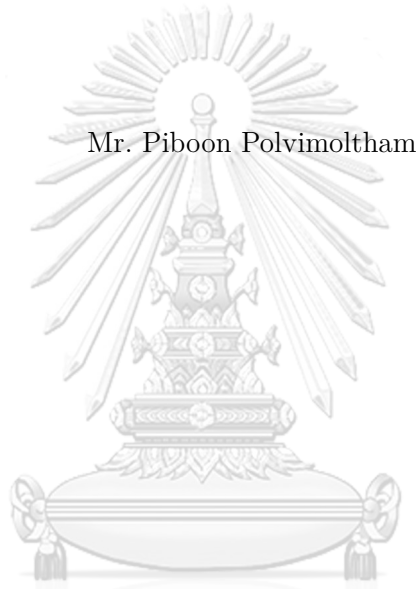
กระบวนการทำความสะอาดตัวอย่างคลาสที่มีจำนวนมากและเพิ่มตัวอย่างคลาสที่มีจำนวนน้อยโดยใช้ความแปรปรวนของอัตราส่วนมวล

นายพิบูล ผลวิมลธรรม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2565

MASS RATIO VARIANCE MAJORITY CLEANSING AND MINORITY OVERSAMPLING TECHNIQUE FOR CLASS IMBALANCED

Mr. Piboon Polvimoltham



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	MASS RATIO VARIANCE MAJORITY CLEANSING AND MINORITY OVERSAMPLING TECHNIQUE FOR CLASS IMBALANCED
By	Mr. Piboon Polvimoltham
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

..... Dean of the Faculty of Science
(Professor Polkit Sangvanich, Ph.D.)

THESIS COMMITTEE

..... Chairman
(Assistant Professor Kitiporn Plaimas, Ph.D.)

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

..... Examiner
(Thap Panitanarak, Ph.D.)

..... External Examiner
(Assistant Professor Chumphol Bunkhumpornpat, Ph.D.)

พิบูล ผลวิมลธรรม : กระบวนการทำความสะอาดตัวอย่างคลาสที่มีจำนวนมากและเพิ่มตัวอย่างคลาสที่มีจำนวนน้อยโดยใช้ความแปรปรวนของอัตราส่วนมวล. (MASS RATIO VARIANCE MAJORITY CLEANSING AND MINORITY OVERSAMPLING TECHNIQUE FOR CLASS IMBALANCED) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร.กรุง สินอภิรมย์สราญ, 57 หน้า.

ขั้นตอนวิธีการสุ่มตัวอย่างเป็นหนึ่งในขั้นตอนวิธีพื้นฐานในการจัดการกับปัญหาคลาสไม่ได้ดุลซึ่งปรากฏในเซตข้อมูลการเรียนรู้ของเครื่อง ชุดข้อมูลที่มีปัญหาความไม่สมดุลจะมีการกระจายของข้อมูลเบ้ไปยังกลุ่มใดกลุ่มหนึ่ง เทคนิคการสุ่มตัวอย่างมี 3 ประเภทซึ่งสามารถใช้ในการแก้ปัญหาคลาสไม่สมดุลได้โดยการปรับดุลของการกระจายตัวของคลาส ได้แก่ เทคนิคการสุ่มลดตัวอย่าง เทคนิคการสุ่มตัวอย่างเพิ่ม และเทคนิควิธีผสมรวมกันของทั้งเทคนิคการสุ่มลดตัวอย่างและเทคนิคการสุ่มตัวอย่างเพิ่ม ในวิทยานิพนธ์นี้คะแนนความแปรปรวนของอัตราส่วนมวลของแต่ละตัวอย่างจะถูกคำนวนแยกคลาส จากนั้นจะถูกใช้กำจัดข้อมูลรบกวนออกจากคลาสส่วนมากและทำการสังเคราะห์ตัวอย่างเพิ่มในคลาสส่วนน้อย ผลลัพธ์ของขั้นตอนวิธีสุ่มที่ถูกเสนอ ปรับปรุงค่ารีคอลให้ดีขึ้นโดยใช้ตัวจำแนกประเภทมาตรฐาน ต้นไม้ตัดสินใจ ป่าสุ่ม ซัพพอร์ตเวกเตอร์แมทชีนแบบเชิงเส้นและ เพอร์เซ็ปตรอนหลายชั้นเหล่านี้ทดสอบกับชุดข้อมูลสังเคราะห์ การรายงานประสิทธิภาพบนชุดข้อมูลสังเคราะห์และชุดข้อมูล UCI ผ่านตัววัดประสิทธิภาพ 3 ตัวคือค่าความแม่นยำ ค่ารีคอลและ ค่าคะแนน F1 ที่ดีขึ้น การทดสอบ Wilcoxon ถูกใช้เพื่อยืนยันประสิทธิภาพที่ถูกปรับปรุงแล้ว

ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	ลายมือชื่อนิสิต	.....
		ลายมือชื่อ อ.ที่ปรึกษาหลัก	.....
สาขาวิชา	คณิตศาสตร์ประยุกต์และวิทยาการคณนา	ลายมือชื่อ อ.ที่ปรึกษาร่วม	.....
ปีการศึกษา	2565		

6270077823 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS : MASSRATIO VARIANCE SCORE / STANDARD CLASSIFIER / IMBALANCED PROBLEM / CLASSIFICATION

PIBOON POLVIMOLTHAM : MASS RATIO VARIANCE MAJORITY CLEANSING AND MINORITY OVERSAMPLING TECHNIQUE FOR CLASS IMBALANCED. ADVISOR : ASSOC. PROF. KRUNG SINAPIROMSARAN, Ph.D., 57 pp.

A sampling method is one of the basic methods to deal with an imbalance problem appearing in machine learning. A dataset having an imbalance problem has a noticeably skewed distribution among different classes. There are three types of sampling techniques to solve this problem by balancing class distributions, undersampling technique, oversampling technique, and combined sampling technique. In this research, the mass ratio variance scores of each data point of the same class are computed and used to remove noise from a majority class and synthesise instances from a minority class. The results of this proposed sampling technique improve recall over standard classifiers: a decision tree, a random forest, Linear SVM, and MLP on all synthesised datasets. Performances are reported on synthesised datasets and UCI datasets via three measures: Precision, Recall, and F1-score. Moreover, Wilcoxon signed-rank tests are used to confirm the improved performance.

Department	: Mathematics and Computer Science	Student's Signature	.....
		Advisor's Signature	.....
Field of Study	: Applied Mathematics and Computational Science	Co-advisor's Signature	.....
Academic Year	: 2022		

ACKNOWLEDGEMENTS

I want to thank Associate Professor Dr Krung Sinapiromsaran my advisor for driving me into the field of machine learning and giving me much knowledge about many topics in this field. When I have an issue in research, my advisor always helps and gives me ideas. He always cheer me up and provided me with motivation. I also receive other notions not only about the research but also about how to handle a problem well and think outside the box. I really appreciate my advisor who loves to work, it affects me the inspiration to work.

I would like to thank every professor at the department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, which has given me the knowledge and the opportunity to complete my graduate study.

I would like to thank myself for deciding to pursue a master's degree. It improves my skill in academic.

CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

This chapter starts with the motivation of machine learning and tasks in Section 1.1. Section 1.2 describes supervised learning for machine learning. One of the tasks is classification. It is the problem to identify the class of instances from historical data. Then this chapter also introduces class imbalance problems in Section 1.3 and its challenge. Section 1.4 presents a review of class imbalance approaches to handling imbalanced datasets. Following another part of the classification process is a review of classification algorithms. Moreover, Section 1.5 reports the performance metrics for a class imbalance dataset. It contains an overview of confusion metrics and reporting measures; accuracy, recall, precision, and F1-Score. Section 1.6 is the motivation of this thesis. Finally, the last section of this chapter is a summary of this thesis.

1.1 Machine learning

Machine learning can be seen in daily life. It is the task that let a machine learns from past experience to apply to future situation. A machine learning model is built using historical data via a machine learning algorithm. This made it possible to adapt the program via learning. In 1959, Arthur Samuel published the research "Some Studies in Machine Learning Using the Game of Checkers". This paper demonstrates how machine can learn to play a checker by Robert Nealey who is a self-proclaimed checkers master. He played the game on an IBM 7094 computer in 1962 and he lost to the computer. This exhibits the good performance of the machine on playing game. Moreover, machine learning is an important component of the growing field of data science. It basically uses statistical methods and computer science algorithms to generate a model or a classifier.

There are many applications of machine learning used in daily life, for example, speech recognition, computer vision, customer service, etc. There are three general topics in machine learning which are categorised by the different types of training data and learning process of a program, supervised machine learning, unsupervised machine learning and reinforcement machine learning. Figure 1.1 shows the categories of Machine learning

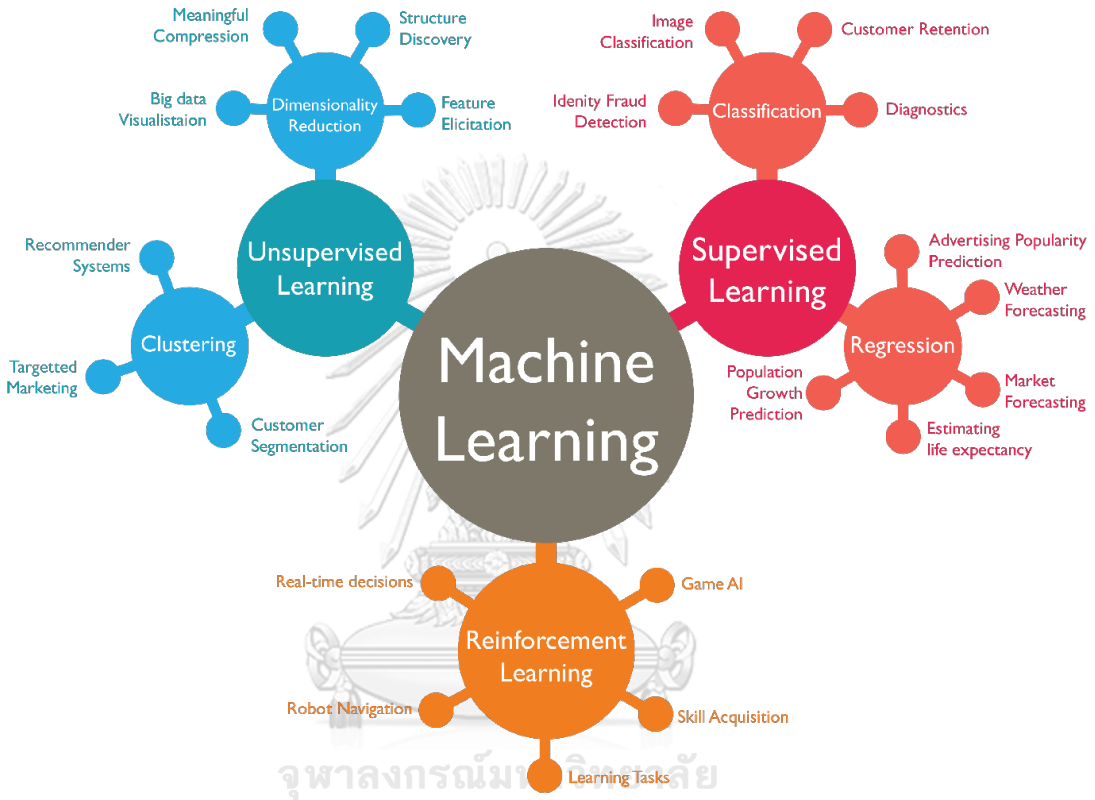


Figure 1.1: Categories of Machine learning
Source: https://www.linkedin.com/pulse/business-intelligence-its-relationship-big-data-geekstyle

1.2 Supervised learning

Supervised learning is a type of machine learning in which machines learn from "labelled" data to classify data and predict outcomes accurately. The labelled data means that input data is already tagged with the target output. In supervised learning, a label or a class or a category of training data is often referred to as a target. The training data provided to the machine works as the supervisor that teaches the machine to learn and then predict the future instance correctly. It applies the same concept as a student

learning under the supervision of the teacher. The process of supervised learning is shown in Figure 1.2. A model is trained using a given labelled dataset where the model learns how target is associated with input data. Once the training process is completed, the model is evaluated based on the test data which guarantees to be difference from training data.

