, Ph.D.)

[illegible]
.... Examiner
(Thap Panitanarak, Ph.D.)

[illegible]
.... External Examiner
(Associate Professor Chumphol Bunkhumpornpat, Ph.D.)

วรวิทย์ จิตภักดีบดินทร์ : ป่าสุ่มด้วยบูทสแทรปรูปแบบควอไทล์สำหรับปัญหาคลาสไม่ดุล. (RANDOM FOREST USING QUARTILE-PATTERN BOOTSTRAPPING FOR CLASS IMBALANCED PROBLEMS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร. กรุง สินอภิรมย์สราญ, 48 หน้า.

ในปัจจุบัน การจำแนกประเภทในการเรียนรู้ของเครื่องเป็นเครื่องมือที่สำคัญสำหรับการดึงข้อมูลและการวิเคราะห์ข้อมูลโลกจริง อย่างไรก็ตาม ปัญหาที่สำคัญในการจำแนกประเภทคือปัญหาของความไม่ดุลของคลาส ซึ่งมีผลกระทบต่อประสิทธิภาพของตัวจำแนกประเภทอย่างมีนัยสำคัญ ในปี 2019 มีการนำเสนอวิธีการใหม่สำหรับการสร้างต้นไม้ตัดสินใจเพื่อแก้ปัญหานี้ — ไมนอริตี้คอนเดนเซชันเอ็นโทรปี (MCE) ซึ่งสามารถจัดการกับชุดข้อมูลที่ไม่ดุลได้อย่างมีประสิทธิภาพ ต่อมาในปี 2021 มีการนำเสนอตัววัดปัจจัยความผิดปกติ เรียกว่าปัจจัยความผิดปกติแมสเรโชแวเรียนซ์ (MOF) ที่สามารถจัดลำดับตัวอย่างตามความหนาแน่นของข้อมูล วิทยานิพนธ์นี้นำเสนอขั้นตอนวิธีป่าสุ่มที่ใช้รูปแบบบูตสแทรปที่รวม MOF และ MCE เพื่อสร้างป่าสุ่มที่สามารถจัดการกับชุดข้อมูลสองคลาสที่ไม่ดุล ผลการทดลองบนชุดข้อมูลสังเคราะห์และชุดข้อมูลจริงแสดงให้เห็นว่าขั้นตอนวิธีที่นำเสนอมีประสิทธิภาพมากกว่าขั้นตอนวิธีที่มีอยู่ในด้านพรีซีชัน รีคอล ตัววัดเอฟ และค่าเฉลี่ยเรขาคณิต แสดงถึงความสามารถในการจัดการกับชุดข้อมูลที่ไม่ดุลและประสิทธิภาพที่ดีกว่าในการจำแนกประเภท

ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	ลายมือชื่อนิสิต	[illegible]
		ลายมือชื่อ อ.ที่ปรึกษาหลัก	[illegible]
สาขาวิชา	คณิตศาสตร์ประยุกต์และวิทยาการคณนา		
ปีการศึกษา	2565		

6470133423 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS : RECURSIVE PARTITIONING ALGORITHM / DECISION TREE / RANDOM FOREST / CLASSIFICATION PROBLEM / CLASS IMBALANCED PROBLEM

WORAWIT JITPAKDEEBODIN : RANDOM FOREST USING QUARTILE-PATTERN BOOTSTRAPPING FOR CLASS IMBALANCED PROBLEMS. ADVISOR : ASSOC. PROF. KRUNG SINAPIROMSARAN, Ph.D., 48 pp.

Nowadays, classification in machine learning serves as a valuable tool for extracting and analyzing real-world datasets. However, an important issue in classification is the problem of class imbalance, which significantly impacts the performance of classifiers. In 2019, a novel approach for a decision tree induction was introduced to address this problem—the Minority Condensation Entropy (MCE) measure that can effectively handle imbalanced datasets. Subsequently, in 2021, a new outlier factor called the Mass-ratio-variance Outlier Factor (MOF) was presented that can rank instances based on the dataset density. This thesis proposes a random forest algorithm using quartile-pattern bootstrapping that incorporates MOF and MCE building a random forest capable of handling binary class imbalanced datasets. The experimental results on synthesized datasets and real-world datasets indicated that the proposed algorithm outperforms other existing algorithms in terms of Precision, Recall, F-measure, and geometric mean, showing its effectiveness in handling imbalanced datasets and improving classification accuracy.

Department	: Mathematics and Computer Science	Student's Signature	[illegible]
Field of Study	: Applied Mathematics and Computational Science	Advisor's Signature	[illegible]
Academic Year	: 2022		

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my dissertation advisor, Associate Professor Krung Sinapiromsaran, Ph.D. for the continuous supporting my studies in the master's degree. His encouragement, motivation and guidance helped me during the time for writing researches and this dissertation until it was accomplished. I further would like to thank all of my dissertation committees: Associate Professor Kitiporn Plaimas, Ph.D., Professor Thap Panitanarak, Ph.D., and Associate Professor Chumphol Bunkhumpornpat, Ph.D., for their insightful comments and suggestions which motivated me to extend my research from various perspectives.

Moreover, I would like to express my special appreciation and thanks to the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University which provided me funding to present my research on international conferences and an opportunity that I received throughout my graduate studies.

Finally, I would like to thank my family for supporting me throughout writing this dissertation. Also, I wish to express my gratitude to all friends and colleagues who stayed with me and provided their encouragement, relaxation, great suggestions and supports in many ways during a hard time studying in my master's degree.

CONTENTS

CHAPTER I

INTRODUCTION

This chapter provides an introduction, research objectives, and an overview. The first section introduces machine learning, classification, imbalanced problems, and MOF. The second section explains the research objectives, and the third section provides an overview of the research.

1.1 Introduction

In the recent years, the amount of information has grown rapidly due to the widespread use of wireless communications. Only human cannot deal with the large information. Therefore, the convenient tools to deal with these data are needed and one of the popular tools is machine learning. The machine learning [3] is the field devoted to understanding and building methods that enable machine to 'learn' from the historical data and experience. In a machine learning algorithm, the sample data, known as training data, is used to build a model in order to make predictions or decisions automatically. For example, in weather forecasting [4], machine learning tries to learn to predict whether the weather of the next day is sunny, rainy, or windy by learning from the historical weather data. Machine learning algorithms are used in a wide variety of applications such as in medicine [30], fraud detection [1], and computer vision [19]. The machine learning can be divided into three categories: Supervised learning, unsupervised learning, and reinforcement learning, as shown in Figure 1.1.

In supervised learning, the training data consist of inputs and their desired outputs. The goal of this learning is to learn a general rule that maps inputs to outputs. Types of supervised-learning include classification and regression. In classification, the outputs are restricted to a limited set of values while the outputs in regression can be any numerical value. The example of classification problems are diagnostics, fraud detection, and image classification. The examples of regression problems are weather forecasting [4], market forecasting, and estimating life expectancy.

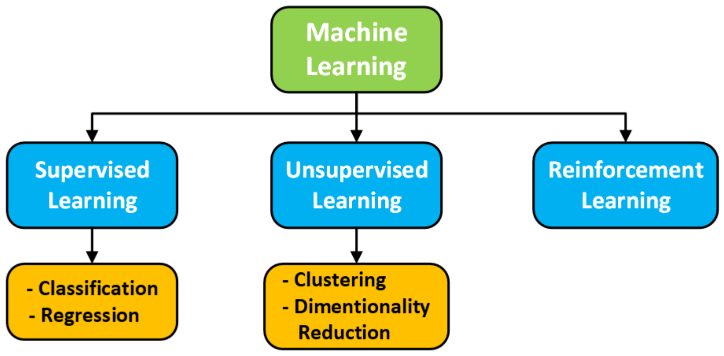


Figure 1.1: Type of machine learning

In unsupervised learning, the training data are unlabeled. This learning focused on analyze and cluster unlabeled data set. The goal of this learning is to discover hidden patterns of the data set. Unsupervised-learning models are utilized for clustering and dimensionality reduction. The clustering algorithm such as k-means clustering algorithm [17] aims to divide the data into clusters while dimensionality reduction such as principle component analysis (PCA) [18] aims to transform the data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data. Figure 1.2 shows an example of a clustering algorithm that divides the data into three clusters. The examples of clustering applications are customer segmentation, recommender system, and targeted marketing. The example of dimensionality reduction applications are structure discovery, big data visualization, and feature selection.

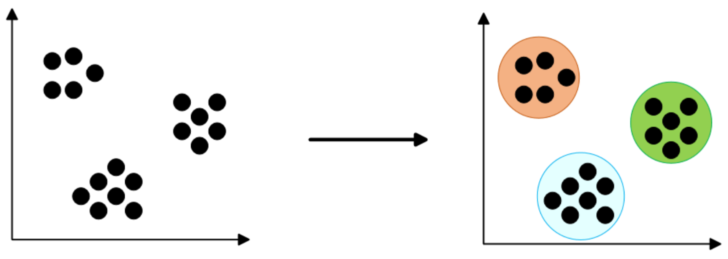


Figure 1.2: Example of clustering

The last learning is reinforcement learning. This learning is different from the previous learning since it does not need labelled input/output pairs to be presented. In general, the main components in reinforcement learning consist of agent, environment, state, and actions. The diagram of reinforcement learning is shown in Figure 1.3. The agent is the learning decision maker. It interacts with the environment that includes everything outside the agent and takes an action to modifies its state. When the agent takes an action, the environment provides a reward. This learning is applied in many fields such as game theory [6], operations research, information theory, and statistics.

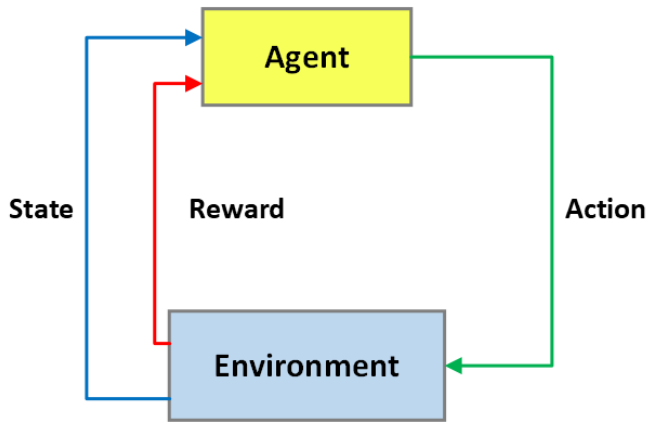


Figure 1.3: Reinforcement learning diagram

This thesis focuses on classification in supervised learning. It aims to build a classifier to classify a target class of an unknown-class observation. There are many classification models which have been proposed and widely used. For example, the Naives Bayes model [26] use Bayes's theorem to calculate the probabilties that instances belong to each class, the k-nearest neighbors classifier (KNN) [13] considers the similartity of the instances that located near the interested instance, a support vector machine (SVM) [8] aims to build a hyperplane to divide the instance space, a neural network [27] imitates the human brain's function, a decision tree [21] employ the recursive partitioning method based on properties of all attributes, and a random forest [9] is a collection of various decision trees. Examples of KNN, support vector machine, and neural network are shown in Figures 1.4, 1.5, and 1.6, respectively.