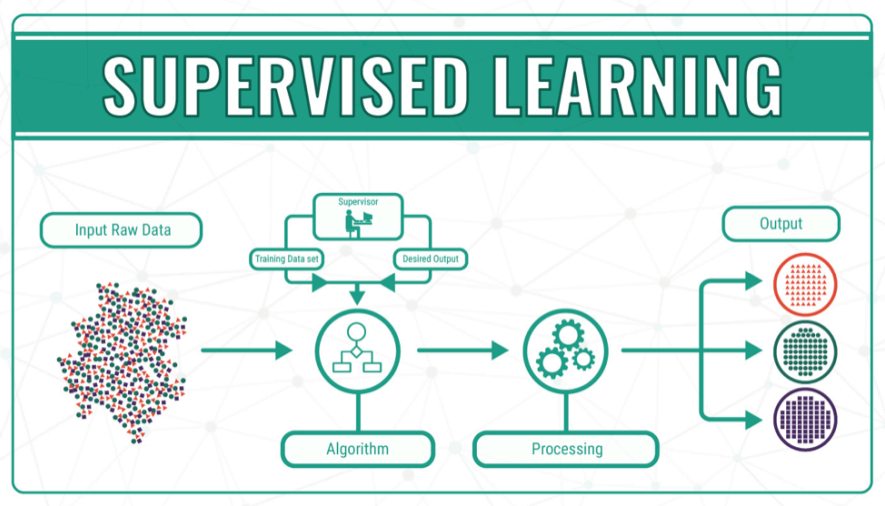


Figure 1.2: Illustrate the process of supervised learning
Source: https://www.crayondata.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/

Classification and regression are two additional subcategories of supervised learning problems that predict discrete and continuous target outputs, respectively. Figure 1.3 illustrates the difference between classification and regression algorithms. Instances in the input dataset may include demographic data such as marital status, gender, age, etc. In contrast with classification, regression can be predicted real values of output such as weather forecasting, market trends, etc. Regression is also suitable to learn a relationship between the input variable and the output variable.

1.3 Classification

Classification is a process of categorising a given data into classes. It is performed on both structured and unstructured data. The difference between these two is that

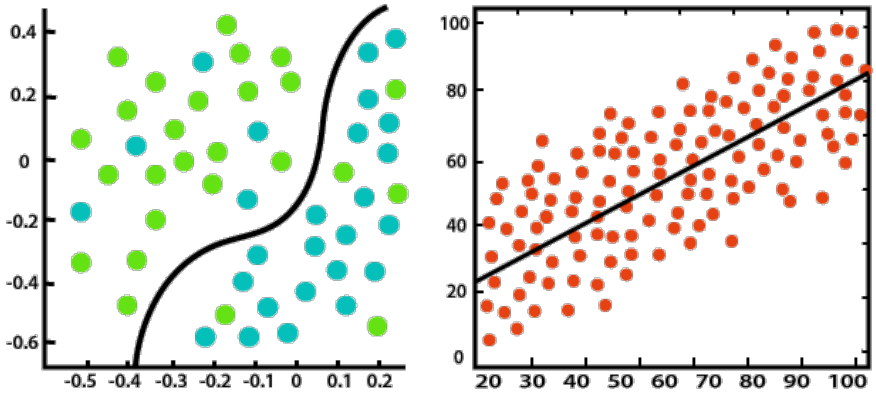


Figure 1.3: Illustrates classification versus regression
Source: https://www.javatpoint.com/regression-vs-classification-in-machine-learning

structured data is highly-organised and formatted as relational databases and unstructured data has no predefined format, making it much more difficult to collect, process, and analyse. The main goal of classification is to identify the target of the new data. To achieve the goal, it needs a classification model to predict a target of new data. Furthermore, classification also applies to many technologies in daily life such as speech recognition, face detection, handwriting recognition, document classification, etc. The process of classification is shown in Figure 1.4. It starts with the learning phase that a classification algorithm builds a classification model or a classifier from training data. The generated classifier must be evaluated for efficiency before use. The remaining given data which does not include the training data is used to evaluate the performance of a built classifier.

There are three types of classification which are categorised by the distinction of the target variable. First, a binary classification refers to classification problems having only two class labels. Generally, one is considered the normal instance and the other is considered to be the abnormal instance. Second, a multi-class classification, this type of classification deals with more than two labels of a target. There can also be a huge number of labels like predicting a picture as to how closely it might belong to one out of the tens of thousands of the faces of the recognition system. Third, a multi-label classification refers

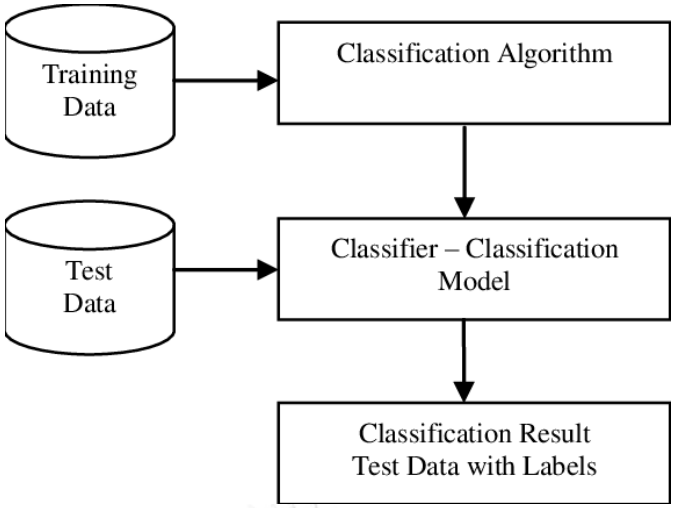


Figure 1.4: Components in the classification process

to those specific classification tasks where two or more specific class labels are assigned to a single instance. A basic example can be photo classification where a single photo can have multiple objects in it like a dog or an apple etc. The main difference is the ability to predict multiple labels and not just one. For more clarity, Table 1.1 shows examples of binary classification, multi-class classification and multi-label classification using a circle, a rectangular, a triangle to represent different class labels.

Instances	Binary targets	Multi-class targets	Multi-label targets
$\mathbf{X}_1$	○	○	○□
$\mathbf{X}_2$	□	□	○□△
$\mathbf{X}_3$	□	△	○

Table 1.1: Examples of binary classification output, multi-class classification output, and multi-label classification output

There is one special problem in a classification of machine learning which differs from the above. It is the classification having different portions of the number of each class and can be either binary or multi-class. This classification is called an imbalanced classification. It refers to this task where the number of instances in each class is unequally

distributed. An imbalanced classification task is a problem in that the majority class of the training data is a normal class type and the rest belongs to the abnormal class. The minority class is the class which has the lowest portion of instances in the dataset which is more important to predict correctly. The imbalanced classification will be discussed in the next section.

1.3.1 Classification algorithms

In classifying a dataset it can be done by finding patterns or general forms of the dataset. The way to find these patterns is to use an algorithm which is called the classification algorithm. There are many types of classification algorithms and each algorithm has a different mechanism to build a classifier. The purpose of the classification algorithm is to build a classifier that predicts correct labels for all instances in the dataset. The process of the classification algorithm is to build a model to capture a pattern from historical data or training data. Each classification algorithm has different ways to use the information from data and they also have their advantages and disadvantages. The following subsections discuss four popular classification algorithms.

1.3.1.1 Decision Tree induction

A decision tree induction is a supervised learning algorithm to build a tree-like structure called a decision tree that can solve regression and classification. The structure of a decision tree is organised from the root to leaves connecting via branches. At the top node of the tree which is called the root node is labelled by the attribute connected to children nodes via branches as the conditions of attribute values. Each leaf node is labelled by the target class. The decision tree induction builds a root node by computing an impurity measure of all attributes from training data and select the attribute with the best impurity measure. Then training data is split to different subsets by the selected attribute using condition assigned to each branch. Each subset data is recursively passed

to the decision tree induction to build a subtree. The stopping criteria is used to stop this tree building process, see an example of a decision tree in Figure 1.5. All of the above processes are called the decision tree induction. Therefore, the training model from the decision tree induction is in the form of a tree-like structure. Then to make predictions, every new instance starts in the root node and moves along the branches until it reaches a leaf node where no further branching is possible.

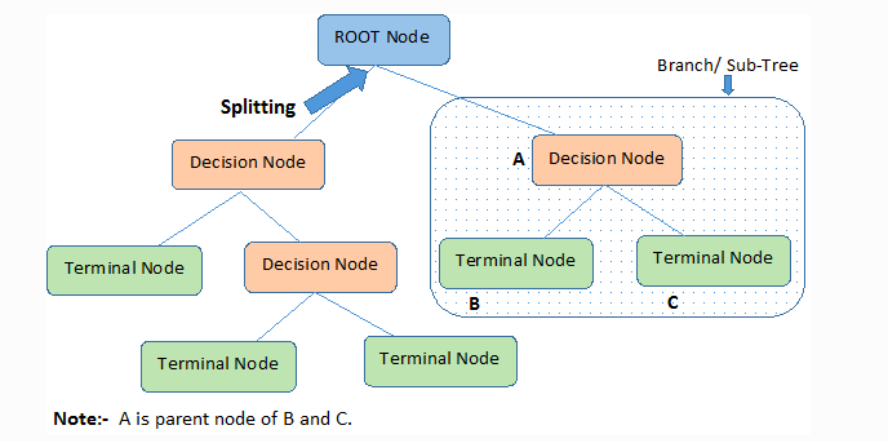


Figure 1.5: Decision tree
Source: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

1.3.1.2 Random Forest algorithm

A random forest classifier was invented by Leo Breiman in 2001 [1]. The algorithm to build a random forest is a supervised learning algorithm which builds a collection of various decision trees. Instead of relying on one decision tree, a random forest collects votes from different trees. Each tree in a forest is independently trained on different training data with different attribute subsets. Different training data for building each tree come from bootstrap subsets of the training data.

Using different decision trees to perform prediction is known as the bagging method which was invented by Breiman in 1996 [2]. The bagging method is designed as the bootstrap aggregation method. It is the method that performs bootstrapping on training data to generate representative for the original data distribution. While bootstrapping

is a random sampling with replacement from data, instances that are not selected from the bootstrapping step can be referred to as out-of-bag (OOB) instances. They can be used as test instances for evaluating the prediction error of each tree in a forest yielding the so-called out-of-bag error [3]. Out-of-bag error is frequently used to evaluate random forest prediction performance. One advantage of the out-of-bag error is that the entire original sample is used for building decision trees and error estimation. The use of the out-of-bag error saves memory and computation time, especially when dealing with large data. These factors may explain why the out-of-bag error is frequently used in random forest error estimation.

Another feature in a random forest algorithm is called feature selection. It is the method that can be used to rank the importance of variables which means each tree in a random forest can calculate the importance of a variable according to its ability to increase the pureness of subsets from a dataset.

Figure 1.6 shows the diagram of the random forest classifier. It starts with two partitions of data into the training data and test data and then applies random forest into the training data. Now the random forest creates multiple decision trees based on the training data and to get the accuracy on testing data by taking the majority vote from all decision tree output.

1.3.1.3 Support Vector Machine algorithm

Support vector machine (SVM) is a classifier which is widely used in pattern recognition, classification and regression. It frequently used in classification and particularly used in noisy and complex domains. SVM was invented by Vladimir Vapnik and Alexey Chervonenkis in 1963. It is a binary classification model that uses a hyperplane to divide input space into two regions. The objective of the SVM algorithm is to find the optimal hyperplane in N-dimensional space that distinctly classifies instances in a dataset. Note

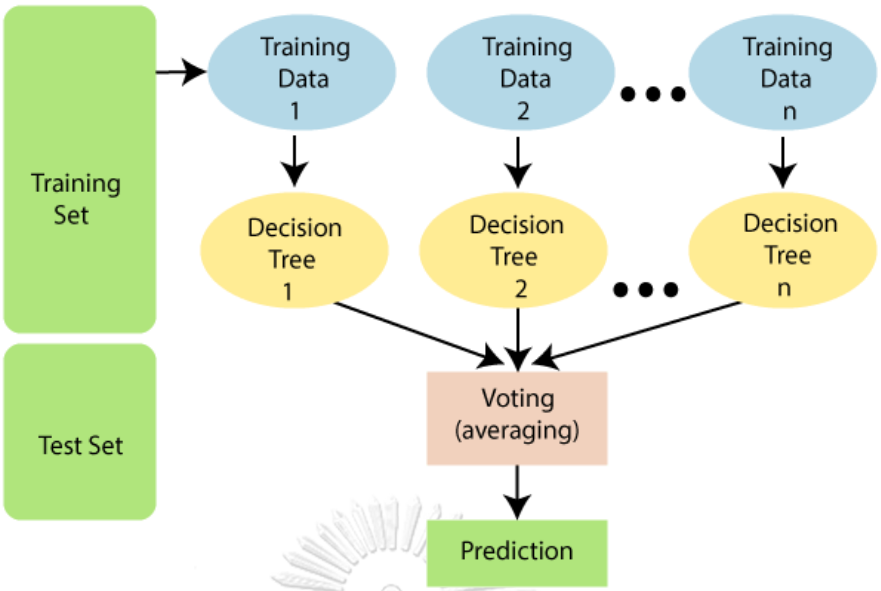


Figure 1.6: Random forest algorithm
Source: https://www.javatpoint.com/machine-learning-random-forest-algorithm

that N is the number of features. A hyperplane is a decision boundary that split a space into two regions. An instance falling on either side of the hyperplane is labelled to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3. Two examples of hyperplanes in 2-dimensional and 3-dimensional are shown in Figure 1.7.

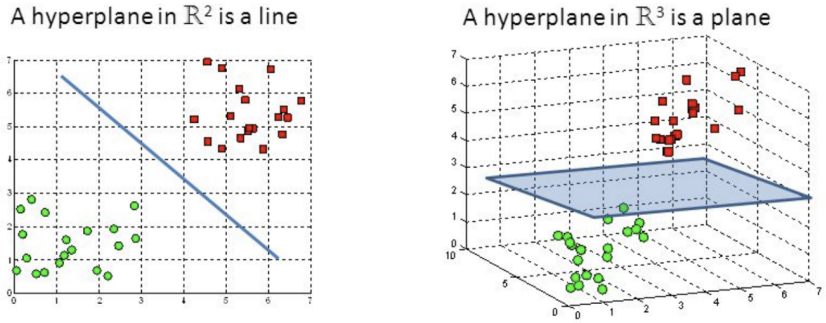
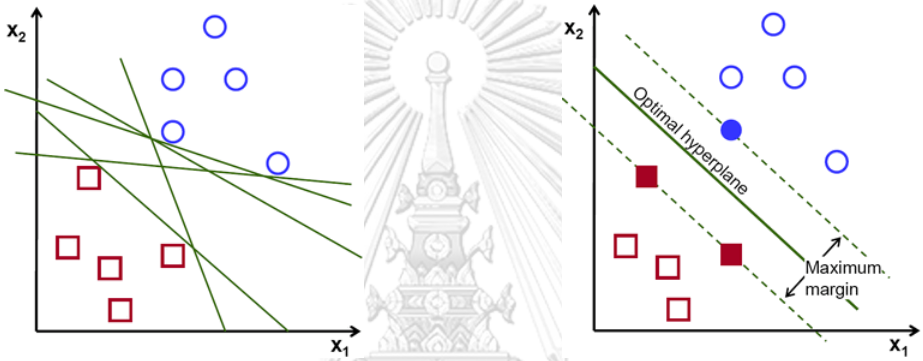


Figure 1.7: 2-dimensional and 3-dimensional hyperplanes
Source: https://www.ques10.com/p/41200/support-vector-machine-1/

In separating two classes of instances, there are many possible hyperplanes that can be chosen. For example in Figure 1.8(a), the green lines are valid splitting hyperplanes of the dataset. To find the optimal hyperplane, a buffer zone, called a margin, around the hyperplane was defined and formulated as an optimization problem to maximise this margin. The larger margin of the hyperplane, the better separation of two classes. The points that define the margin are called support vectors. As in Figure 1.8(b), two filled-red instances and a filled-blue instance are the support vectors for this model. In addition, deleting these support vectors will change the position of the optimal hyperplane.



(a) A possible hyperplanes **(b)** The optimal hyperplane and maximum margin

Figure 1.8: Example of support vectors
Source: https://www.ques10.com/p/41200/support-vector-machine-1/

Generally, high-dimensional or real-world datasets are not perfectly separable by any hyperplane in the space. Rarely all instances from one class will be on one side and those from the other class be on the other side. An SVM algorithm constructs the best separating hyperplane by introducing a penalty for instances that fall on the wrong side of its class. Note that the margin size governs a trade-off between correctly classifying training data and generalising to future data. A wide margin means more misclassified training instances while SVM will be more suitable to split future instances. Moreover, if instances in the dataset can not be separated by any linear hyperplane from the current dimensions then the remedy is to add another dimension. This technique is called kernel trick.

The objective of SVM kernel trick is to use a predetermined nonlinear mapping method to map the input vectors x into a high-dimensional feature space Z. In this space, an optimal separating hyperplane is constructed. Figure 1.9 shows the example of the mapping input vector into high-dimensional feature space. A kernel usually uses a set of mathematical functions. The function of the kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be of different types such as linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

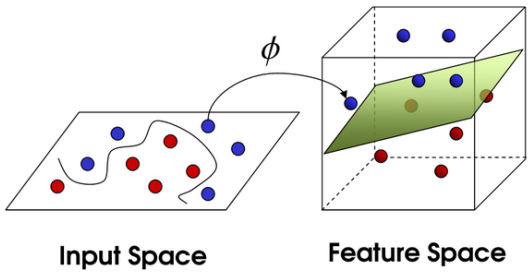
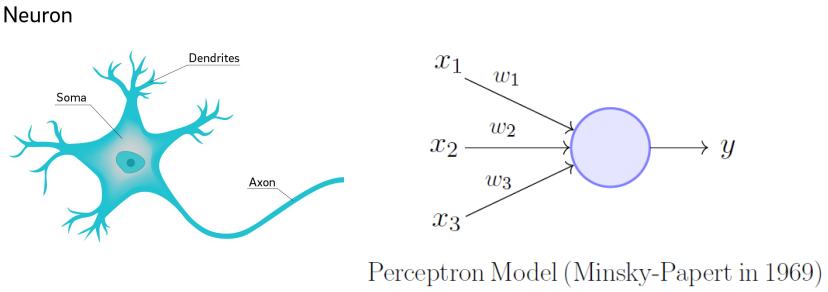


Figure 1.9: The mapping input vector into high-dimensional feature space
Source:https://learnopencv.com/wp-content/uploads/2020/10/svm_kernel_trick.png

1.3.1.4 A neural network algorithm

A neural network (NN) is sometime referred as an artificial neural network (ANN). It is a classifier in machine learning which is a sub-field of artificial intelligence (AI). It is a computer system that has been inspired by the biological neuron system which occurs in a human brain. A neural network is used in almost every machine learning application because of its reliability and mathematical power such as an image recognition. The learning process in human brains consists of many separate neurons, each neuron in the brain receiving its chemical input and communicating to each other by sending signals through its dendrites. Figure 1.10(a) shows the visual of biological neurons and Figure 1.10(b) shows the visuals of artificial neurons. In the same way, the learning process of a neural network is composed of many processing components or neurons which are

commonly called nodes. It is organised as a directed graph which contains nodes and edges connecting each node.



(a) Biological neuron **(b)** Artificial neuron

Figure 1.10: Two visualizations of a neuron
Source(a): https://scx1.b-cdn.net/csz/news/800a/2018/2-whyareneuron.jpg
Source(b): https://miro.medium.com/max/786/1*-JtN9TWuoZMz7z9QKbT85A.png

In addition, a neural network is defined by its weights treating as parameters. These weights will be fine-tune via incoming data. The weight on each arc will multiply with the input value. All inputs with weight multiplication will be sum and passing to the activation function to generate an output. This output is then passed to the next layer of neurons which use them as inputs and so it continues until every layer of neurons has been considered and the terminal neurons have received their inputs. Then the terminal neurons output the final result for the model.

There are several kinds of neural network models. A multilayer perceptron (MLP) classifier is one kind of neural networks. It is designed to fully connecting all nodes from the lower layer to the ones from the higher layer. MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. The visual of MLP with three layers is shown in Figure 1.11. Moreover, MLP utilises a supervised learning technique called backpropagation for weight adjustment. The backpropagation technique is an important method of neural networks. It is a key trigger for renewed interest in neural networks from Werbos's (1975). Backpropagation distributes the error term back

up through the layers, by modifying the weights at each node. Through backpropagation, each time the output is labelled as an error during the supervised training phase, the information is sent backwards. Each weight is updated proportionally according to their contributions to the output. Hence, the error is used to recalibrate the weight of the neural network's unit connections to take into account the difference between the desired outcome and the actual one. After finish training, the neural network will "learn" by minimizing the chance of errors and unwanted results.

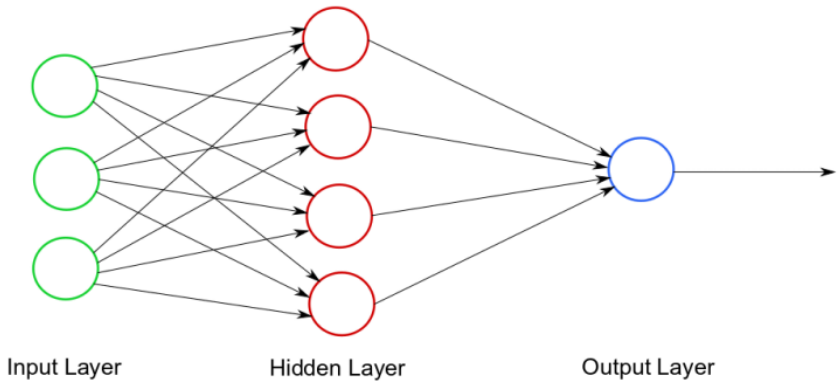


Figure 1.11: A visualise of a simple neural network
Source: https://impicode.com/wp-content/uploads/sites/2/2020/12/neural_network_layers_diagram..png

1.4 Class imbalance problem

In the last few years, there have been major changes and evolution in data collection. Many systems and devices are now supplied data to the system. This leads to an increase in the size of data. The classification of data is also affected by this problem. It becomes more difficult to classify enormous data and imbalance of data. An imbalanced classification is an example of a classification where the distribution of examples across the known classes is biased or skewed. It becomes one of the important topics in classification from machine learning. A class imbalanced problem [4] is a problem of building a classifier in the presence of underrepresented class instances or highly skewed class distributions. This occurs when the number of instances representing an important class is much smaller than those from other classes. In general, the class with the smallest

number of instances is referred to as the minority class or the positive class while another class is referred to as the majority class or the negative class. In addition, Figure 1.12 shows a graphical representation of an imbalanced dataset in which the blue points are instances from the majority class and the orange points are instances from the minority class. The minority class is important than the majority one. It is always labelled as "+" or positive and another class or the majority class always labelled as "-" or negative.

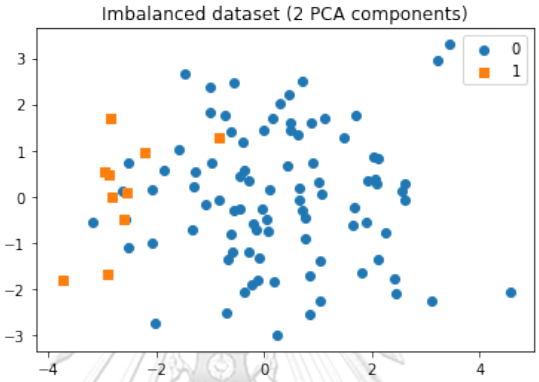


Figure 1.12: A graphical representation of an imbalanced binary data
Source: https://machinelearningmastery.com/wp-content/uploads/2019/10/Scatter-Plot-of-Binary-Classification-Dataset-With-1-to-100-Class-Distribution.png

The main purpose of classification on this problem is to identify minority instances as accurately as possible. However, most of the machine learning algorithms currently in use were designed around the assumption of a uniform distribution over all classes. The accuracy can not be used as the effective measure for the model performance since the proportion of minorities is too small. Some models will treat minorities as noise and ignore instances from this class. This presents a challenging algorithm to build a classifier to recognize minority class. The problem is more sensitive to classification errors for the minority class than the majority class.

In real world applications such as fraud transactions in the fraud detection [5], the fraud transaction is seldom appeared but it is more important than the regular transaction. For ailing patients in the medical diagnosis [6], patients with disease are more serious to be detected than patients without disease and the number of disease patients is

smaller than the number of normal patients. For default loans in the credit approval [7], applicants who cannot make payment in time are more significant than applications who make payments on time. In addition, a class imbalance problem in the medical diagnosis is to detect and diagnose the patterns of certain diseases within patient electronic healthcare records. It is normal that some life threatening diseases are rare among patients. The misclassification of these cases can lead to the patient's death so the ailing patients that identify as healthy should not occur, i.e. the number of false negative patients should be small. However, when a standard classifier was applied on an imbalanced dataset, a minority instance tends to be misclassified because a classifier is more focused on classification of majority instances while ignoring or misclassifying minority instances due to its tiny portion.

As previously stated, there are two components in the learning phase of the classification process which can be changed to deal with class imbalance. The first component is the data component, the method may change the distribution of the data from imbalanced to balanced. Then the standard classifier should be able to predict the target class. The second component is the classification algorithm, the process inside the classification algorithm should be changed to generate a model that recognizes the minority class. These two approaches will be reviewed next.

1.4.1 Techniques to solve a class imbalance problem

There are three main methodologies to deal with an imbalance problem. Each of these methodologies has their own advantages and disadvantages. First, a data-level methodology[8, 9, 10, 11] balances the distribution of class instances by resampling the distribution. It allows a user to apply his/her preferred classifier because this method concentrates on modifying the training data, which removes or adds data within classes to balance the data. Data can be balanced via two techniques, undersampling or oversampling techniques. There are many techniques in this approach such as an oversampling

technique[12, 13, 14] which synthesises instances randomly from the minority class while ignore those from majority class, or an undersampling technique[15, 16] which randomly removed instances from the majority class to expand the minority region of instances in the minority class or the combines of both an oversampling and undersampling technique[17]. Second, an algorithmic-level methodology enhances or reimplements the classification algorithms to be more resilient to noise while successfully handling minority instances[18, 19, 20, 21, 22]. An algorithmic-level methodology efficiently deals with the class imbalanced problem using the original data that is available without any modification. Unlike data level techniques, the working area of these approaches is the algorithm's internal structure. However, algorithmic-level approaches have the disadvantage of being less efficient when applied to a high ratio of an imbalanced class dataset. Third, a hybrid methodology combines both the data-level methodology and the algorithmic-level methodology such as Adaboost[23], Boosting[24], Bagging[2], etc.