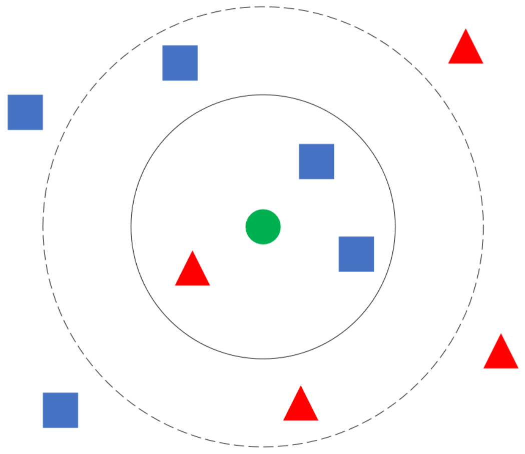


Figure 1.4: Example of kNN

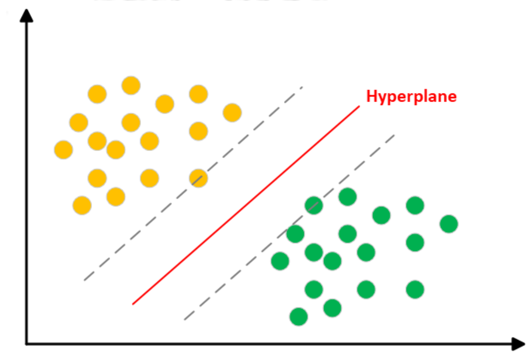


Figure 1.5: Example of a support vector machine

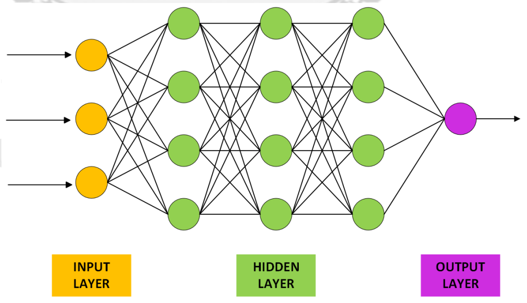


Figure 1.6: Example of a neural network

However, the efficiency of the classifier will drop when it faces the class imbalanced problem [2]. The class imbalanced problem refers to the problem of dataset having an extremely unequal class distribution. An example of an imbalanced dataset is shown in Figure 1.7. A class with a small number of instances is called a minority class and a class with a large number of instances is called a majority class. The imbalanced problem causes the misclassification for a minority instances. Usually, minority instances are more important than majority instances in most real-world applications such as detecting a small number of diabetes patients from a large number of patients [14]. Therefore, many methods are proposed to solve this problem.

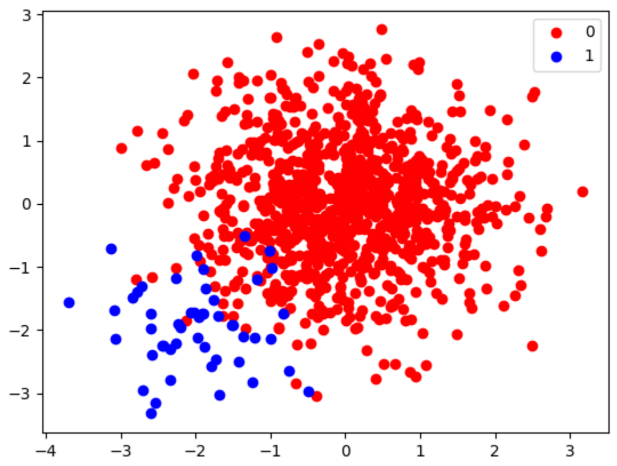


Figure 1.7: Example of an imbalanced dataset

The methods to handle class imbalanced problem can be categorized into four methodologies. The data-level methodology focuses on preprocessing the data to make it balance before using the data to build a classifier. This methodology can be divided into undersampling method and oversampling method. The concept of undersampling method is to decrease the number of majority instances by sampling the majority instances. On the contrary, the oversampling aims to increase the number of minority instances by synthesizing the minority instances. Figure 1.8 shows an example of undersampling and oversampling methods, with SMOTE [24] being an example of the oversampling method. The algorithm-level methodology such as OMCT [25] attempts to update the classification algorithms to handle imbalanced dataset automatically. The cost-sensitive methodology such as ANNs [29] and AdaCost [15] will consider the misclassification cost of minority and majority instances and treat them differently. The ensemble methodology such as AdaBoost [16] will combine all mentioned methodologies. This thesis interests to propose an improved random forest from the algorithm-level methodology since it does not change the original data distribution, and it is more adaptable to various characteristics of imbalanced datasets.

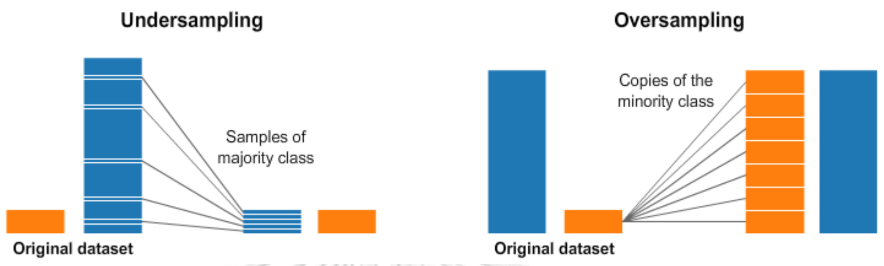


Figure 1.8: Data-level methodology

One of the popular classification models is the decision tree. A decision tree is one of the predictive models used in statistics, data mining, and machine learning. It is a tree-like structure model consisting of a set of nodes connected with branches. In order to build a decision tree, a splitting measure is calculated to select a splitting condition in each node of a tree. In 2020, Sagoolmuang and Sinapiromsaran proposed a new splitting measure called minority condensation entropy (MCE) [23] which can handle class imbalanced dataset. This method computes the entropy of the set of instances within minority range instead of all datapoints. The minority range is the range between the minimum and maximum values of minority instances that are within the inner fence of minority datasets. A decision tree built based on MCE is called the minority condensation decision tree (MCDT).

This thesis interests in random forest which is the extension of a decision tree. The random forest is one of the most popular classification model since it has high accuracy compared with other models. The algorithm to build a random forest consists of two main steps which are (1) a bootstrap on a training set and (2) attributes randomization to build decision trees. Nevertheless, when the bootstrapping is used on an imbalanced dataset, the probability that the minority instances will appear in bootstrap data is extremely low. Consequently, minority instances in some of the bootstrap data cannot maintain its original distribution. This issue has an impact on the performance of the model in a class-imbalanced problem. Therefore, this thesis plans to improve bootstrapping method in random forest algorithm to be able to maintain the distribution of instances. The idea of the improvement is to keep the outliers and the border datapoints based on their MOF scores.

In 2021, Changsakul, Boonsiri, and Sinapiromsaran [11] proposed an outlier factor algorithm called Mass-ratio-variance based Outlier Factor or MOF which is calculated from the variance of the mass-ratio distribution from the rest of data points. The higher the MOF, the more likely the data point will be an outlier. Normally, the variance of the mass-ratio of the datapoint located near the center of the distribution tends to have a smaller value than the other points; i.e., the lower the MOF score, the closer a datapoint is to the cluster center. Therefore, MOF score can be used to identify whether the datapoint is an outlier, border data point, or normal data point.

From this motivation, our research aims to improve the random forest algorithm by applying MOF and MCDT. The workflow of the algorithm is divided into two steps: (1) In a bootstrapping step, MOF of minority instances are calculated to divide the instances in four quartiles and treat them differently. The improved bootstrapping method is called quartile-pattern bootstrapping. (2) Then, the standard decision tree algorithm and MCDT algorithm are applied to generate trees in the forest.

1.2 Research Objective

The objective of this thesis is to propose an improved random forest algorithm called random forest algorithm using quartile-pattern bootstrapping (RFQP) which is able to handle class-imbalanced dataset. The algorithm is implemented and applied on synthetic datasets and real-world datasets to compare the performance with other methods using Precision, Recall, F-measure, and G-Mean as their performance measures.

1.3 Research Overview

The remainder of this research is organized as follows. In the second chapter, the background knowledge used in this thesis are described. Then, the third chapter demonstrates the detail of RFQP. In the fourth chapter, the experiments to measure the performance of RFQP and the results are presented. Finally, the fifth chapter provides the discussion and conclusion of this work.

CHAPTER II

BACKGROUND KNOWLEDGE

This chapter covers a classification process overview, the structure of a dataset that is used in the research, a class imbalanced problem, a decision tree induction and the updated decision tree induction called the minority condensation decision tree which can handle imbalanced dataset, a random forest which is an extension of a decision tree, and the mass-ratio-variance outlier factor.

2.1 Classification Process Overview

The classification process, shown in Figure 2.1, refers to the process of designing a model with the ablility to identify the class of unknown observations. The process is divided into two phases: the training phase and the testing phase. In the training phase, the classification model is built from the training dataset (or the set of labeled instances) using the classification algorithm. Then, in the testing phase, the classification model is used to predict the classes of unlabeled instances in the testing dataset with known class labels.

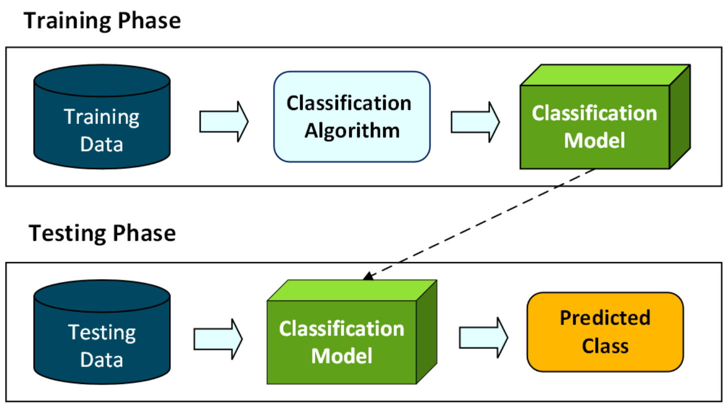
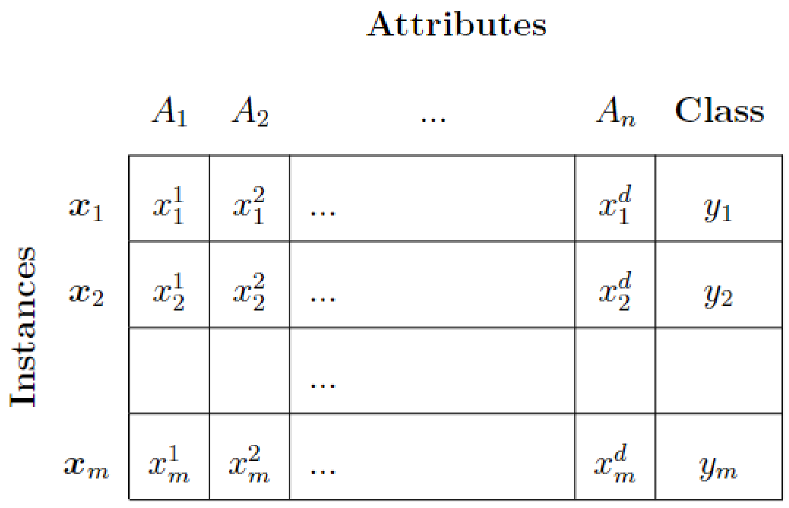


Figure 2.1: Classification process

To evaluate the performance of a classification model, it is applied to classify a class of the labeled dataset and then measure the difference between predicted class and actual class. However, the model is normally overfitted with a dataset that is used to build from the training dataset. Therefore, the set of instances used in the training process and the testing process should not be the same. This research use the holdout method which directly divides a dataset into two subsets by the randomization.