This thesis proposes the data level technique to deal with an imbalanced class in the classification. The idea is to remove some majority instances which are placed near the minority region and synthesises more minority instances near minority that is placed far away from other minority instances to make the dataset "relatively balanced". There are different situations that require the use of undersampling or oversampling techniques. An undersampling technique is very effective when dealing with datasets with a low proportion of class imbalance, whereas an oversampling technique is very effective when dealing with datasets with a high proportion of data imbalance. An oversampling technique also increases the size of the training data due to new synthesised instance, which causes overfitting and a long learning time. In the opposite way, an undersampling technique is more useful in cases when the less calculation time is needed because this technique reduces the size of the dataset. Additionally, the hybrid approach combines both undersampling and oversampling techniques, and it is used when there is no pattern to directly increase or decrease the dataset. Figure 1.13 shows the concept of undersampling and oversampling

techniques.

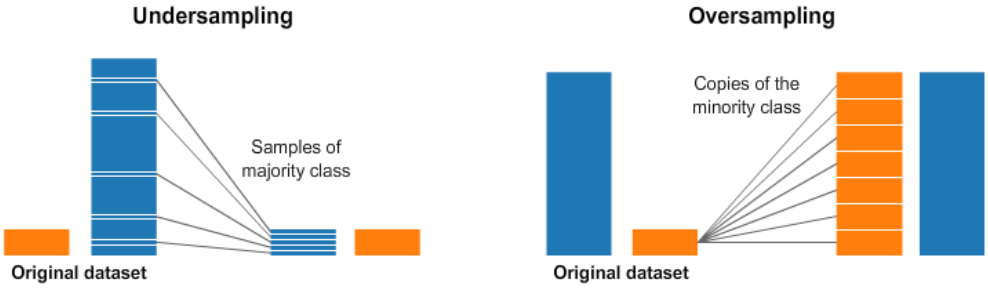


Figure 1.13: An overview of resampling techniques
Source: https://raw.githubusercontent.com/rafjaa/machine_learning_fecib/master/src/static/img/resampling.png

An undersampling algorithm reduces the amount of data that the model must learn by focusing on removing instances from the majority class. Currently, there are many undersampling strategies including DBMUTE 2017 and MUTE 2011, but the random undersampling algorithm (RUS) is the most straightforward approach that remove minority instances arbitrarily and without any constraints. One of intelligent approaches toward undersampling is tomek-link[25]. It groups the borderline minority instance with the closest majority instance before eliminating those majority instances using 1-nearest-neighbor which clears border instances from the majority class and makes a classifier to partition class regions easily.

In contrast to undersampling, an oversampling algorithm increases the number of minority instances. The most basic method is the random oversampling algorithm (ROS), which duplicates instances from the minority class at random. The synthetic minority oversampling technique (SMOTE)[8] is a popular oversampling technique for expanding the region of the minority class. SMOTE operates within the current feature space to produce synthetic instances. The new synthetic instances are extracted from interpolation, so the original dataset still has significance. This makes SMOTE also avoid the issue of overfitting when increasing minority class instances.

1.5 Performance measures

In classification, each classifier must evaluate its performance to ensure that it will perform better on future data. Accuracy is the most common benchmark in the classification problem without imbalanced issue. Accuracy is computed by the number of true predicted instances both positive and negative divided by the number of all instances in the dataset. The formula of accuracy is also shown in Equation 1.1 below. However, it is not appropriate when the dataset is imbalanced. Because a classifier can achieve high accuracy by just predicting entire instances as the majority class which misclassifies all instances from the minority class. Moreover, the imbalanced ratio (IR) used to indicates the proportion of minority instances represent in the dataset. The imbalanced ratio is the division of the number of minorities by the total number of instances, see Equation 1.2. A low IR value indicates that there are few minority instances in the data which mean it is highly imbalanced, whereas a high IR value indicates that the data is more balanced.

$$Accuracy = \frac{\text{True positive}}{\text{number of all instances}} \tag{1.1}$$

$$IR = \frac{\text{the number of minorities}}{\text{the total number of instances}} \tag{1.2}$$

The confusion matrix defines the base for binary classification performance measures from a classifier. Most of the performance measures are derived from the confusion matrix, i.e. accuracy, misclassification rate, precision, recall and so on. The confusion matrix shows how many accurate and inaccurate predictions a classifier made by counting the number of true and false instances from each class prediction. The confusion matrix consists of 4 quadrants in which each row in the confusion matrix represents the instances in an actual class while each column represents the instances predicted by a classifier. The visualisation of the confusion matrix is shown in Table 1.2. The true positive (TP)

is the count of true positive instances: the actual class is positive and the predicted class is also positive. The true negative(TN) is the count of true negative instances: the actual class is negative and the predicted class is negative. The false positive(FP) is the count of false positive instances: the actual class is negative but the predicted class is positive. The false negative(FN) is the count of false negative instances: the actual class is positive but the predicted class is negative.

Confusion matrix	**Actual Positive**	**Actual Negative**
Predicted Positive	TP: True Positive	FP: False Positive
Predicted Negative	FN: False Negative	TN: True Negative

Table 1.2: Confusion matrix

An importance of each measures is not equal for a class imbalanced problem. TP is the most importance which stand for the number of correct predicted instances of the positive class, FP is the number of negative instances that are predicted as positive and FN is the number of positive instances that are predicted as negative which should be very low for a class imbalance problem. These three measures are more importance than TN due to the portion of the majority class. So recall, see Equation 1.4, will be more emphasised than precision, see Equation 1.3. Nevertheless, to incorporate both measures, F1-score is used as the harmonic mean of precision and recall, see Equation 1.5.

$$Precision = \frac{\text{True positive}}{\text{True positive+False positive}} \quad (1.3)$$

$$Recall = \frac{\text{True positive}}{\text{True positive+False negative}} \quad (1.4)$$

$$F1 - score = 2 \times \frac{\text{Precision}}{\text{Precision+Recall}} \quad (1.5)$$

1.6 Our thesis

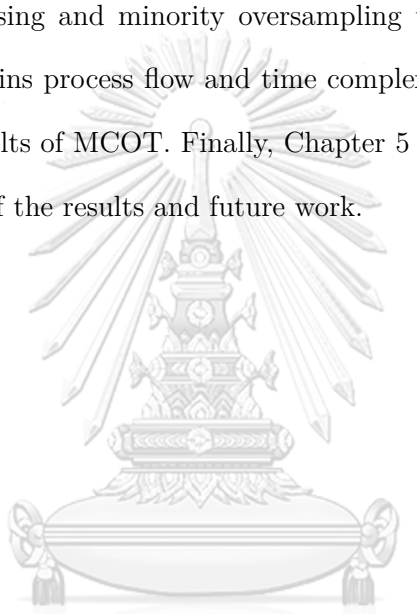
Our thesis motivation is to improve the performance of any classifier by reducing a number of misclassified instances. Typically, a minority instance that has been incorrectly classified is located farther away from normal minority instances or other minority instances. Additionally, it should synthesise a few minority instances that are close to these minority instances. Moreover, a classifier should be assisted in identifying them by an algorithm that removes some neighboring majority instances from the overlapped region. This can be done automatically via the use of MOF or the Mass-ratio variance based outlier factor[26]. The MOF algorithm generates an MOF score for each instance based on its density. It gives a very high score to an outlier. Therefore, the mass ratio variances from majority instances can be used to cleanse majority instances and the mass ratio variances from minority instances will be used for oversampling.

In this thesis, the MOF algorithm was used in this thesis to identify abnormal instances in both majority and minority classes. An abnormal instance from the majority class is treated as noise and is removed to clear the decision boundary of a dataset, whereas abnormal instances from the minority class are packed with synthesised minority instances. Performance of the proposed method was evaluate using four standard classifiers which are SVM, decision tree, random forest, neural network. They will be run

on synthesised and UCI datasets producing precision, recall and F1-score. Finally, the Wilcoxon signed-rank test will be used to demonstrate the effectiveness of the proposed method for unseen instances.

1.7 Summary of remain chapters

This thesis is divided into five chapters. Chapter 2 contains the background knowledge about the thesis. Chapter 3 is the proposed method and explains the mass-ratio variance majority cleansing and minority oversampling technique or the MCOT algorithm. There also contains process flow and time complexity analysis. Chapter 4 shows the experiment and results of MCOT. Finally, Chapter 5 is presented a summary of this thesis and conclusions of the results and future work.



CHAPTER II

BACKGROUND KNOWLEDGE

This chapter covers background knowledge related to the proposed method; a new resampling technique for dealing with a class imbalance problem. It begins with an overview of the mass-ratio-variance based outlier factor (MOF) which is an outlier score for a finite dataset in the Euclidean space. The definition that is used to compute MOF is also included. Moreover, this section also includes the MOF flowchart, the MOF algorithm analysis and the MOF pseudocode. The algorithm from this thesis applies the data cleansing step on majority instances and applies the oversampling techniques to minority instances concentrating on ones with high MOF scores. In addition, an overview of the oversampling techniques SMOTE and how to generate synthesised instances are also explained in this chapter. The last section of this chapter is a summary of the nonparametric test; the Wilcoxon-signed rank test which is used to confirm the finding.

2.1 Mass-ratio-variance based Outlier Factor (MOF)

Mass-ratio-variance based outlier factor[26] (MOF) is one of the parameter-free outlier scores for the outlier detection. The outlier detection identifies instances which appear to be different from other instances in a dataset[27]. An outlier may cause by a change in a system, a human error, or simply through natural deviations in populations[28]. An outlier is defined as an instance that significant differs from others. It is normally located far from others or surrounded by a few instances from another class in the Euclidean space. Many popular unsupervised outlier scoring algorithms[29, 30, 31] assign scores to instances in a dataset called factors and requires some parameters. With an improper setting of these parameters, the detection may not be reliable. For density-based algo-

rithms, k-nearest neighbours is globally used requiring the setting of parameter k. It is normally assigned as the average distance of the k-nearest neighbours as proposed in the fast outlier detection in the high dimensional space[32]. The change of the value of k leads to change in rank of all instances. Hence, a parameter is very sensitive to instances ranking.

In contrast, MOF is a parameter-free density-based outlier score which is defined as the variance of the mass-ratio distribution from other instances. Note that the density is defined as the ratio of mass to volume. With the fixed volume, the ratio of the density between two instances is the same as the ratio of the mass for both. The MOF algorithm assigns a single score to each instance by the variance of the mass-ratio distribution generated from all other instances in the dataset. According to the experiment, the variance of the mass-ratio distribution of outliers is greater than any normal instance which makes the score of an outlier very high and low when it is a normal instance.

2.1.1 Definition

This section states the definitions of MOF.

Definition 2.1 (Distance between x and y)**.** Given a dataset $D \subseteq \mathbb{R}^d$, the Euclidean distance between instance $x = (x_1,...,x_d) \in D$ and instance $y = (y_1,...,y_d) \in D$ denoted as $d(x, y)$ is defined as

$$d(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Definition 2.2 (Neighbours of x with respect to radius r)**.** Given a dataset $D \subseteq \mathbb{R}^d$, the set of all instances within neighbourhood of instances $x \in D$ with respect to the radius r is defined as the set of instances that lies within the ball centred at instance x with the radius r:

$$N(x, r) = \{z \in D | d(x, z) \leq r\}$$

Definition 2.3 (The mass-ratio of instance y with respect to instance x)**.** Given a dataset $D \subseteq \mathbb{R}^d$ and instance $y \in D$, for any instance $x \neq\in D$. The mass-ratio of instance y with respect to instance x is defined as

$$mr_x(y) = \frac{|N(y, d(x, y))|}{|N(x, d(x, y))|}$$

Definition 2.3 shows the computation of the mass ratio of each instance in the dataset. Assume that instance x is an outlier, the denominator will be a small number, causing the high mass-ratio values except the one closest to x. This mass-ratio will be close to one if this instance x is among other instances in a dataset.

Definition 2.4 (MOF of instance x)**.** Given a dataset $D \subseteq \mathbb{R}^d$ and for instance $x \in D$, $\bar{mr}_x$ is defined as mean of the mass-ratio distribution of instance x and MOF of instance x is defined as the variance of the mass-ratio distribution of instance x:

$$\bar{mr}_x = \frac{\sum_{i=1, y_i \neq x}^{n} mr_x(y_i)}{n-1}$$

$$MOF(x) = \frac{\sum_{i=1, y_i \neq x}^{n} (mr_x(y_i) - \bar{mr}_x)^2}{n-1}$$

The proposed method used MOF to distinguish between abnormal and normal instances. In the following section, the threshold used to distinguish between abnormal and normal instances will be selected and explained in details.