2.1.1 Training Dataset Used For Classification

To define the structure of the training dataset, the number of instances, attributes, and classes in a training dataset is denoted by m, d, and p, which are indexed by i , j, and k, respectively. Let a $= \{A_j | j = 1, 2, ..., d\}$ be a set of input attributes and y be a target attribute (class label). Define the dataset $D = \{(\boldsymbol{x}_i, y_i) | i = 1, 2, .., m\}$ where $\boldsymbol{x}_i = (x_i^1, x_i^2, ..., x_i^d)$. The x_i^l refers to the value of instance i at attribute A_l, and y_i refers to the class of the instance $\boldsymbol{x}_i$. The example of a data structure is shown in Figure 2.2



		Attributes				
		A_1	A_2	...	A_n	Class
Instances	$\boldsymbol{x}_1$	x_1^1	x_1^2	...	x_1^d	y_1
	$\boldsymbol{x}_2$	x_2^1	x_2^2	...	x_2^d	y_2
				...		
	$\boldsymbol{x}_m$	x_m^1	x_m^2	...	x_m^d	y_m

Table 2.1: A structure of data

Figure 2.2: A structure of data

2.2 Classification Model

A classification model or classifier is designed to categorize input data into distinct classes or categories. It achieves this by learning patterns and relationships from a labeled training dataset, where instances are already assigned to specific classes. Once trained, the model can predict the class of new, unseen data based on the learned patterns. This section provides information on two classifiers: the decision tree and the random forest.

2.2.1 Decision Tree

A decision tree is one of the predictive models used in classification. It is a tree-like structure model consisting of a set of nodes connected with branches. Nodes can be categorized into three types. A root node that has a path to all nodes in the decision tree is the node at the top of the tree. Leaf nodes are nodes that have no branch. Internal nodes are nodes having a parent node and connect to other nodes by branches. In the decision tree, the non-leaf node which are the root node and the internal nodes are labeled with the attribute while all branches represent the outcome from that attribute, and the leaf nodes are labeled by the class. An example of decision tree structure is shown in Figure 2.3.

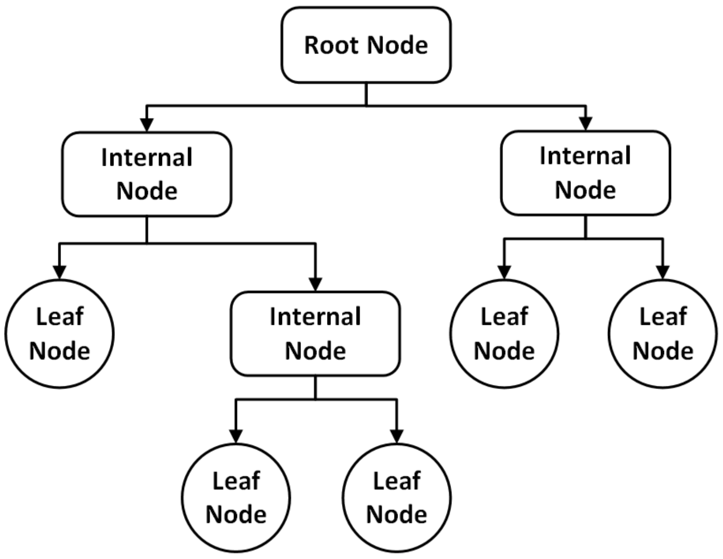


Figure 2.3: A decision tree structure

A decision tree example is shown in Figure 2.4. It was built from the data in Table 2.1. This tree predicts whether people decided to go outside or not. Each non-leaf node represents the condition on some attribute and each outcome is shown in each branch. For instance, the root node of the tree uses attribute "Outlook" to separate sunny, overcast, and rainy. For each leaf node, it represents the class, either go outside ("Yes") or not ("No").

To classify an unlabeled instance, the instance will start at the root node and moving through the internal nodes until it reaches the leaf node. In each non-leaf node, an instance will check the condition at the node and moves to the appropriate branch based on the outcome of the condition. The example of an unlabeled instance is shown in Figure 2.5. To predict the class of this instance using a decision tree in Figure 2.4, the instance will start at the root node and consider the attribute "Outlook". Since the value in attribute "Outlook" of the instance is "Rain", the instance will move to "Windy" node. After that, the instance will check the value in attribute "Windy" which is True. Therefore, the instance moves to the next node which is the leaf node. Hence, the prediction of this instance is "No".

Outlook	**Temperature**	**Humidity**	**Windy**	**Go outside**
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rain	Mid	High	False	Yes
Rain	Cool	Low	False	Yes
Rain	Cool	Low	True	No
Overcast	Cool	Low	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Low	False	Yes
Rain	Mid	Low	False	Yes
Sunny	Mid	Low	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Low	False	Yes
Rain	Mid	High	True	No

Table 2.1: Example of a decision tree training set

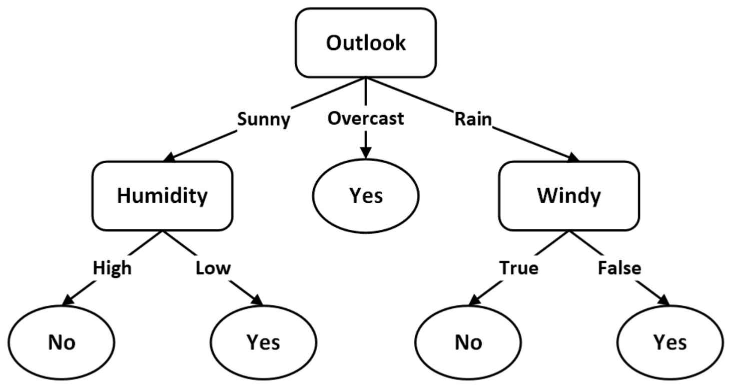


Figure 2.4: Example of a decision tree

Outlook	Humidity	Windy	Go out?
Rain	Low	True	

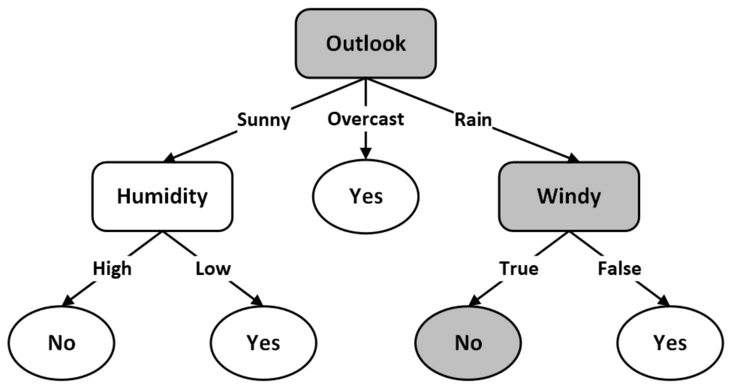


Figure 2.5: A decision tree classification

2.2.1.1 Decision Tree Induction

The task to build a decision tree is called the decision tree induction. It constructs a tree by learning from the training instances using an algorithm called the recursive partitioning algorithm which is shown in Algorithm 1. The algorithm starts with creating the root node for the entire training dataset. Then, the algorithm will check whether the dataset correspond to the stopping criteria or not. Normally, the algorithm will check that a tree reach the maximum depth and the class of all instances in the dataset are same. If the dataset corresponds to the criteria, the leaf node is created corresponding the majority class of the dataset. Otherwise, the best attribute for splitting the dataset is selected as the splitting condition and seperates the dataset into several partitions. Then, the algorithm will perform until all partitions meet stopping criteria.

In general, a single attribute is selected as the splitting condition in each non-leaf node. For each attribute, all possible values to partition the dataset called candidates are considered, which depends on the types of attributes.

- **Categorical attributes** Categorical attribute is an attribute which has a limit number of possible values. The dataset is divided into subsets based on the number of posssible values of the categorical attribute. For instance, in Figure 2.6, "Sex" which is a categorical attribute has 2 distinct values, which are "Male" and "Female".

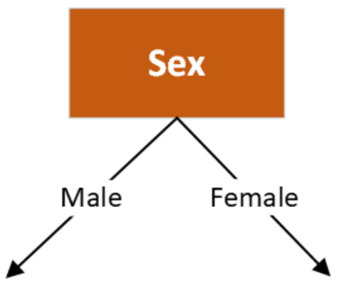


Figure 2.6: Example of the categorical attribute

- **Numerical attributes** Numerical attribute or continuous attribute is an attribute which has infinite possible values. Due to the infinite possible values, partitioning the set of instances by a numeric attribute could not use all values as the outcomes of the splitting condition. Normally, a particular value called splitting value is used to split a dataset into 2 subsets. For instance, in Figure 2.7, Age is a numerical attribute. Hence, the dataset is divided into 2 subsets. The first subset consist of instances having a value in an attribute more than the splitting value which is 20 while the value of the second subset is less than or equal to the splitting value.

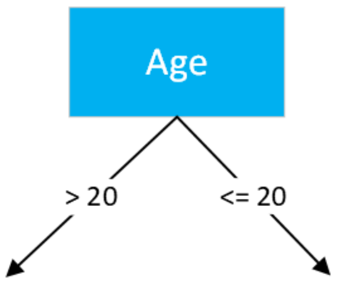


Figure 2.7: Example of the numerical attribute

Algorithm 1 RecursivePartitioning(D)

```
    Input: a dataset $D$
    Output: a decision tree
1: create the root node of the tree
2: if D correspond to the stopping criteria then
3:     return the leaf node with respect to the majority class
4: else
5:     select the splitting condition
6:     separate the dataset into $q$ partitions corresponding to the outcomes of
7:     splitting conditions: $D = D^{(1)} \cup D^{(2)} \cup ... \cup D^{(q)}$
8:     iterate for each partition RecursivePartitioning($D^{(l)}$) for $l = 1, 2, ..., q$
```

2.2.1.2 Splitting Measures

To select a splitting condition of each non-leaf node in the decision tree induction, the measure called splitting measure is applied to each attribute and the attribute that provides the optimal value of the specific splitting measure will be chosen. There are many splitting measure that are proposed and most of them are under the concept of the impurity of the dataset such as the Gini index, the Shannon's entropy [12](or SE or entropy) and the misclassification error which are defined by (2.1), (2.2) and (2.3), respectively. The $P_k(D)$ denotes the proportion of instances from class c_k that are contained in dataset D, i.e., $P_k(D) = \frac{|D_k|}{|D|}$. The highest value will occur when the number of instances in each class are same, and the lowest value, normally zero, is occur when all instances in the dataset have the same class. The comparison among the impurity measures, Entropy, Gini index, and the misclassification errors is shown in Figure 2.8.

$$Gini(D) = 1 - \sum_{k=1}^{p} (P_k(D))^2 \tag{2.1}$$

$$Entropy(D) = -\sum_{k=1}^{p} P_k(D) \log_2 P_k(D) \tag{2.2}$$

$$ClassError = 1 - \max_{k=1,2,\ldots,p} P_k(D) \tag{2.3}$$

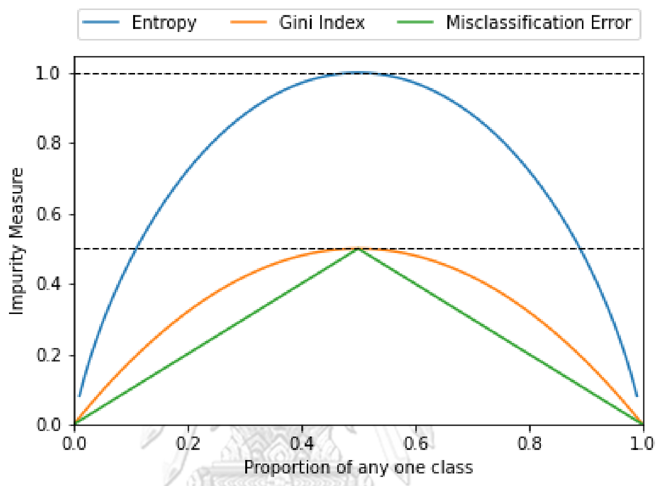


Figure 2.8: The comparison of each splitting measure for the binary-class dataset

In practice, the splitting measure that is used to compare for selecting the splitting condition is computed from the weighted average of the assigned weight values from a splitting measure calculated from each subset of instances after partitioning dataset which is defined by (2.4) where SplitMeasure represents any splitting measure and c denotes a candidate of the splitting condition that partitions the dataset D into q subsets, i.e., $D = D^{(1)} \cup D^{(2)} \cup ... \cup D^{(q)}$. If the splitting measure is impurity-based, the candidate that provides the lowest value will be chosen as the splitting condition.

$$SplitMeasure_c(D) = \sum_{l=1}^{q} \frac{|D^{(l)}|}{|D|} SplitMeasure(D^{(l)}) \tag{2.4}$$

In many well-known decision tree algorithms, the different splitting measure is applied. For example, the classification and regression tree (CART) algorithm uses the Gini index.

In 1986, the Iterative Dichotomiser 3 (ID3) algorithm, proposed by Quinlan [20], uses the information gain to determine the splitting condition. The information gain is calculated from the difference of the entropy of the entire dataset before splitting and the weighted average of the entropy of each partition after splitting, which is shown in (2.5). The candidate that offers the highest information gain is chosen to be the splitting condition. However, the use of the information gain tends to favor the attribute having many distinct values. So, the decision algorithm known as the C4.5 algorithm [22] is introduced to handle this issue. It uses the gain ratio, shown in (2.7), as the splitting measure that normalizes the information gain by dividing with the split information defined by (2.6), in which $P^{(l)}(D)$ denotes the proportion of instances in $D^{(l)}$ compared to entire dataset D, i.e., $P^{(l)}(D) = \dfrac{|D^{(l)}|}{|D|}$

$$InfoGain_c(D) = Entropy(D) - Entropy_c(D) \tag{2.5}$$

$$SplitInfo_c(D) = -\sum_{l=1}^{q} P^{(l)}(D)\log_2 P^{(l)}(D) \tag{2.6}$$

$$GainRatio_c(D) = \frac{InfoGain_c(D)}{SplitInfo_c(D)} \tag{2.7}$$

2.2.1.3 Examples of Inducing a Decision Tree

In this section, an example of the decision tree induction is demonstrated using the dataset in Table 2.2, named $D1$, as the training set. It consists of 8 instances having three attributes, i.e., Gender, Salary, and Age. It is the binary-class dataset that indicates whether the individual is likely to buy a car (Buy car? = “Yes”) or not (Buy car? = “No”).

The decision tree induction follows the Algorithm 1. It starts with creating the root node of the tree. After that, $D1$ is consider whether it corresponds to the stopping criteria or not. Since the class of all instances in the $D1$ are not same, then the splitting attribute has to be selected using the Shannon’s entropy as the splitting measure. The entropy calculation of each attribute are demonstrated as follows:

Gender	Salary	Sex	Buy car?
Male	High	18	No
Female	Medium	28	No
Female	Low	40	No
Female	Medium	48	No
Male	Medium	20	Yes
Male	Low	32	Yes
Male	High	54	Yes
Female	High	45	Yes

Table 2.2: Sample dataset $D1$

For entire dataset $D1$, three input attributes are considered.

1. The attribute Gender is a categorical attribute having two distinct values which are "Male" and "Female". Then, dataset $D1$ will be divided into two subsets according to the splitting condition $c_1 = \{$Gender : (1) Gender = "Male", (2) Gender = "Female"$\}$.

   - There are four instances that their values of attribute Gender are "Male", in which three instances are from "Yes" class and one instance is from "No" class. Therefore,

   $$Entropy(D1^{(1)}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.81128$$

   - There are four instances that their values of attribute Gender are "Female", in which one instance is from "Yes" class and three instances are from "No" class. Therefore,

   $$Entropy(D1^{(2)}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.81128$$

Hence, the entropy after partitioning dataset $D1$ by attribute Gender is equal to

$$\begin{aligned} Entropy_{c_1}(D1) &= \frac{4}{8} Entropy(D1^{(1)}) + \frac{4}{8} Entropy(D1^{(2)}) \\ &= \frac{4}{8} \cdot 0.81128 + \frac{4}{8} \cdot 0.81128 \\ &= 0.81128 \end{aligned}$$

2. The Salary attribute is a categorical attribute having three distinct values, i.e., “High”, “Medium”, and “Low”. Then, dataset $D1$ will be divided into three subsets according to the splitting condition $c_2 = \{$Salary : (1) Salary = “High”, (2) Salary = “Medium”, (3) Salary = “Low”$\}$.

   - There are three instances that their values of attribute Salary are “High”, in which two instances are from “Yes” class and one instance is from “No” class. Therefore,

   $$Entropy(D1^{(1)}) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183$$

   - There are three instances that their values of attribute Salary are “Medium”, in which one instance is from “Yes” class and two instances are from “No” class. Therefore,

   $$Entropy(D1^{(2)}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.9183$$

   - There are two instances that their values of attribute Salary are “Low”, in which one instance is from “Yes” class and one instance is from “No” class. Therefore,

   $$Entropy(D1^{(3)}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

Hence, the entropy after partitioning dataset $D1$ by attribute Salary is equal to

$$\begin{aligned} Entropy_{c_2}(D1) &= \frac{3}{8}Entropy(D1^{(1)}) + \frac{3}{8}Entropy(D1^{(2)}) + \frac{2}{8}Entropy(D1^{(3)}) \\ &= \frac{3}{8}\cdot 0.9183 + \frac{3}{8}\cdot 0.9183 + \frac{2}{8}\cdot 1 \\ &= 0.93872 \end{aligned}$$

3. The Age attribute is a numeric attribute having eight distinct values, i.e., 18, 20, 25, 32, 40, 45, 48, and 54. Then, all middle values must be considered as the candidates of splitting value and computed their values of entropy. The calculation of the value that gives the lowest entropy will be the shown here, which is 19. Thus, dataset $D1$ will be divided into 2 subsets according to the splitting condition $c_3 = \{\text{Age} : (1)\ \text{Age} < 19, (2)\ \text{Age} \geq 19\}$.

   - There is only one instance that its value of attribute Age less than 19, in which it is from "No" class. Therefore,

   $$Entropy(D1^{(1)}) = -\frac{1}{1}\log_2\frac{1}{1} = 0$$

   - There are seven instances that their values of attribute Age greater than or equal to 19, in which four instances are from "Yes" class and three instances are from "No" class. Therefore,

   $$Entropy(D1^{(2)}) = -\frac{4}{7}\log_2\frac{4}{7} - \frac{3}{7}\log_2\frac{3}{7} = 0.98523$$

   So, the entropy after partitioning dataset D1 by splitting value 19 of attribute Age is equal to

$$\begin{aligned} Entropy_{c_3}(D1) &= \frac{1}{8}Entropy(D1^{(1)}) + \frac{7}{8}Entropy(D1^{(2)}) \\ &= \frac{1}{8}\cdot 0 + \frac{7}{8}\cdot 0.98523 \\ &= 0.86207 \end{aligned}$$

From the above calculation, the attribute Gender provides the least value of entropy. Therefore, this attribute will be chosen to be the splitting condition for partitioning dataset $D1$. Then, the dataset $D1$ is divided into $D2$ and $D3$ which is shown in Table 2.3 and Table 2.4, respectively. After that, the calculation steps above is applied to $D2$ and $D3$, and recursively perform until every partitions contain only instances from the same class. The result of the decision tree induction is shown in Figure 2.9.

Gender	**Salary**	**Sex**	**Buy car?**
Male	High	18	No
Male	Medium	20	Yes
Male	Low	32	Yes
Male	High	54	Yes

Table 2.3: Dataset $D2$

Gender	**Salary**	**Sex**	**Buy car?**
Female	Medium	28	No
Female	Low	40	No
Female	Medium	48	No
Female	High	45	Yes

Table 2.4: Dataset $D3$

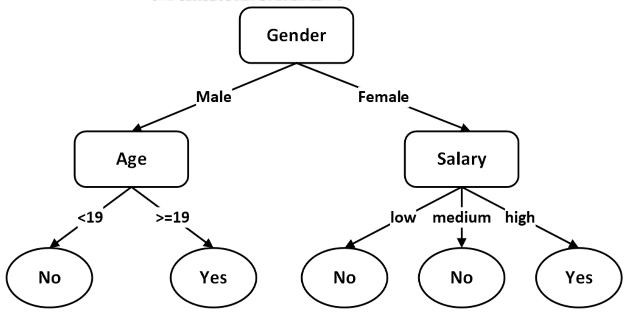


Figure 2.9: The decision tree from dataset $D1$

2.2.2 Random Forest

The random forest is a commonly-used machine learning algorithm proposed by Leo Breiman and Adele Cutler [9]. It consists of a collection of different decision trees. To build the different trees, the bootstrapping method and attribute randomization is applied. The bootstrapping method will generate several subsets of data and attribute randomization will randomly select the number of attributes to generate a decision tree. A general random forest algorithm is shown in Algorithm 2.

2.2.2.1 Bootstrap Aggregation (Bagging)

In statistics and data science, bootstrap is a resampling technique that involves repeatedly drawing samples from the dataset with replacement. The "with replacement" means that any instance may appear in the sampling dataset multiple times. This technique is normally used to estimate a population parameter.

Bootstrap sampling is used in a machine learning ensemble algorithm called bootstrap aggregating (also called bagging). It helps the machine to avoid overfitting and improves the stability of machine learning algorithms. In bagging, a certain number of equally sized subsets of a dataset are extracted with replacement. Then, a machine learning algorithm is applied to each of these subsets and the outputs are ensembled as shown in Figure 2.10.