2.1.2 The MOF algorithm

Figure 2.1 shows the flowchart of the MOF algorithm containing five processes. The first process starts with reading input dataset D. Then the second process is computing distances using Definition 2.1 between pairs of instances having computational

complexity of $O(n^2)$. The third process uses Definition 2.2 to calculate the number of neighbours $N(y, d(x, y))$ and $N(x, d(x, y))$ by counting the number of instances lie in the sphere of the ball between instances x and y in D, So the computational complexity for determining whether instances lie within the neighbourhood of x or y is $O(n^2 \log(n))$. The fourth process uses Definition 2.3 to compute the mass ratio or $mr_x(y)$ which is the ratio of $N(y, d(x, y))$ and $N(x, d(x, y))$ having time complexity of $O(n^2)$. The last process calculates MOFs of all instances which are variances of $mr_x(y)$ from other instances in D using Definition 2.4 which has computational complexity of $O(n^2)$. Therefore, time complexity of the MOF algorithm is $O(n^2 \log(n))$.

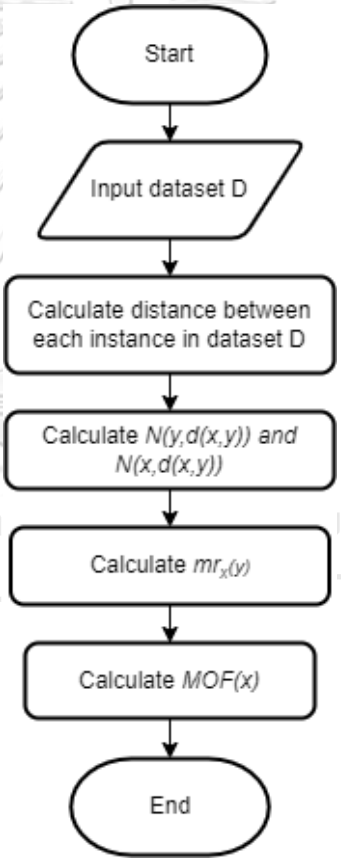


Figure 2.1: Flowchart of MOF algorithm

The following algorithm is the pseudocode which demonstrates the step-by-step of the MOF algorithm.

Algorithm 1 Compute_MOF(X,y)

Input: Array of data X vector of target y
Note:
1. # is used for a line comment
2. pairwise_distances(X) returns the Euclidean distance matrix from a vector array X
3. len(X) returns the number of items in an object X.
4. range(n) returns a series of numbers from 0 to n-1
5. sum(*conditions*) returns the number of instances which match the condition
6. var(X) returns the variance along the specified axis.
7. Variable_name[*conditions* or *index*] is a numpy array represent a group of selected instances
Output: MOF of dataset D

```
# Calculate distance between each instance in dataset D
DistanceMatrix = pairwise_distances(X)
# Calculate N(y,d(x,y)) and N(x,d(x,y)) step
for x in range(len(X)) do
    for y in range(len(X)) do
        #Count the number of neighbours of each instance
        nNeighbour[x,y] = the number of neighbors that is close to x than the distance from x to y
# Calculate mr_x(y) step
for x in range(len(X)) do
    for y in range(len(X)) do
        if y ≠ x then
            mr[x,y] = nNeighbour[y][x]/nNeighbour[x][y]
#Calculate MOF(X) step
MOF = var(mr)
return MOF
End Compute_MOF
```

2.2 Data cleansing process

In machine learning, data cleansing step is a part of data preprocessing step. It is used to weed out irrelevant or incorrect information. Data cleansing not only refers to removing chunks of unnecessary data but it is also often associated with fixing incorrect information within the train-validation-test dataset and reducing duplicates. Data

cleansing step is an important part of any machine learning pipeline. Cleansing data can help algorithm generate the appropriate model.

A cleansing process normally removes noises from a dataset. Noises are defined as instances that are disturbing the decision boundary or appearing in the overlapping regions between two classes. Removing these noises help the classifier to easily determine the decision boundary. The outliers from the majority class should be removed since they interfere with the decision boundary of the classifier for the minority cluster. Most machine learning algorithms, predominantly linear regression models, need to be dealt with outliers, or else the variance of the model would be very high, which further leads to false conclusions by the model.

2.3 Oversampling technique, ROS and SMOTE

An oversampling technique is a technique that will be applied to instances in the minority class. It is the technique that replicates instances to balance the dataset and keeping original information. Furthermore, an oversampling technique can create minority instances using a pre-determined criterion. However, the disadvantage of this technique is that the number of instances can increase significantly which causes the long learning time.

Most basic oversampling technique is the random oversampling technique (ROS) which balances the dataset by selecting minority instances at random and duplicate them. ROS also referred to as the naive sampling technique because this technique randomly selects an instance from the minority class and duplicates it without assuming any constraint about the dataset. Each instance can be selected with a replacement which means that instances from the minority class can be chosen and added to the new balanced dataset multiple times. This method is simple to implement.

Alternatively, an oversampling technique in the Euclidean space create synthetic

instances nearby minorities. In other words, it enlarges the decision region of the minority class by producing artificial data, as opposed to replicating instances. The synthetic minority oversampling technique (SMOTE) is a popular technique for creating synthetic instances. Based on the feature space similarities from minority instances, the SMOTE algorithm generates artificial data that make a classifier recognize instances in favour of the minority class.

SMOTE generates artificial instances primarily between that instance and its nearest neighbour by interpolates values from k-nearest neighbours. In generating synthetic minority instances, SMOTE calculates distances between each minority instance and then finds its nearest neighbour. After multiplying these distances by a random number between 0 and 1, this generates a random line segment between every existing pair of features. As a result, a new instance is consequently created close to the original instance in the dataset. The process was repeated for every minority in the dataset. Figure 2.2 illustrates the process of generating synthetic minority instances using SMOTE. The green instances are the synthetic instances which generate between two minority instances. Furthermore, there are many algorithms which enhanced from SMOTE, and deal with some majority instances during the synthetic process such as borderline-SMOTE[9] and safe-Level-SMOTE[10]

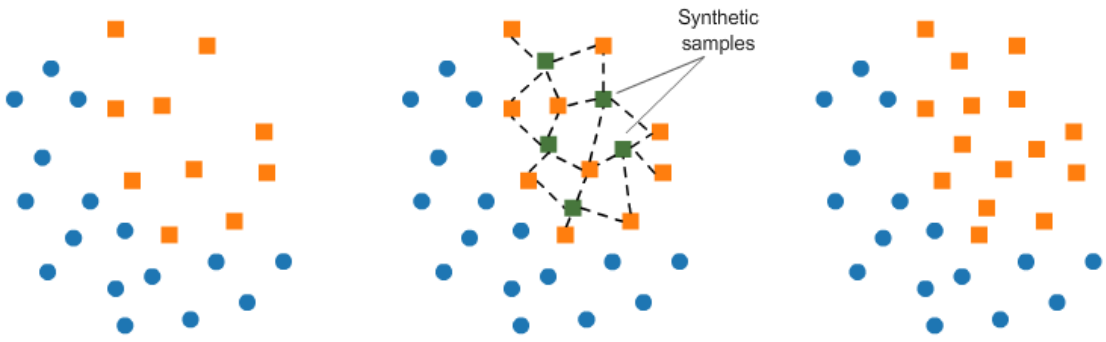


Figure 2.2: Synthesis operation of SMOTE
Source: https://raw.githubusercontent.com/rafjaa/machine_learning_fecib/master/src/static/img/smote.png

2.4 Non-parametric test

A non-parametric test is a statistical method for statistical hypothesis testing that does not require assumptions regarding the underlying population. It does depend on any particular distribution from the population which does not require any parameter. Moreover, non-parametric tests are also known as distribution-free tests because they do not rely on the underlying population. Furthermore, the non-parametric test has the advantage of being simple to understand, having short calculations, and applying to all types of data. However, it still has drawbacks because it uses a distribution-free method, which means that the results may or may not give an accurate response and is less effective than the parametric one.

2.4.1 Wilcoxon sign-rank test

The Wilcoxon signed-rank test was proposed by Frank Wilcoxon in "Individual comparisons by ranking methods"[33]. Wilcoxon signed-rank test is a non-parametric test used to compare two related samples, matched samples, or to conduct a paired difference test of repeated measurements on a single sample to assess whether their population mean ranks differ. In statistics, the term "non-parametric" usually means that the population data does not have a normal distribution. Wilcoxon signed-rank test is the non-parametric analogue to the paired t-test. If the distribution of differences between pairs is severely non-normally distributed, the Wilcoxon signed rank test should be used.

Wilcoxon signed-rank tests come in two different types. First, the Wilcoxon signed rank test compares a sample's median to a hypothetical median. Second, the Wilcoxon matched-pairs signed rank test calculates the difference between each set of matched pairs and then compares the sample against a median using the same steps as the signed-rank test.

The null hypothesis for this test is the difference in the median between pairs of

samples is zero. The Wilcoxon signed-rank test is referred to as the W-statistic which if the paired observations n is greater than 10 then the W-statistic approximates a normal distribution. This test is working by ranking the absolute value of the differences between observations from smallest to largest by giving the rank of one to the smallest difference value, then the next larger difference gets the next rank till the last difference pair. In case of a tie, the average rank is assigned to both. Next, add the ranks of all positive differences in one direction, and then add the ranks of all negative differences in the other direction. The smaller of these two sums is the test statistic W.

CHAPTER III

MASS RATIO VARIANCE MAJORITY CLEANSING AND MINORITY OVERSAMPLING TECHNIQUE (MCOT)

This chapter covers the proposed work and explains its motivation. The proposed method is the mass ratio variance majority cleansing and minority oversampling technique (MCOT), it is the resampling technique which uses MOFs to detect abnormal instances and then apply resampling techniques to them. The motivation for each component of MCOT is explained in the first section. The overall process is demonstrated in the flowchart and the pseudocode. Moreover, they also include some examples to demonstrate the computation of each component. Furthermore, the algorithm analysis is performed.

3.1 MCOT

This thesis proposed the mass ratio variance majority cleansing and minority oversampling technique (MCOT). It uses MOFs to detect abnormal instances which are high for anormaly and low for other instances. Abnormal instances normally lies further away from other instances that cause a classifier to misclassify them. MCOT is an algorithm to help a classifier to recognize abnormal instances and remove some surrounding majority instances within the overlapping region. In addition, MCOT also synthesise a small number of minority instances near given minority instances. It separately computes MOF for each class. An abnormal instance from the majority class is treated as noise which will be removed while abnormal instances from the minority class will be packed with synthesised minority instances.

3.1.1 Mass-ratio-variance score (MOF)

Mass-ratio-variance score is computed from the MOF algorithm. It was originally designed to identify outliers of a static dataset by assigning a high MOF score to an outlier and a low MOF score to a normal instance. In this thesis, The MCOT algorithm use MOF to identify abnormal instances from normal ones. An abnormal instance is an instance which lies further away from other instances in a class. It can be described as an instance which has low density. In other words, it contains a few instances within neighbourhood. The criterion to separate an abnormal instance from a normal one is using a threshold for MOFs.

The criteria to separate abnormal instances from normal instances is called an abnormal threshold. IQR rule which is $Q_3 + 1.5IQR$ is suggested to identify outliers for univariate data distribution. Note that $IQR = Q_3 - Q_1$ where Q_1 is the 25^{th} percentile and Q_3 is the 75^{th} percentile. Nevertheless, to find an appropriate abnormal threshold for MCOT, this thesis uses the decision tree with 50^{th}, 60^{th}, 70^{th}, 80^{th} and 90^{th} percentiles and $Q_3+1.5IQR$. Ignoring $Q_1-1.5IQR$ since MOFs are always positive. The experiment used ten combinations of synthesised datasets and two real world datasets see Table 3.1 for additional details. The synthesised datasets generates using combination of following setting; (IR, c, d, N) where IR is the imbalanced ratio $\in \{0.1, 0.2, 0.5\}$, c is the number of clusters $\in \{2, 3, 5\}$, d is the number of features $\in \{2, 3, 5\}$, and N is the number of instances $\in \{100, 300\}$. Each combination will be synthesised for 30 datasets. Two real world datasets which are “Ecoli” and “Wine” dataset from UCI repository are also included. This experiment used the datasets after applying the resampling technique with different thresholds and compare the performance using the ranking method. Each set of synthesised dataset performance will be averaged before ranking. F1-scores was used to give the rank to the best threshold from the results, a threshold with the highest F1-score will receive the low rank value. The number reported is just the average rank

for all datasets. Hence, the best threshold is the one with the lowest average rank, see Figure 3.1. The best threshold based on these experiments is the 90^{th} percentile. Therefore, 10% of majority instances will be removed in the data cleansing step and 10% of minority instances will be used in the oversampling step to make sure that there are enough minority instances for a classifier to recognize them.