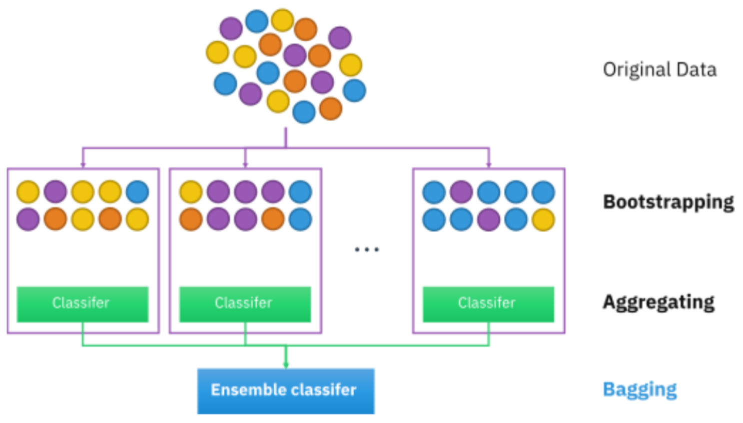


Figure 2.10: Bagging

To create one bootstrap sample, the dataset D with n cases is sampled n times. In the standard bootstrap, this sampling is uniform, i.e., each instance has the same probability to be chosen. Hence, each of the n data points in D has the same probability $\frac{1}{n}$ of being selected for the bootstrap sample. Thus, the probability that a point is not selected for the bootstrap sample is $\left(1 - \frac{1}{n}\right)^n$, which is approximately $e^{-1} \approx 0.368$ for large n.

2.2.2.2 Random forest algorithm

The pseudocode of building a random forest is shown in Algorithm 2. The first step to is apply the bootstrapping method to the original dataset D. This step generates a certain number of subsets of data. Then, the number of attributes in each subdata is randomly selected. Finally, several decision trees are built from each subdata. To predict the class of an unlabeled instance, each tree is applied to predict the class. After that, the results of each tree are aggregated and return the majority class as the final result.

```
Algorithm 2 RandomForest($D, n$)
    Input: a dataset $D$, a number of decision trees $n$
    Output: a random forest
1: for $iteration = 1, 2, ..., n$ do
2:     Generate a bootstrap sample of the training data
3:     Build a decision tree based on bootstrap sample
4:     for each split do
5:         Randomly selected $k$ features from total $d$ features
6:         Select the splitting condition among the $k$ features
7:         Partitions the data corresponding to the outcomes of the
8:         splitting conditions
```

2.3 Anomaly Detection

In statistics, an outlier is a data point that differs significantly from other observations. It can have a disproportionate effect on statistical results, such as the mean, which can result in misleading interpretations. The outlier occurs due to many causes such as system behaviour, fraudulent behaviour, human error, instrument error or simply through natural deviations in populations. Thus, the process to detect the outliers, also called anomalous detection, become an important step to purify the data before a dataset is used. Many anomaly detection techniques have been proposed in various concepts. The traditional methods such as boxplot and Z-score use statistical tools to detect outliers. However, it can only be applied on a dataset having a single attribute. Hence, the concept of assigning a score to each instance to detect the outliers become more popular. In this concept, the higher the score, the more likely the data point will be the outlier. The mentioned score is called the outlier factor. The top-n rank is normally used to screen these outliers.

2.3.1 Mass-ratio-variance Outlier Factor

In 2021, the parameter-free density-based outlier factor algorithm called Mass-ratio-variance based Outlier Factor or MOF was proposed. It uses the idea of density which is generally defined as the ratio between mass and volume. So, for a fixed volume, the ratio of density is the same as the ratio of mass. Hence, the mass-ratio can be used as the ratio between data point density. From extensive experiments, the variance of the mass-ratio distribution of outliers has a higher value than the variance of the mass-ratio distribution of normal data points. So, MOF of any data point is defined as the variance of the mass-ratio distribution among other data points where the mass-ratio of the computed data point is defined as the ratio of the number of data points within the sphere of the distance from this computed data point to that of other data points. The main mathematical notations are defined as follows:

Definition 2.1. (Distance between $\boldsymbol{p}$ and $\boldsymbol{q}$) Given a dataset $D \subseteq \mathbb{R}^d$, the Euclidean distance of data point $\boldsymbol{p} = (p_1, ..., p_d) \in D$ to data point $\boldsymbol{q} = (q_1, ..., q_d) \in D$ denoted as $d(\boldsymbol{p}, \boldsymbol{q})$ is defined as

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$

Definition 2.2. (Neighborhoods of data point $\boldsymbol{q}$ with respect to data point $\boldsymbol{p}$). Given a dataset $D \subseteq \mathbb{R}^d$, the set of all data points within the neighborhood of data point $\boldsymbol{q} \in D$ with respect to data point $\boldsymbol{p} \in D$ is defined as the set of points that lies within the ball centered at data point $\boldsymbol{q}$ with the radius $d(\boldsymbol{q}, \boldsymbol{p})$:

$$N_p(q) = \{\boldsymbol{o} \in D | d(\boldsymbol{q}, \boldsymbol{o}) \leq d(\boldsymbol{q}, \boldsymbol{p})\}$$

Definition 2.3. (The mass-ratio of data point $\boldsymbol{q}$ with respect to data point $\boldsymbol{p}$) Given a dataset $D \subseteq \mathbb{R}^d$ and data point $\boldsymbol{p} \in D$. For any data point $\boldsymbol{q} \in D - \{\boldsymbol{p}\}$, the mass-ratio of data point $\boldsymbol{q}$ with respect to data point $\boldsymbol{p}$ is defined as

$$MassR_p(q) = \frac{|N_p(q)|}{|N_q(p)|}$$

Definition 2.4. (MOF of data point $\boldsymbol{p}$) Given a dataset $D \subseteq \mathbb{R}^d$ and the number of data points in D is n. For data point $\boldsymbol{p} \in D$, μ_p is defined as mean of mass-ratio distribution of data point $\boldsymbol{p}$ and MOF of data point $\boldsymbol{p}$ is defined as the variance of the mass-ratio distribution of data point $\boldsymbol{p}$:

$$\mu_p = \frac{\sum_{i=1, q_i \neq p}^{n} MassR_p(q_i)}{n-1}$$

$$\mathrm{MOF}(\boldsymbol{p}) = \frac{\sum_{i=1, q_i \neq p}^{n} (MassR_p(q_i) - \mu_p)^2}{n-1}$$

2.4 Class Imbalanced Problem

In real-world situations, many datasets suffer from the class imbalanced problem which refer to the problem of unequal class distribution in the dataset. The dataset is considered imbalanced when the number of instances in each class is significantly different. The binary-class dataset is the dataset having 2 different classes while the multi-class dataset have more than 2 classes. A binary-class dataset that is imbalanced is called binary-class imbalanced dataset which can be found in many real-world datasets. Many researchers have proposed new techniques to handle binary-class imbalanced dataset including Minority condensation entropy or MCE.

2.4.1 Minority Condensation Entropy

In a binary-class imbalanced dataset, a class having a larger number of instances is called the majority class or the negative class, and a class having a smaller number of instances is called the minority class or the positive class. To measure the degree of imbalanced for each binary-class dataset, the imbalanced ratio (I.R.) is used, which is defined by the ratio between the number of instances in the majority class and the minority class as shown in (2.8), in which m_- refers to the number of instances from the majority class, and m_+ refers to the number of instances from the minority class. For the problem of building the model to classify the binary-class imbalanced dataset, it is called the binary-class imbalanced problem. In this case, a classifier tends to predict an instance to be the majority class which causes the misclassification for the minority instances.

$$I.R. = \frac{m_-}{m_+} \tag{2.8}$$

In 2019, a new splitting measure called Minority Condensation Entropy (MCE) [23] is proposed to handle binary-class imbalanced dataset having only numerical attributes. This splitting measure is modified from the minority entropy (ME) [7] which ignores majority instances outside the range for all minority instances, called the minority range. All instances outside the range that would not affect the ability to predict minority class instances are excluded. However, if some minority instance values are outliers, the minority range will be wider and cover more unnecessary majority instances. Therefore, MCE which is the improvement of ME is proposed. The pseudocode of MCE is presented in Algorithm 3. In MCE, the interquartile range (IQR) rule is applied to the set of minority instance values for detecting the outliers. It defines the boundary that represents the range of acceptable values for the minority instances based on the Tukey's boxplot [28]. The lower inner fence is defined by the first quartile minus 1.5 times of IQR, while the upper inner fence is defined by the third quartile plus 1.5 times of IQR. The example of IQR is shown in Figure 2.11.

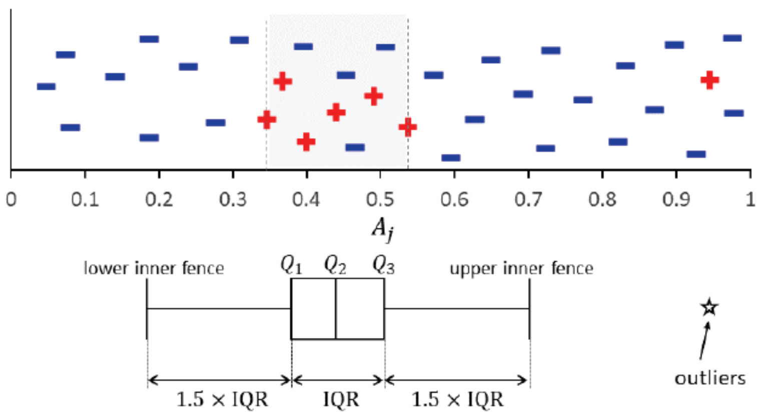


Figure 2.11: Applying the interquartile range rule to detect outliers before determining the minority range

Let s be a function that maps each instance $\boldsymbol{x}_i$ in dataset D to the set of real numbers, i.e., $s : D \to \mathbb{R}$. For example, with the two-dimensional dataset $D \subseteq \mathbb{R}^2$, define the instance value function as $s(\boldsymbol{x}_i) = s(x_1^i, x_2^i) = x_1^i + x_2^i$. Then, for any set of instance values $s(D) = \{s(\boldsymbol{x}_i) \in \mathbb{R} | (\boldsymbol{x}_i, y_i) \in D \text{ for } i = 1, 2, ..., m\}$ corresponding to D, define $s^*(D)$ be the subset of $s(D)$ that ignores the outliers shown in (2.9) as follows:

$$s^*(D) = \{s(\boldsymbol{x}_i) \in s(D) | \\ Q_1 - 1.5 \times IQR \leq s(\boldsymbol{x}_i) \leq Q_3 + 1.5 \times IQR \text{ for } i = 1, 2, ..., m\} \quad (2.9)$$

where Q_1 is the first quartile of $s(D)$, Q_3 is the third quartile of $s(D)$, and IQR is the interquatile range of $s(D)$ which is equal to $Q_3 - Q_1$.

Then, the minority range that ignores the outliers of $s(D)$ is determined by the range between the minimum value and the maximum value of $s^*(D_+)$, i.e., $\phi^*_+(s(D)) = [\min s^*(D_+), \max s^*(D_+)]$ where D_+ denotes the set of minority instances from dataset D. Then, $\Phi^*_+(s(D))$ implies the subset of instances within the minority range that ignores the outliers, $\Phi^*_+(s(D)) = \{(\boldsymbol{x}_i, y_i) \in D | s(\boldsymbol{x}_i) \in \phi^*_+(s(D)) \text{ for } i = 1, 2, ..., m\}$. Thus, the definition of the minority condensation entropy according to $s(D)$ is determined by (2.10) as follows:

$$MCE_s(D) = Entropy(\Phi^*_+(s(D))) \quad (2.10)$$

Algorithm 3 MCE(D, s)

```
    Input: a dataset $D$, a function to obtain the instance values $s$
    Output: the minority condensation entropy of $D$ with respect to $s$
1:  Generate the set of instance values corresponding to the minority class without
    the outliers: $s^*(D_+)$
2:  Create the minority range that ignores the outliers: $\phi^*_+(s(D))$
3:  Compute the subset of instances within the minority range that ignores the
    outliers: $\Phi^*_+(s(D))$
4:  return $Entropy(\Phi^*_+(s(D)))$
```

2.4.2 Minority Condensation Decision Tree

The minority condensation decision tree or MCDT is a decision tree which apply MCE to build and the algorithm to build MCDT is called MCDT algorithm which its pseudocode is shown in Algorithm 4. Mathematically, let $\pi_j(D)$ denotes a set of instance values along attribute A_j, i.e., $\pi_j(D) = \{\pi_j(\boldsymbol{x}_j) = x_j^i | (\boldsymbol{x}_i, j_i) \in D$ for $i = 1, 2, ..., m\}$ For each attribute A_j, the minority range that ignores the outliers of $\pi_j(D)$, denoted by $\phi_+^*(\pi_j(D))$, is determined by the range between the minimum value and the maximum value of $\pi_j^*(D)$, i.e. $\phi_+^*(\pi_j(D)) = [\min \pi_j^*(D_+), \max \pi_j^*(D_+)]$ and the subset of instances within the minority range that ignores the outliers, denoted by $\Phi_+^*(\pi_j(D))$, can be written as $\Phi_+^*(\pi_j(D)) = \{(\boldsymbol{x}_i, y_i) \in D | s(\boldsymbol{x}_i) \in \phi_+^*(\pi_j(D))$ for $i = 1, 2, ..., m\}$. Thus, the minority condensation entropy according to attribute A_j is determined by (2.11) as follows:

$$MCE_{\pi_j}(D) = Entropy(\Phi_+^*(\pi_j(D))) \tag{2.11}$$

Algorithm 4 MCDT(D)

```
    Input: a dataset $D$
    Output: a minority condensation decision tree
 1: Create the root node of the tree
 2: if D correspond to the stopping criteria then
 3:     return the leaf node with respect to the majority class
 4: else
 5:     for each attribute $A_j$ do
 6:         Apply the greedy approach on $\pi_j(D)$ based on the minority
 7:         condensation entropy via Algorithm 3
 8:         Update the best splitting condition
 9:     Obtain the splitting condition
10:     Separate the dataset into 2 partitions corresponding to the outcomes of
11:     splitting conditions: $D = D^{(left)} \cup D^{(right)}$
12:     Iterate for each partition MCDT($D^{(left)}$) and MCDT($D^{(right)}$)
```

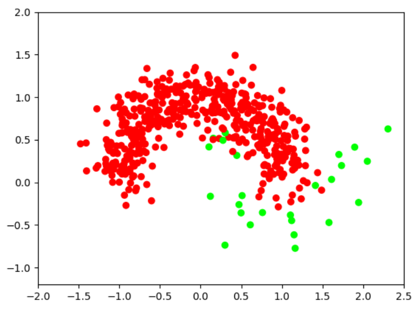
CHAPTER III

RESEARCH METHODOLOGIES

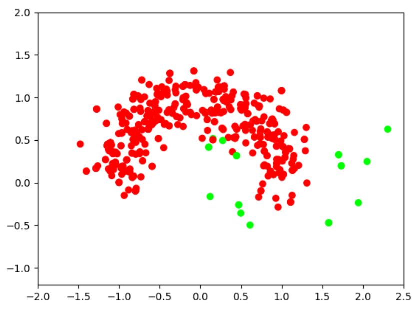
In this chapter, an updated random forest algorithm handling class imbalanced dataset in classification is proposed. The first section presents an improved bootstrapping method called quartile-pattern bootstrapping. It applies MOF to maintain the structure of the bootstrap data. After that, the quartile-pattern bootstrapping is used in the random forest algorithm to build a random forest called the random forest using quartile-pattern bootstrapping or RFQP where the methodology and its algorithm is demonstrated in the second section.

3.1 Quartile-pattern Bootstrapping

In the random forest algorithm, the bootstrapping method is used to generate several subsets of data from the original dataset. However, when bootstrapping is applied to the imbalanced data, the probability that the minority instances appear in the bootstrap data is extremely low. Thus, the structure of the minority instances in the bootstrap data has been changed which affects the performance of each decision tree in random forest. Figure 3.1 shows the example of bootstrap data compared to its original data. The orginal data is two-moons data which has the distribution of the instances in two classes each in crescent moon shape. However, after bootstrapping, the distribution of the minority instance is not in crescent moon shape. To prevent this problem, an enhanced bootstrapping method called quartile-pattern bootstrapping is proposed.



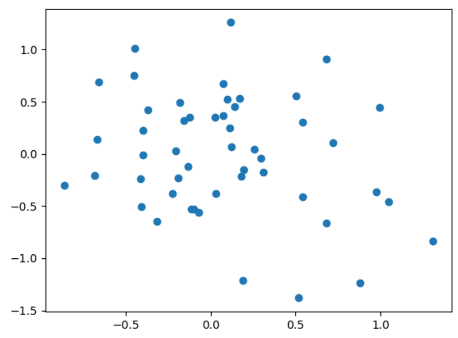
(a) The original data



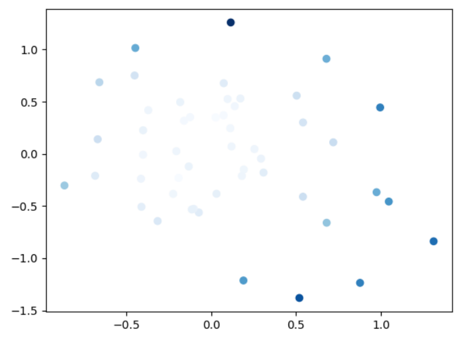
(b) A bootstrap data from normal bootstrapping method

Figure 3.1: An example of bootstrapping method

The idea to maintain the structure of the minority instances in bootstrap data is to keep outliers and borderline instances of the minority instances since they are the instances that outline the structure of data. To identify the outliers and borderline instances, an outlier factor algorithm called mass-ratio-variance outlier factor or MOF is applied. The important property of MOF is that it can identify whether the instance is outlier, borderline instance, or normal instance since MOF produces the low value for the instance surrounded by other instances, and the high value for the instance positioned very far from any instance. In Figure 3.2, the example data and those which are colored according to MOF is showned. The color of the outliers and borderline instances is darker than the normal instances since MOF of outliers and borderline instances are higher than normal instances. In general, the users can define the number of instances to be the outliers. For this method, the half of the instances with highest MOF is assume to be outliers and borderline instances to keep. After that, the bootstrapping method is applied to the rest of minority instances and the majority instances separately.



(a) The original data



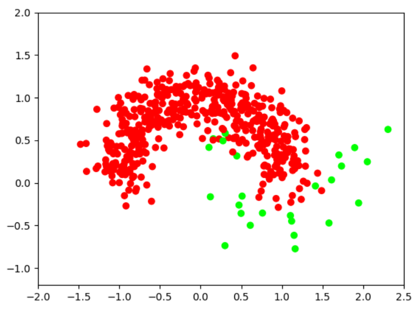
(b) The data colored according to MOF

Figure 3.2: A distribution of the example data colored according to MOF

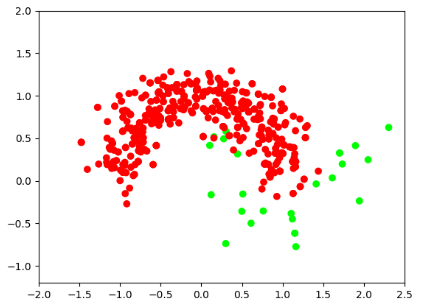
3.1.1 Proposed Algorithm

Figure 3.4 presents the framework of the quartile-pattern bootstrapping technique and its algorithm is described in Algorithm 5. This algorithm requires two parameters which are dataset $D = \{(\boldsymbol{x}_i, y_i) | i = 1, 2, ..., m\}$, where m is a number of instances in the data, and the number of bootstrap data N. Here, D is a binary-class imbalanced dataset consisting only of numeric attributes. The algorithm starts with dividing the dataset into the dataset of the minority class D^{Minor} and the dataset of the majority class D^{Major}. For the minority class dataset D^{Minor}, the MOF algorithm is applied to calculate MOF of each instance. Then the minority instances are sorted according to MOF in ascending order and divided into four quartiles which are D^{Q1}, D^{Q2}, D^{Q3}, and D^{Q4}. The D^{Q4} contains the 25% of minority instances having highest MOF while D^{Q1} contains the 25% of minority instances having lowest MOF. The instances in D^{Q3} and D^{Q4} are assumed to be outliers and some border instances since their MOF are high while the instances in the D^{Q1} and D^{Q2} are the normal points that are located near the center of the cluster. After the data is partitioned, the normal bootstrapping method is applied to D^{Q1}, D^{Q2}, and D^{Major} which generate N bootstrap data for each data. Finally, each of the obtained sampling subsets are merged with instances from D^{Q3} and

D^{Q4}. The method outputs several subsets of the dataset which are appropriate to build decision trees. Figure 3.3(b) shows the example of bootstrap data from quartile-pattern bootstrapping on a dataset in Figure 3.3(a). The result confirms that the bootstrap data from quartile-pattern bootstrapping can maintain the structure of the original data.