Datasets	**#instances**	**#features**	**IR**	**#clusters**
synthesised 1	100	5	0.1	5
synthesised 2	100	2	0.1	2
synthesised 3	100	3	0.1	3
synthesised 4	100	3	0.2	3
synthesised 5	100	2	0.2	2
synthesised 6	300	3	0.2	3
synthesised 7	300	2	0.2	2
synthesised 8	300	2	0.5	2
synthesised 9	300	5	0.5	5
synthesised 10	300	5	0.1	5
Ecoli	178	13	0.2687	-
Wine	336	7	0.1041	-

Table 3.1: Information of datasets used in the experiment to find threshold values

3.1.2 Data cleansing method

One component of the proposed method is the data cleansing step which is performed on abnormal instances from the majority class. The idea is to remove abnormal majority instances or border majority instances which lies in the border of the decision boundary of the majority class region. Hence, after this step, all abnormal instances from the majority class are removed and some majority instances in the overlapping region

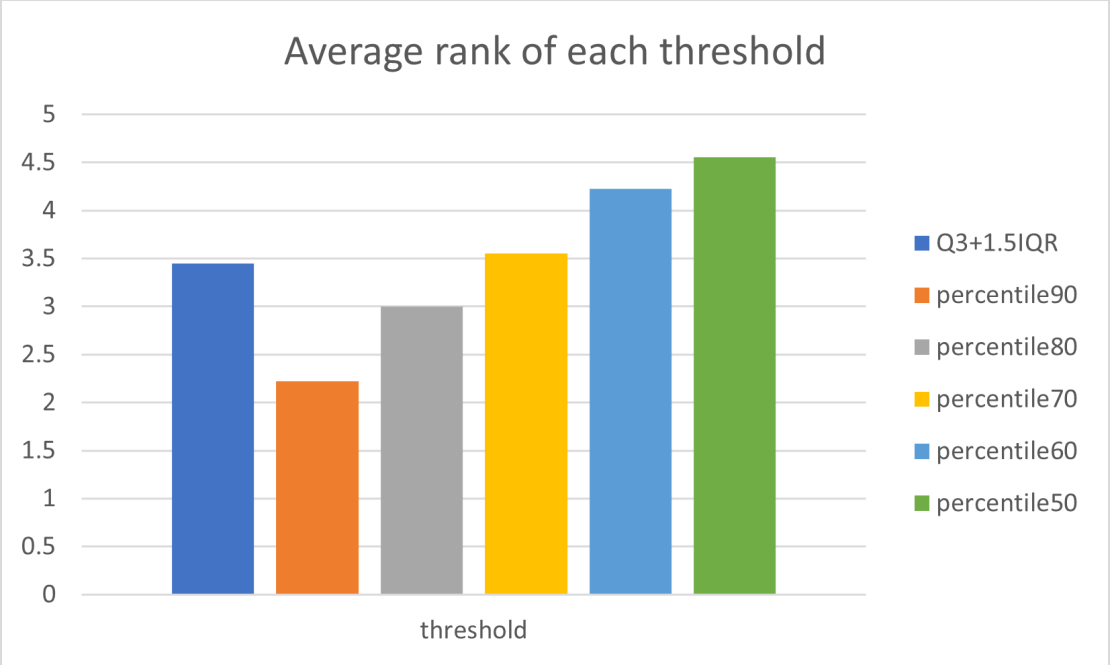


Figure 3.1: Average rank of each threshold compared

of the minority class will be removed which make the dataset clean and make the area between decision boundaries more clear. In the next section, the oversampling technique will be applied to the minority class so it will be balanced by this step.

3.1.3 Oversampling method

The proposed oversampling method was only performed on abnormal instances from the minority class, hence it will focus on the abnormal minority instances. The main idea of this method is to synthesise the minority instances that are far away from other minority instances. In this work, the Euclidean distance is used to measure the distance from an abnormal minority instance to the nearest majority instance which is defined as the synthesised radius. The radius was used as the maximum distance of the synthesis region in each direction to avoid the overlapping with other classes.

This oversampling method used abnormal minority instances as the centres of synthesise region. The process of this oversampling method started with computing MOF

and selecting abnormal instances from the minority class for synthesising. The next step is computed the number of synthesised minorities. The number of synthesis is defined as a proportion of the different number of instances between two classes according to its MOF see Definition 3.1. Abnormal instances having high MOF will receive a large number of synthesised instances with the reason of that an instance having high MOF is an instance having low density. For an instance with lower MOF, it does not need more minorities surrounding it. So few numbers of neighbours are needed. The next step is to find the radius of each abnormal minority instance which is the distance from an abnormal minority instance to the nearest majority instance. These radiuses guarantee that the region to be generated will not be overlap with other classes. Then, the last step is the synthesising step, minority instances will be synthesised into the synthesised region according to the number of synthesised instances. The input dataset distribution will be altered and become balance.

Definition 3.1 (The number of synthesised instances)**.** Given dataset $D \subseteq \mathbb{R}^d$ and a set of abnormal instances $A \in D$, N_- is the number of instances in the majority class, N_+ is the number of instances in the minority class. For abnormal instance $x \in A$, $MOF(x)$ is a mass-ratio-variance score of instance x. The number of synthesised instances for x is defined as

$$N_{syn}(x) = (N_- - N_+) \times \frac{MOF(x)}{\sum\limits_{y \in A} MOF(y)}$$

3.2 The MCOT process flow

In this section, the MCOT process flow is shown. The details of each process will be explained in this section. Also, some examples will be given to demonstrate how each process is performed.

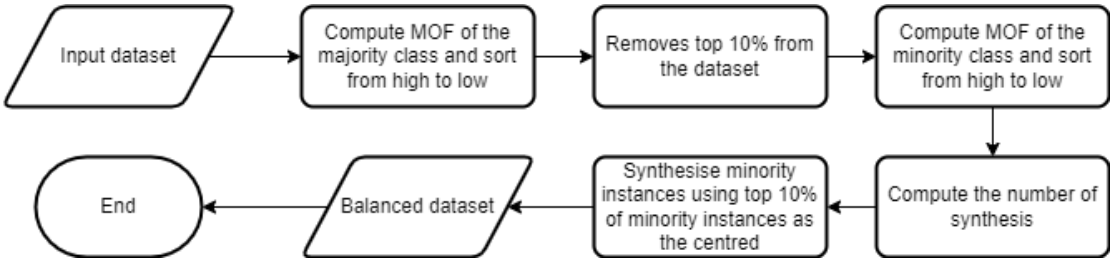


Figure 3.2: The MCOT process flow

3.2.1 Data processing methods and the MCOT algorithm

Figure 3.2 is the MCOT process flow. It starts with a dataset as input. The input dataset can be both an imbalanced or balanced dataset. Then it computes MOFs for instances from the majority class. After this process, top 10% of majority instances sorted by their MOFs are removed. This helps cleaning up abnormal majority instances and overlapping majority instances. Then MOFs for instances from the minority class are computed and top 10% of minority instances are selected to be used for synthesising, call abnormal minority instances. The number of synthesised minority instances is computed for each abnormal minority instance. The last process is the synthesising step, this process will generate artificial minority instances randomly into each hyper-ball according to the number of synthesised minority instances. After all processes are done, the balance dataset for both the majority class and the minority class will be returned..

3.2.2 Demonstrated MCOT examples

This part shows the computation of the MCOT process flow based on a given example. It includes the input of dataset, selected abnormal minority instances, oversampling method and the result dataset.

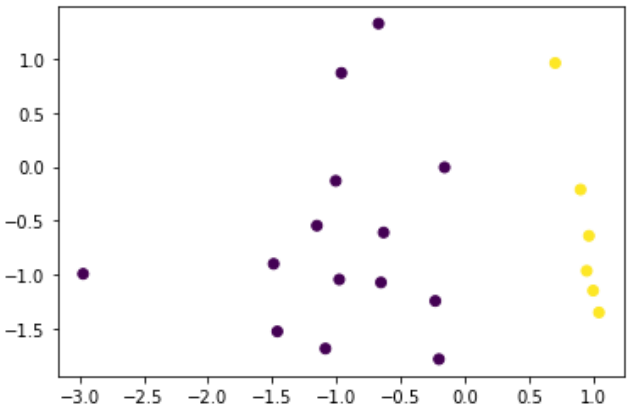


Figure 3.3: Example imbalanced dataset

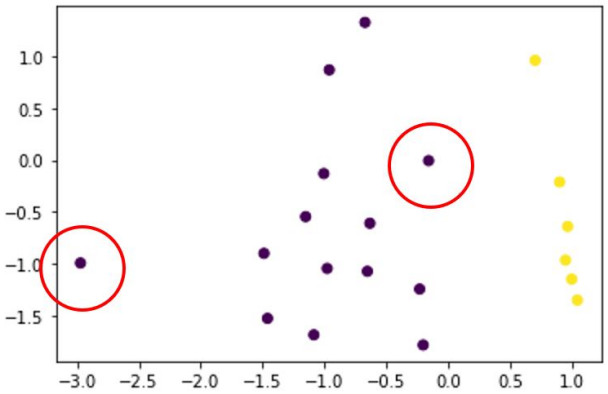


Figure 3.4: Majority instances to be removed

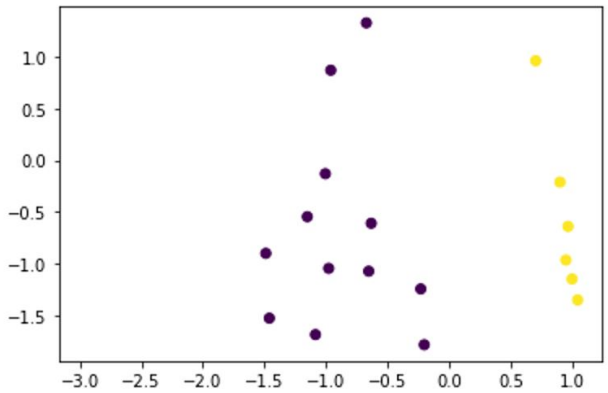


Figure 3.5: Cleaned dataset

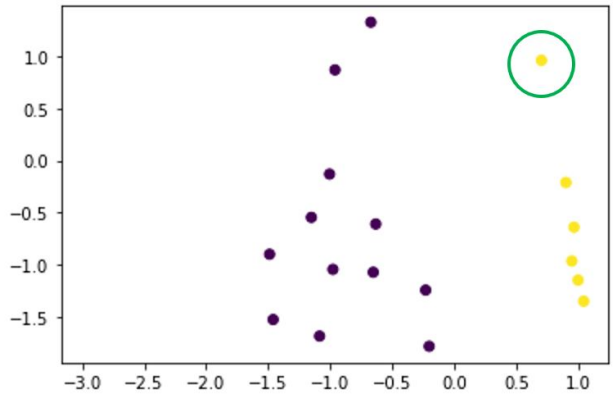


Figure 3.6: Abnormal minority instance

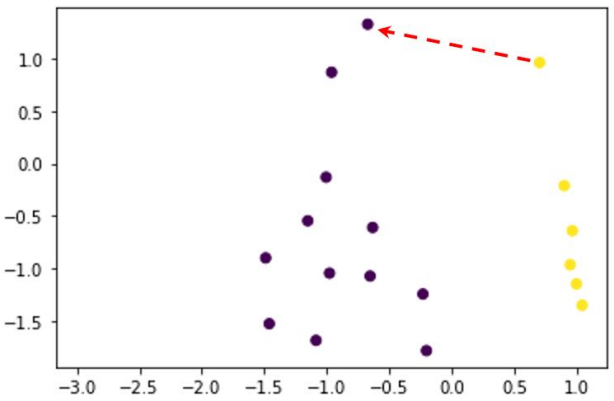


Figure 3.7: The radius of minority abnormal instance to the nearest majority instance

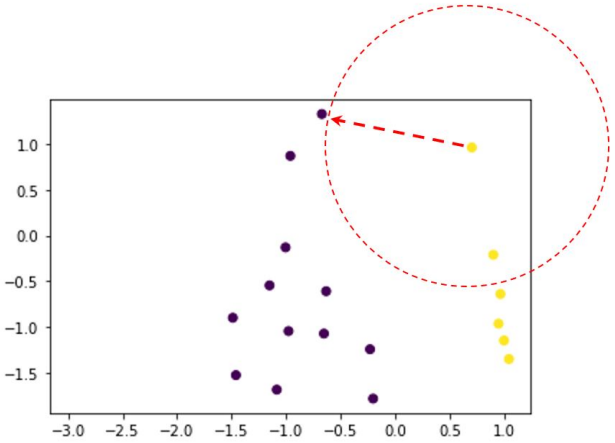


Figure 3.8: Open ball to be synthesised

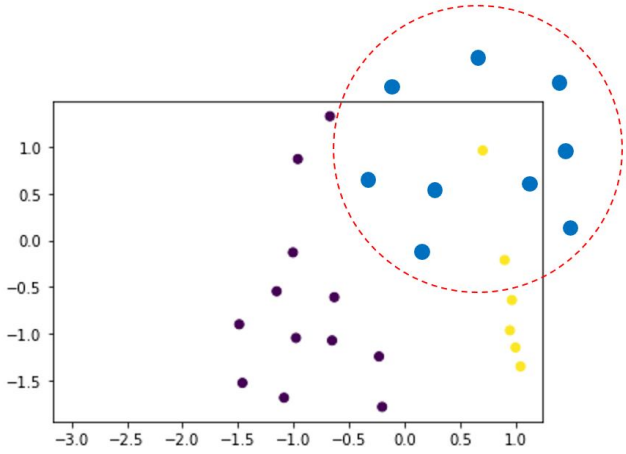


Figure 3.9: Synthesise minority instances into the ball according to the ratio of MOF

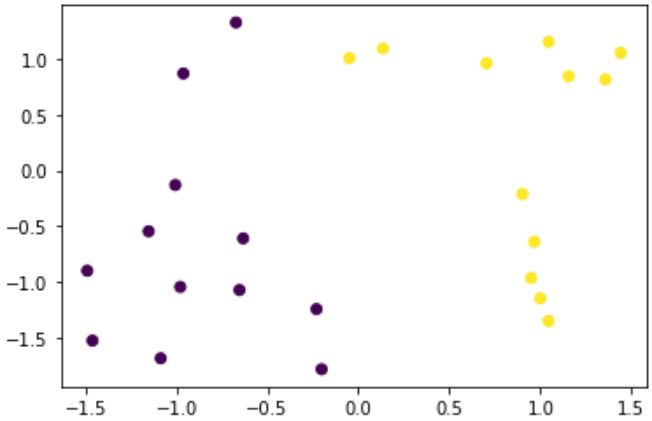


Figure 3.10: Balanced dataset

3.3 Pseudo code of the MCOT algorithm

The following pseudocode demonstrates the step-by-step of the MCOT algorithm.