(a) The original data



(b) A bootstrap data from quartile-pattern bootstrapping method

Figure 3.3: An example of the bootstrapping method

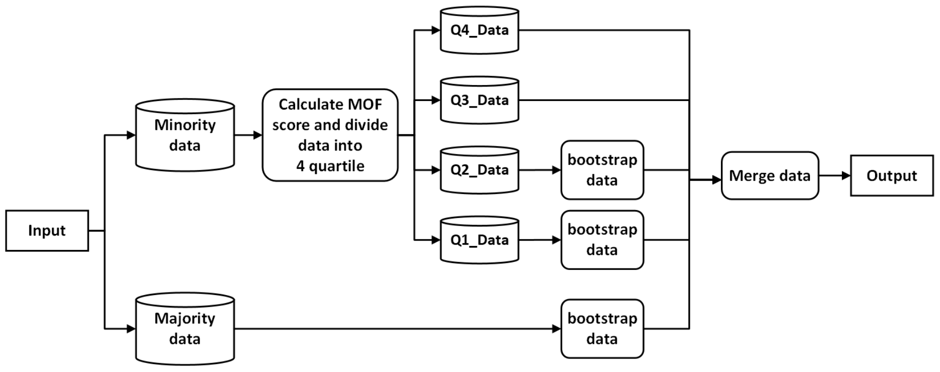


Figure 3.4: A framework of the quartile-pattern bootstrapping

3.1.2 Time Complexity

The time complexity of Algorithm 5 can be derived from each step dropping all lower order terms. Line 1 generates an empty set for collecting bootstrap data having the time complexity of $O(1)$. Line 2 partitions a dataset D into D^{Minor} and D^{Major} having the time complexity of $O(m)$. Line 3 performs the MOF algorithm on D^{Minor} having the time complexity of $O(d|D^{Minor}|^2 \log|D^{Minor}|)$ where d represents the number of attributes in dataset D and $|D^{Minor}|$ represents the number of instances in D^{Minor}. Line 4 sorts D^{Minor} according to MOF and divide D^{Minor} into 4 quartiles which are D^{Q1},D^{Q2},D^{Q3}, and D^{Q4} having the time complexity of $O(|D^{Minor}| \log|D^{Minor}|)$. Line 5 performs n bootstraps on D^{Q1} having the time complexity of $O(n|D^{Minor}|)$. Line 6 performs n bootstraps on D^{Q2} having the time complexity of $O(n|D^{Minor}|)$. Line 7 performs n bootstraps on D^{Major} having the time complexity of $O(n|D^{Major}|)$ where $|D^{Major}|$ represents the number of instances in D^{Major}. Then, n loops to combine five subdata and add to bootstrap_collection is performed having the time complexity of $O(n)$. Hence, the time complexity for this Algorithm is $O(n|D^{Major}|)$ or about $O(nm)$ wher m is the number of instances in a dataset D since $|D^{Minor}|$ is significantly smaller than $|D^{Major}|$.

Algorithm 5 Quartile-pattern bootstrapping(D, n)

```
    Input: a dataset $D$, a number of bootstrap data $n$
    Output: a collection of bootstrapping sampling subsets
 1: bootstrap_collection $\leftarrow$ empty set
 2: Partition $D$ into $D^{Minor}$ and $D^{Major}$
 3: Compute MOFs of all instances in $D^{Minor}$
 4: Divide $D^{Minor}$ into 4 quartiles: $D^{Q1}, D^{Q2}, D^{Q3}, D^{Q4}$
 5: Perform bootstrapping on $D^{Q1} \rightarrow D_1^{Q1}, D_2^{Q1}, ..., D_n^{Q1}$
 6: Perform bootstrapping on $D^{Q2} \rightarrow D_1^{Q2}, D_2^{Q2}, ..., D_n^{Q2}$
 7: Perform bootstrapping on $D^{Major} \rightarrow D_1^{Major}, D_2^{Major}, ..., D_n^{Major}$
 8: for $i = 1$ to $n$ do
 9:     Merge $D_i^{Q1}, D_i^{Q2}, D_i^{Major}, D^{Q3}$, and $D^{Q4} \rightarrow D_i$
10:     Add $D_i$ to bootstrap_collection
```

3.2 Random Forest Using Quartile-pattern Bootstrapping

The random forest algorithm using quartile-pattern bootstrapping or RFQP applies the quartile-pattern bootstrapping technique and build both MCDT and a standard decision tree on each subset.

3.2.1 Proposed Algorithm

The workflow of RFQP is shown in Figure 3.5 and the algorithm is described in Algorithm 6. It starts with applying the quartile-pattern bootstrapping to the dataset. The outputs from the quartile-pattern bootstrapping step are subsets of the training dataset having the similar structure as the training dataset. Then, the decision tree algorithm and the MCDT algorithm are applied to each subset. Therefore, each subset will generate two decision trees which are the traditional decision tree and MCDT. Finally, all trees are aggregated together, and then returned as the random forest.

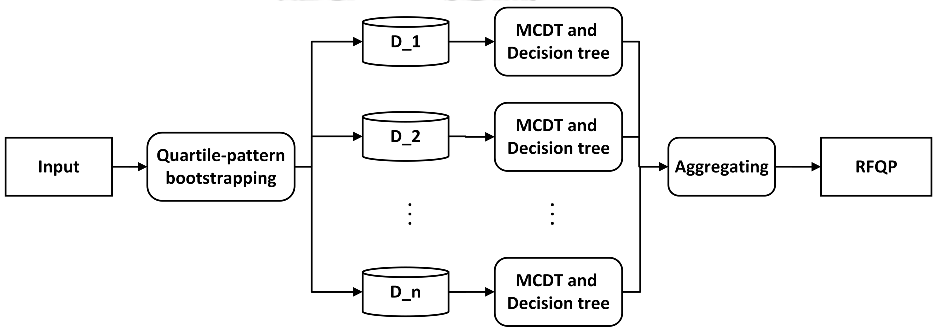


Figure 3.5: A framework of building the random forest using quartile-pattern bootstrapping

Algorithm 6 RFQP(D, n)

```
    Input: a dataset $D$, a number of bootstrap data $n$
    Output: a random forest
1: Forest $\leftarrow$ empty set
2: Perform quartile-pattern bootstrapping$(D, n) \rightarrow D_1, D_2, ..., D_n$
3: for $i = 1$ to $n$ do
4:     Build a decision tree from $D_i \rightarrow Tree_i^1$
5:     Build a MCDT from $D_i \rightarrow Tree_i^2$
6:     Add $Tree_i^1$ and $Tree_i^2$ to Forest
```

3.2.2 Time Complexity

The time complexity from Algorithm 6 can be computed from each step dropping all lower order terms. Line 1 takes $O(1)$. Line 2 takes $O(nm)$ to call Algorithm 5. Then it performs n loop for building a decision tree and MCDT having $O(nm^2d)$. Hence, the time complexity for this Algorithm 6 is $O(nm^2d)$ which is the work for building a decision tree in each bootstrapping subset.

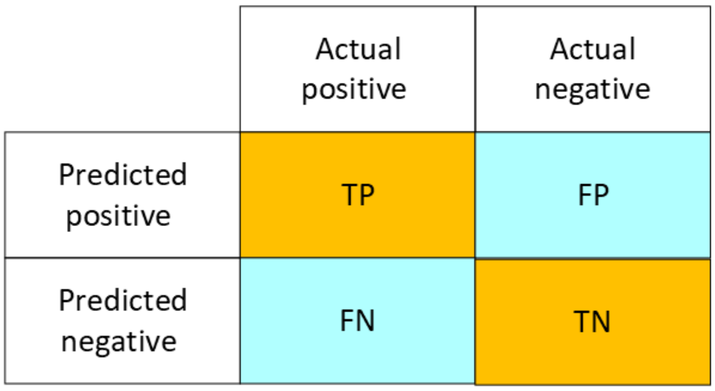
CHAPTER IV

EXPERIMENTS AND RESULTS

This chapter presents all experiments which are implemented on the Python programming language. The experiments are divided into two parts. In the first part, the performance of RFQP is compared with the standard random forest based on two synthetic dataset having 1000 instances. In the second part, the performance of RFQP is compared with three algorithms which are the standard random forest, the MCDT, and the random forest with Safe-level SMOTE, based on 10 real-world datasets.

4.1 Evaluation metrics

An evaluation metrics are the measurement used to evaluate the performance of a predictive model. In the experiments, four metrics, Precision, Recall, F-Measure (F1), and Geometric mean (GM), are used to compare the performance of the models. These measures are computed from the values in the confusion matrix shown in Figure 4.1.



	Actual positive	Actual negative
Predicted positive	TP	FP
Predicted negative	FN	TN

Figure 4.1: Confusion matrix

The entries in the confusion matrix are TP, FP, FN, and TP:

- True Positive (TP) is the number of positive instances that are correctly predicted as positive instances.
- True Negative (TN) is the number of negative instances that are correctly predicted as negative instances.
- False Positive (FP) is the number of negative instances that are incorrectly predicted as positive instances.
- False Negative (FN) is the number of positive instances that are incorrectly predicted as negative instances.

The formula of four metrics are shown below.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

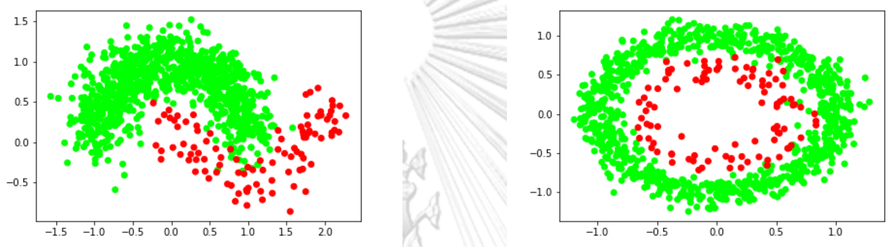
$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.3}$$

$$GM = \sqrt{Precision \times Recall} \tag{4.4}$$

4.2 Experiments on Synthetic Datasets

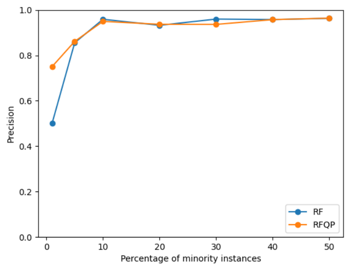
RFQP is compared with the traditional random forest algorithm based on two collections of synthesized datasets. The first collection has two-class instances forming the moon structures for each class overlapping with one another (see Figure 4.2(a)). The second collection has two-class instances forming a circle from one class inside another circle from another class (see Figure 4.2(b)). In each collection, there are seven subcollections having different percentages of minority instances from 5%, 10%, 15%, 20%, 30%, 40%, and 50%. A dataset in a subcollection has two numeric attributes with the total instances equal to 1,000. Each experiment is repeated 10 times and the average performance is used in the report. In each run, a dataset is split into a training dataset and test datasets with ratio 8:2. After that, the proposed algorithm and the traditional random forest algorithm are used on the training set to generate the classification models. Then, the obtained models are evaluated on test dataset based on precision, recall, F-measure and GM.



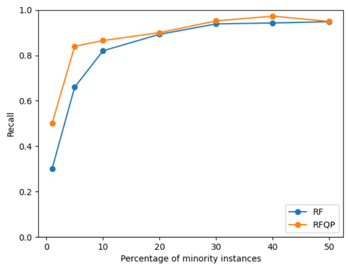
(a) Two-moons dataset **(b)** Two-circles dataset

Figure 4.2: A sample of the two-moons dataset and the two-circles dataset

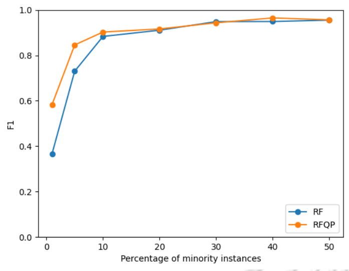
The experimental results are shown in Figures 4.4 and 4.3. In Figure 4.4, recall, F-measure, and GM of RFQP are higher than those of RF while precision of RFQP is better than RF only at 1%, 5%, and 50% minority percentage data for the two-circle dataset. In Figure 4.3, recall, F-measure, and GM of RFQP are higher than those of RF while precision of RFQP is higher than RF only at 1%, 5%, 20%, and 50% minority percentage data. The result confirms that the quartile-pattern bootstrapping technique can easily keep the border instances of the circle while the standard bootstrapping method may lose this structure randomly. In addition, the performance of RF and RFQP tend to be equal when the minority percentage of data converges to 50% that is the data become more balance.



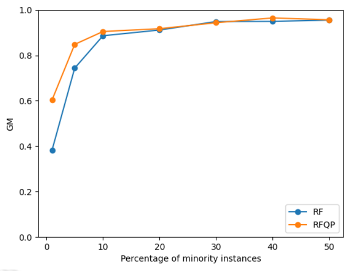
(a) Precision



(b) Recall

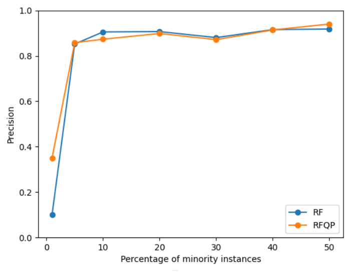


(c) F-measure

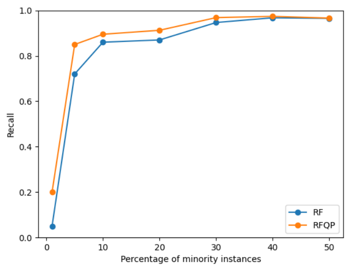


(d) GM

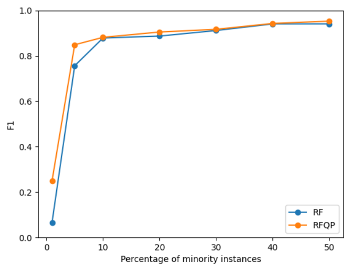
Figure 4.3: The experimental results on two-moons datasets



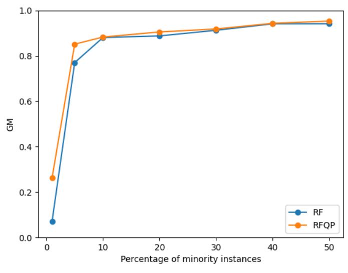
(a) Precision



(b) Recall



(c) F-measure



(d) GM

Figure 4.4: The experimental results on two-circles datasets

4.3 Experiments on Real-world Datasets

In this section, the performance of RFQP is evaluated with the experiments on real-world datasets. The real-world datasets which are used in this thesis are from the UCI repository[5] having a total of 10 datasets with the selected classes to be treated as the binary-class imbalanced problems. The information of the datasets are shown in Table 4.1. The first two columns represent the index and the name of each dataset. For the third column and the fourth column, they indicate the number of instances (# Inst.) and the number of attributes (# Att.), respectively. Particularly, the classes determining as the majority class (Maj. Cl.) and the minority class (Min. Cl.) are shown in the fifth and the sixth columns. The last column shows the imbalanced ratio of the data.

No.	Dataset	# Inst.	# Att.	Maj. Cl.	Min. Cl.	I.R.
1	OpticDigits	1108	64	"0"	"8"	1
2	Pima	768	8	"0"	"1"	1.87
3	StatlogVehicle	846	18	The rest	"bus"	2.88
4	LibrasMovement	360	90	The rest	"1", "2", "3"	4
5	BreastTissue	106	9	The rest	"fad"	6.07
6	Ecoli	336	7	The rest	"imU"	8.6
7	StatlogLandsat	6435	36	The rest	"4"	9.28
8	Glass	214	9	The rest	"5"	15.46
9	PageBlocks	5473	10	The rest	"2"	15.64
10	Winequality-red	1599	11	The rest	"3", "4"	24.38

Table 4.1: The characteristics of the experimental real-world binary-class datasets

In the experiment, each dataset is divided into the training set and the testing set with ratio 8:2. After that, RFQP and other 3 classifiers are built from the training set and their performance are evaluated using the testing set. The experiment are repeated 10 times on each dataset. The other three classifiers are a standard random forest, MCDT, and Safe-Level SMOTE [10]+RF. The average results of each classifier are shown in Table 4.2, in which the higher value indicates the better performance.

Metrics	RFQP	RF	MCDT	Safe-Level SMOTE + RF
Precision	0.741417	**0.770203**	0.678107	0.601428
Recall	0.660596	0.632240	0.629465	**0.811199**
F-measure	**0.681022**	0.674160	0.640925	0.668972
GM	**0.690296**	0.686571	0.647146	0.686464

Table 4.2: The average experimental results on the real-world binary-class datasets

For comparison by precision, RFQP provide the average precision at 0.741417 which is lower than RF but higher than MCDT and Safe-level SMOTE+RF. It means that the number of RFQP predicted minority instances to be the majority class is lower than MCDT and Safe-level SMOTE+RF but higher than RF.

In addition, the average recall of RFQP is at 0.660596 which is less than Safe-level SMOTE+RF but higher than RF and MCDT. This means that the number of RFQP predicted majority instances to be the minority class is more than Safe-level SMOTE+RF but less than RF and MCDT.

For comparison by F-measure and GM, the average performance of RFQP is at 0.681022 and 0.690296 respectively, which are higher than the other models.

CHAPTER V

CONCLUSIONS

This thesis enhances the standard random forest algorithm or RF using the quartile-pattern bootstrapping technique and building both the standard decision tree and the minority condensation decision tree on each bootstrapped subset. The improved random forest algorithm is called the random forest algorithm using quartile-pattern bootstrapping or RFQP which can handle imbalanced binary datasets. The performance RFQP are reported by two collections of experiments on synthetic binary-class imbalanced numeric datasets and real-world binary-class datasets from UCI, respectively.

The first experiment uses two collections of synthesized datasets which are two-moons and two-circles datasets to compare the performance of RF and RFQP. The results show that RFQP outperforms RF on recall, F-measure, and GM. In the second experiments, 10 real-world datasets are used to compare the performance of RFQP to three models and the results show that RFQP outperforms RF, MCDT, and Safe-level SMOTE+RF on F-measure and GM. Hence, RFQP outperforms RF, MCDT, and Safe-level SMOTE+RF on both synthesized datasets and real-world datasets.

Discussion

From the experimental results on synthesized dataset, the average result RF is better than RFQP for some datasets. This situation occurs because there is an overlapping region between majority dataset and minority dataset. Therefore, in the RFQP algorithm, there are some majority instances that are predicted to be minority instances which cause false positive and decrease the precision of RFQP.

On the other hand, the minority instances have morce chance to be correctly predicted since trees in the RFQP are built from the data that consist of outliers and borderline datapoints of the original data. Thus, the true positive of RFQP increases. Consequently, the recall of RFQP is better than RF. In addition, the performance of RF and RFQP tend to be equal when the minority percentage of data converges to 50% because the performance of MCDT in RFQP is not different from a normal decision tree when the data is balance.

For the experimental results on real-world dataset, the average recall of RFQP is significantly lower than Safe-level SMOTE+RF. This happens because of the number of minority instances in bootstrap data may not enough to maintain the structure of the data and the outliers and borderline datapoints from MOF may not be the real one since the distance between 2 datapoints in MOF is calculated directly but each attribute does not have a same unit so it can not be calculated appropriately.

Future Works

Since RFQP was only compared with three models in the experiment, this proposed algorithm can be extended by comparing RFQP with other existing models, such as neural networks and support vector machines. In addition, although RFQP is currently limited to binary-class imbalanced datasets with numeric attributes, there is potential to extend the applicability of this algorithm to multi-class imbalanced datasets, including those with categorical attributes. However, adapting the concept of anomaly detection would be necessary, as the current approach is not suitable for datasets containing categorical attributes.

REFERENCES

[1] A cost-sensitive decision tree approach for fraud detection. *Expert Systems with Applications*, 40(15):5916–5923, 2013.

[2] A. Ali, S. M. Shamsuddin, and A. Ralescu. Classification with class imbalance problem: A review. 7:176–204, 01 2015.

[3] E. Alpaydın. *Introduction to Machine Learning, Third Edition.* The MIT Press, 2014.

[4] D. Anbarasi and V. Radha. Classification for big data driven marine weather forecasting using machine learning techniques. *International Journal of Scientific Research in Engineering and Management*, 07, 02 2023.

[5] C. Blake, E. Keogh, and C. Merz. UCI repository of machine learning databases. 01 1998.

[6] M. Block-Berlitz, M. Bader, E. Tapia, M. Ramirez-Ortegon, K. Gunnarsson, E. Cuevas, D. Zaldivar, and R. Rojas. Using reinforcement learning in chess engines. 35:31–40, 01 2008.

[7] K. Boonchuay, K. Sinapiromsaran, and C. Lursinsap. Decision tree induction based on minority entropy for the class imbalance problem. *Pattern Analysis and Applications*, 20, 08 2017.

[8] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifier. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 5, 08 1996.

[9] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.

[10] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. Safe-level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. volume 5476, pages 475–482, 04 2009.

[11] P. Changsakul, S. Boonsiri, and K. Sinapiromsaran. Mass-ratio-variance based outlier factor. In *2021 18th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5, 2021.

[12] R. Chen. A brief introduction on shannon's information theory, 01 2016.

[13] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[14] K. F. H. A. Elseddawy, A. and D. Khafaga. Predictive analysis of diabetes-risk with class imbalance. *Computational Intelligence and Neuroscience*, 2022, 10 2022.

[15] W. Fan, S. Stolfo, J. Zhang, and P. Chan. Adacost: Misclassification cost-sensitive boosting. *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, 05 1999.

[16] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 55:119–139, 12 1999.

[17] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[18] I. Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[19] A. I. Khan and S. Al-Habsi. Machine learning in computer vision. *Procedia Computer Science*, 167:1444–1451, 2020. International Conference on Computational Intelligence and Data Science.

[20] E. Ogheneovo and P. Nlerum. Iterative dichotomizer 3 (id3) decision tree: A machine learning algorithm for data classification and predictive analysis. *International Journal of Advanced Engineering Research and Science*, 7:514–521, 01 2020.

[21] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, mar 1986.

[22] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[23] A. Sagoolmuang. Self-balancing recursive partitioning algorithm for classification problems. 2019.

[24] A. Sagoolmuang and K. Sinapiromsaran. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 05 2002.

[25] A. Sagoolmuang and K. Sinapiromsaran. Oblique decision tree algorithm with minority condensation for class imbalanced problem. *Engineering Journal*, 24:221–237, 02 2020.

[26] P. Sarang. *Naive Bayes*, pages 143–152. Springer International Publishing, Cham, 2023.

[27] I. Tetko, D. Livingstone, and A. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35:826–833, 09 1995.

[28] J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.

[29] Z.-H. Zhou and X.-Y. Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *Knowledge and Data Engineering, IEEE Transactions on*, 18:63– 77, 02 2006.

[30] M. Zhu, J. Xia, X. Jin, M. Yan, G. Cai, J. Yan, and G. Ning. Class weights random forest algorithm for processing class imbalanced medical data. *IEEE Access*, 6:4641–4652, 2018.

BIOGRAPHY

Name	Mr. Worawit Jitpakdeebodin
Date of Birth	February 14, 1999
Place of Birth	Nakhonratchasima, Thailand
Educations	B.Eng. (Computer) (First Class Honours), Chulalongkorn University, 2021

Publications

- W. Jitpakdeebodin and K. Sinapiromsaran. Random forest algorithm using quartile-pattern bootstrapping for a class imbalanced problem. In Proceedings of the 2023 5th International Conference on Image, Video and Signal Processing, IVSP '23, page 191–196, New York, NY, USA, 2023. Association for Computing Machinery.