Algorithm 2 MCOT(X, y, p) part1

Input: Array of data X; vector of target y; percentile threshold p (default = 90)
nFeat = number of features, nPos = number of minority instances,
nNeg = number of majority instances.
Note:
1. y_i is 0 for a majority instance and 1 for a minority instance
2. # is used for a line comment
3. compute_MOF(S) returns MOF scores of each instance in S
4. percentile(S, p) returns the p^{th} percentile value from S
5. Selected instances is represented in a numpy array as Variable_name[*conditions* or *index*]
6. ball(x, nFeat, r, n) returns n synthesised instances within a ball centred at x and radius r
7. enumerate(S) returns a sequence of (i, s) where i is the index of s in S
8. concatenate(S, T) returns the new set that concatenate T to S
Output: the balanced dataset X and y

```
XNeg is the subset of X having target y = 0
MOFNeg = compute_MOF(XNeg)
NegThreshold = percentile(MOFNeg, p)
NegAbnormal is the set of instances from XNeg having MOFNeg > NegThreshold
nNegAbnormal = the number of instances from NegAbnormal
X = is the subset of X removing NegAbnormal
nNegNormal = nNeg - nNegAbnormal
nSyn = nNegNormal - nPos # nSyn will be the number of synthesised instance
if nSyn > 0 then
    # Do the oversampling step
    XPos is the subset of X having target y = 1
    MOFPos = compute_MOF(XPos)
    PosThreshold = percentile(MOFPos, p)
    PosAbnormal is the set of instances from XPos having MOFPos > PosThreshold
    nPosAbnormal = the number of instances from PosAbnormal
    nSynPos=nSyn*(MOFPos[PosAbnormal]/ Sum(MOFPos[PosAbnormal]))
    NegIndex = index of nearest majority instance from instances in PosAbnormal
    radius = distance[NegIndex] # radius is set as the distance from the minority instance to its nearest majority instance
```

Algorithm 3 part2

```
19:     for i, abnormal in enumerate(PosAbnormal) do
20:         SynPos = ball(abnormal, nFeat, radius[i], nSynPos[i])
21:         X = concatenate(X, SynPos)
22:         Y = concatenate(Y, 1)
23: return X,y
    End MCOT
```

3.4 Computational complexity

This section shows the algorithm analysis of the MCOT algorithm. Time complexity will be computed using the worst case analysis. Given a dataset D having n instances since the MCOT algorithm is used for an imbalanced dataset with m as the number of majority instances. The time complexity will depend mostly on m which has the highest proportion of instances in the dataset. The MCOT algorithm contains 4 steps. The first step is to compute MOFs of instances from the majority class. This has the time complexity of $O(m^2 \log(m))$. The second step is the cleansing step which removes abnormal majority instances which takes $O(m)$. The third step is to compute MOFs of instances from the minority class that will be dominated by $O(m^2 \log(m))$ from the majority class. The next step is the oversampling step which composes from two parts. The first part of the oversampling step is to select an abnormal minority instances and compute the number of synthesis which take time significantly less than $O(m)$. The second part of the oversampling step is the synthesised step, this step generates minority instances for each minority abnormal instance then it takes the time complexity of $O(m)$ to make it balance. Hence, the summary of the time complexity of the MCOT algorithm is $O(m^2 \log(m))$.

CHAPTER IV

EXPERIMENTS AND RESULTS

This chapter discusses about the experiments and results of the MCOT algorithm. It contains four sections. The first section shows the setting of the parameters, datasets and classification algorithms used in the experiments. The datasets for this experiment are both synthesis and real world datasets. The parameters of synthesised datasets and details of real world datasets are also shown in this section. The number of selected classification algorithms is four along with the reasons for choosing them. The second section reports the classification metrics of the experiments performed by four classifiers and compares them. The third section covers the result of the nonparametric test using Wilcoxson signed-rank test to confirm findings. Finally, the analysis of the experimental results and conclusion are then described.

4.1 Experimental setting

This section demonstrates the experimental configurations of the MCOT algorithm using four classifiers. Datasets from the real world and synthesised datasets are both included in the experiments. Three collections of synthesised datasets have an imbalance ratio (IR) of 0.1, 0.2, and 0.3.

4.1.1 Synthesised datasets used in this experiment

The experiment has 36 types that can be grouped by the subcollection and by the collection. There are three collections of synthesised datasets based on their IR values. Each collection contains three subcollections based on the number of clusters (2, 3, or 4). Each subcollection will be generated four types based on the number of features (3, 5) $\times$ the number of instances (100, 300), see Table 4.1. The datasets are then generated at random 30 times for each type, and the average performance for each collection and

subcollection from these 120 datasets. There are 1080 synthesised datasets in total. The results are average performance measures based on various classifiers and measures for each subcollection.

Synthesised datasets			
Collection (IR)	**Subcollection (#Clusters)**	**#features**	**#instances**
1 (IR = 0.1)	1.1 (2)	3,5	100,300
	1.2 (3)	3,5	100,300
	1.3 (4)	3,5	100,300
2 (IR = 0.2)	2.1 (2)	3,5	100,300
	2.2 (3)	3,5	100,300
	2.3 (4)	3,5	100,300
3 (IR = 0.3)	3.1 (2)	3,5	100,300
	3.2 (3)	3,5	100,300
	3.3 (4)	3,5	100,300

Table 4.1: Information of synthesised datasets used in the experiment

4.1.2 Real world datasets used in this experiment

In the experiment, five UCI datasets are used. The minority class in the experiment is chosen as in Table 4.2. Five UCI datasets are briefly described, along with their characteristics. The process of converting multiclass datasets to binary datasets involves choosing one class as the minority class and the rest as the majority class. The target class in Table 4.2 is designated as the minority class in the "minority target" column, while the other classes are designated as the majority class. These five datasets together have an average IR value of 0.2467.

Real world datasets					
Datasets	**#instances**	**#features**	**minority target**	**#minority**	**IR**
Wine	178	13	“3”	48	0.2687
Parkinsons	195	22	“0”	48	0.2461
Haberman	306	3	“2”	81	0.2647
Ecoli	336	7	“imU”	35	0.1041
Pima	768	9	“1”	268	0.3489
Average					0.2467

Table 4.2: Information of UCI datasets used in the experiments

4.1.3 Classification algorithms and performance metric used in this experiment

In this experiment, four classifiers are used: a decision tree, a random forest, a linear support vector machine, and a multi layer perceptron. These four classifiers were previously discussed in Chapter 1. Chapter 1 also provides a description of the performance metrics that were employed in this experiment. Figure 4.1 compares the average precision, recall, and F1-score performances of four classifiers when using the original datasets and datasets after applying the MCOT algorithm.

4.2 Results

4.2.1 Result 1: Synthesised datasets

The results of the experiments of synthesised datasets are shown in Table 4.3 and Figure 4.1. Each cell in Table 4.3 reports mean±sd from each setting, where mean is the average performance and sd is the standard deviation. To visualize the performance in Figure 4.1, the barplots are used for each measurement compared between the original

and MCOT. It has a 3 × 3 barplots by the collections and the subcollections.

Collection	**Subcollection**	**Mean±sd**			
IR	**#cluster**	**Processed**	**Precision**	**Recall**	**F1-score**
IR = 0.1	2	None	0.6030 ± 0.1670	0.3974 ± 0.1077	0.4474±0.1118
		MCOT	0.5035 ± 0.1200	0.5110 ± 0.1175	0.4767 ± 0.1063
	3	None	0.4403 ± 0.1972	0.2399 ± 0.1054	0.2810 ± 0.1144
		MCOT	0.3774 ± 0.1463	0.3904 ± 0.1330	0.3499 ± 0.1268
	4	None	0.4591 ± 0.0398	0.3446 ± 0.1746	0.3571 ± 0.1740
		MCOT	0.4368 ± 0.1296	0.5068 ± 0.1740	0.4344 ± 0.1367
IR = 0.2	2	None	0.7699 ± 0.0949	0.6147 ± 0.0877	0.6608 ± 0.0871
		MCOT	0.6656 ± 0.0917	0.7139 ± 0.0700	0.6675 ± 0.0734
	3	None	0.6577 ± 0.1086	0.4536 ± 0.0979	0.5046 ± 0.0948
		MCOT	0.5653 ± 0.0975	0.6003 ± 0.0655	0.5562 ± 0.0728
	4	None	0.6780 ± 0.1435	0.5069 ± 0.1356	0.5485 ± 0.1371
		MCOT	0.5685 ± 0.1200	0.6265 ± 0.1224	0.5743 ± 0.1187
IR = 0.3	2	None	0.8023 ± 0.0693	0.7254 ± 0.0721	0.7461 ± 0.0670
		MCOT	0.7194 ± 0.0671	0.7942 ± 0.0447	0.7450 ± 0.0556
	3	None	0.7441 ± 0.0889	0.5938 ± 0.1072	0.6401 ± 0.1024
		MCOT	0.6537 ± 0.0806	0.6963 ± 0.0734	0.6614 ± 0.0761
	4	None	0.7366 ± 0.0995	0.6277 ± 0.1322	0.6606 ± 0.1191
		MCOT	0.6618 ± 0.0806	0.7052 ± 0.1014	0.6730 ± 0.0905
UCI		None	0.5750 ± 0.2381	0.4774 ± 0.2453	0.4843 ± 0.2559
		MCOT	0.5646 ± 0.1903	0.6087 ± 0.1939	0.5551 ± 0.2054

Table 4.3: Information of UCI datasets used in the experiment

According to Table 4.3, all experiments show a decrease in precision and an increase in recall. All experiments indicate an improvement in F1-scores. Notice that the improvement in recalls and F1-scores for the number of clusters = 2 is small compared to the larger number of clusters for all collections. The cause of this behaviour is that by eliminating majority abnormal instances, the minority region will be expanded, allowing a classifier to correctly classify more minority instances, increasing recall. However, this

behaviour will reduce precision because the likelihood that some majority instances will be predicted as minority instances increases as the number of minority predictions increases. As a result, F1-score should be used to assess MCOT performance. This demonstrates how the MCOT algorithm can assist classifiers in providing a better improvement for original datasets with IR of less than or equal to 0.2.

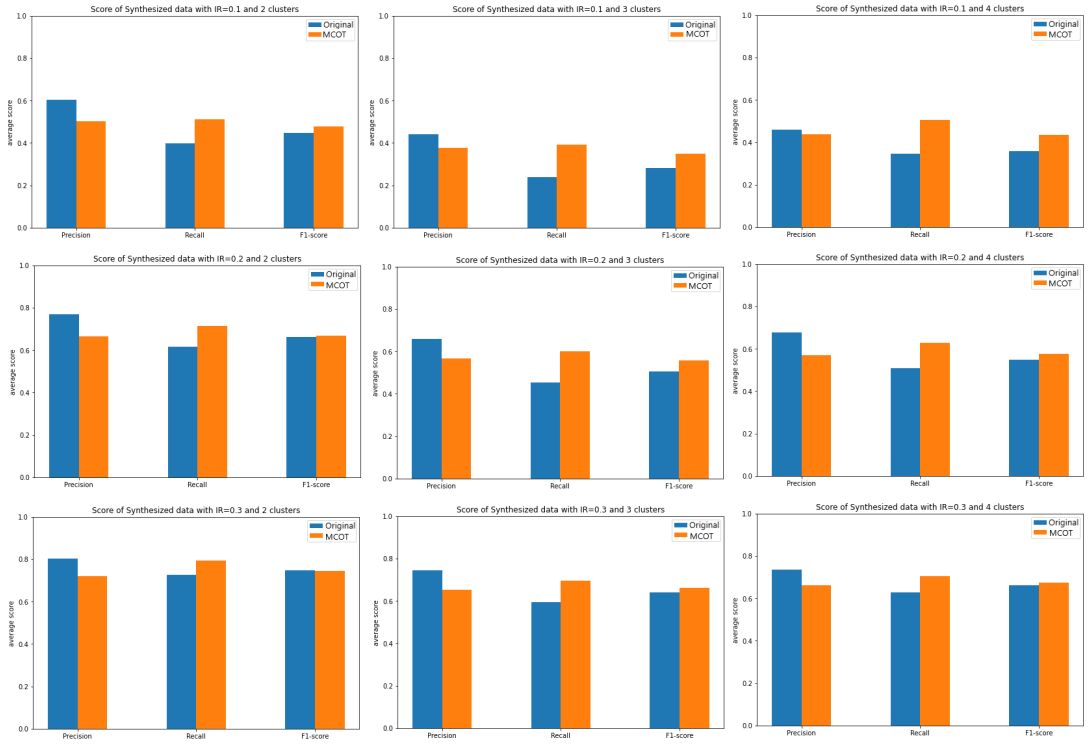


Figure 4.1: Average precision, recall and F1-score of each collection

4.2.2 Result 2: Real world datasets

Based on previous observations, the MCOT algorithm can assist classifiers in achieving better recall and F1-score for IR less than 0.3, so improved performance for these five UCI datasets with an average IR of 0.2467 is expected. The average performances of the UCI datasets are displayed in Figure 4.2. It demonstrates that recall and F1-score have improved. From this, it can be inferred that the MCOT algorithm can aid in classifiers' improvement of F1-score and recall.

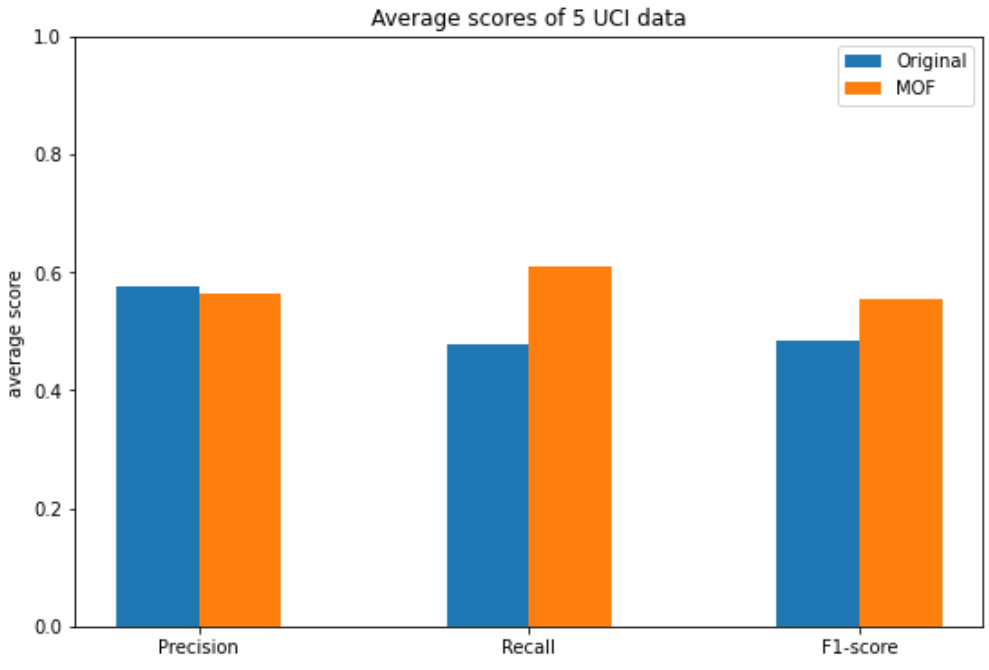


Figure 4.2: Average precision, recall and F1-score of five UCI dataset

4.3 Nonparametric test (Wilcoxson signed-rank test)

The statistical significance of datasets generated by the MCOT algorithm compared to the original datasets is assessed using the Wilcoxon signed-rank tests. Four common classifiers will perform on the original datasets in each test, and they will be compared to the datasets produced by the MCOT algorithm using the same classifier. The Wilcoxon signed-rank test's p values were displayed in Table 4.4. The p-value must be less than 0.05 in order for it to be deemed significantly different.

Collection	Subcollection	P-value		
IR	#cluster	Precision	Recall	F1-score
IR = 0.1	2	0.004181	3.05E-05	0.015503
	3	3.35E-02	6.10E-05	0.000214
	4	4.04E-01	3.05E-05	0.000305
IR = 0.2	2	6.10E-05	3.05E-05	4.04E-01
	3	4.27E-04	3.05E-05	9.16E-05
	4	6.10E-05	3.05E-05	6.29E-03
IR = 0.3	2	3.05E-05	3.05E-05	0.175354
	3	9.16E-05	3.05E-05	0.028992
	4	3.05E-05	3.05E-05	0.433197
UCI		4.75E-01	3.62E-05	7.30E-03

Table 4.4: p values of synthesised and UCI datasets used in the experiment

Collection 3 with IR = 0.3 and cluster sizes of 2 and 4 displays no significantly different F1-score performances because their p-values are higher than 0.05. Since the p-values for all other collections are less than 0.05, the MCOT algorithm help achieve a significant improvement. It can be said that MCOT is more effective on datasets with imbalances and an IR value of less than or equal to 0.2.

4.4 Results analysis

The MCOT algorithm generates a new dataset that increases recall for a classifier because more positive instances can be easily identified, but it decreases precision due to the enlarged minority regions, so the F1-score is the preferred measure that combines recall and precision. All results indicate decrease in precision and increase in recall and F1-score. Recalls and F1-scores increase from three collections with varying IR values. The more evenly distributed the datasets are, the less improvement MCOT will have.

When the number of clusters is equal to 4, MCOT shows the highest F1-score among the results of various clustering densities. This could result from the minority instances spreading across all clusters. Therefore, MOF can be very effective at detecting abnormal instances when the dataset contains a greater number of minority instances. MCOT increases recall and F1-score while decreasing precision for UCI datasets. F1-scores between the original datasets and the datasets from the MCOT algorithm with respect to four classifiers statistically improves.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

This is the conclusion chapter. It summarizes the MCOT algorithm and the findings. It also provides the future work for this thesis.

5.1 Conclusions

Our thesis proposed the resampling techniques to deal with a class imbalance problem, called the mass ratio variance majority cleansing and minority oversampling technique (MCOT). The parameter-free technique for anomaly scoring algorithm MOF is used to select instances from both the majority class (for cleansing) and the minority class (for synthesising). The top 10% MOF are determined by the experiment on the decision tree. The MCOT algorithm is composed of two main steps. The first step is to cleanse majority noises and majority border instances. The second step performs the oversampling technique to abnormal minority instances by generating minority instances inside the ball that does not include any majority instance. This expands the minority regions and helps classifiers identify more minority instances which increases recall and F1-score.

Experiments with synthesised datasets performs on three collection of IR values and three subcollections according to the number of clusters. The show that the MCOT algorithm provides the dataset that improve recall and F1-score. Furthermore, the results from the Wilcoxon signed-rank test show the significant improvement for the datasets with IR less than 0.3. As a result, when a dataset is imbalanced, the MCOT algorithm can handle it very effectively.

5.2 Future work

More datasets should be tested and investigated further via MCOT. In this thesis the oversampling method applied only on abnormal minority instances but it will be interesting to apply it to all minority instances. By partitioning the number of syntheses to all minority instances to give the different opportunities to increase density of the whole region. An instance having high MOFs still receives a high number of synthesis. This idea will make the area of minority class evenly distributed more evenly than the proposed method.

Since MCOT uses both data cleansing and oversampling methods then the threshold values for data cleansing and oversampling should be investigated. The threshold value to decide abnormal instances from each class can be changed, it can set to different values for each class.

Moreover, the time complexity of the MCOT algorithm depends only on the number of majority instances during the computation of MOF. To improve the time complexity of the MCOT, the MOF step should be updated by improving the ways of storage and collecting the number of neighbours.

Due to the MCOT process framework, the binary class imbalance problem is performed for each class. So it should be easily to extend this concept to the multiclass imbalance problem by determining whether to perform cleansing for some classes or to perform the oversampling technique for other classes.

In addition, the MCOT algorithm is not designed to work with discrete variables. It will be more useful if this method could extend to work with any type of variable.

REFERENCES

[1] L. Breiman, "Random forests machine learning, vol. 45," 2001.

[2] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[3] S. Janitza and R. Hornung, "On the overestimation of random forest's out-of-bag error," *PloS one*, vol. 13, no. 8, p. e0201904, 2018.

[4] A. Ali, S. M. Shamsuddin, and A. L. Ralescu, "Classification with class imbalance problem," *Int. J. Advance Soft Compu. Appl*, vol. 5, no. 3, 2013.

[5] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," in *2017 international conference on computing networking and informatics (ICCNI)*, pp. 1–9, IEEE, 2017.

[6] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[7] I. Bratko, J. Žabkar, and M. Možina, "Argument-based machine learning," in *Argumentation in artificial intelligence*, pp. 463–482, Springer, 2009.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[9] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*, pp. 878–887, Springer, 2005.

[10] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 475–482, Springer, 2009.

[11] N. Rout, D. Mishra, and M. K. Mallick, "Handling imbalanced data: a survey," in *International proceedings on advances in soft computing, intelligent systems and applications*, pp. 431–443, Springer, 2018.

[12] A. Gosain and S. Sardana, "Handling class imbalance problem using oversampling techniques: A review," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*, pp. 79–85, IEEE, 2017.

[13] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Dbsmote: density-based synthetic minority over-sampling technique," *Applied Intelligence*, vol. 36, no. 3, pp. 664–684, 2012.

[14] C. Chiamanusorn and K. Sinapiromsaran, "Extreme anomalous oversampling technique for class imbalance," in *Proceedings of the 2017 International Conference on Information Technology*, pp. 341–345, 2017.

[15] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.

[16] C. Bunkhumpornpat and K. Sinapiromsaran, "Dbmute: density-based majority under-sampling technique," *Knowledge and Information Systems*, vol. 50, no. 3, pp. 827–850, 2017.

[17] N. Junsomboon and T. Phienthrakul, "Combining over-sampling and under-sampling techniques for imbalance dataset," in *Proceedings of the 9th international conference on machine learning and computing*, pp. 243–247, 2017.

[18] Y. Sahin, S. Bulkan, and E. Duman, "A cost-sensitive decision tree approach for fraud detection," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5916–5923, 2013.

[19] K. Boonchuay, K. Sinapiromsaran, and C. Lursinsap, "Decision tree induction based on minority entropy for the class imbalance problem," *Pattern Analysis and Applications*, vol. 20, no. 3, pp. 769–782, 2017.

[20] A. Sagoolmuang and K. Sinapiromsaran, "Oblique decision tree algorithm with minority condensation for class imbalanced problem," *Engineering Journal*, vol. 24, no. 1, pp. 221–237, 2020.

[21] A. Sagoolmuang and K. Sinapiromsaran, "Decision tree algorithm with class overlappingbalancing entropy for class imbalanced problem," *International Journal of Machine Learning and Computing*, vol. 10, no. 3, pp. 444–451, 2020.

[22] S. Kanchanasuk and K. Sinapiromsaran, "Recursive tube-partitioning algorithm for a class imbalance problem," *Thai Journal of Mathematics*, vol. 18, no. 4, pp. 2041–2051, 2020.

[23] P. Thanathamathee and C. Lursinsap, "Handling imbalanced data sets with synthetic boundary data generation using bootstrap re-sampling and adaboost techniques," *Pattern Recognition Letters*, vol. 34, no. 12, pp. 1339–1347, 2013.

[24] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[25] D. Devi, B. Purkayastha, *et al.*, "Redundancy-driven modified tomek-link based undersampling: A solution to class imbalance," *Pattern Recognition Letters*, vol. 93, pp. 3–12, 2017.

[26] 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)*, pp. 1–5, IEEE, 2021.

[27] A. Zimek and E. Schubert, *Outlier Detection*, pp. 1–5. New York, NY: Springer New York, 2017.

[28] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[29] M. Amer and M. Goldstein, "Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer," 08 2012.

[30] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[31] M. Amer and S. Abdennadher, "Comparison of unsupervised anomaly detection techniques," *Bachelor's Thesis*, 2011.

[32] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *European conference on principles of data mining and knowledge discovery*, pp. 15–27, Springer, 2002.

[33] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*, pp. 196–202, Springer, 1992.

APPENDIX

BIOGRAPHY

Name Mr. Piboon Polvimoltham

Date of Birth May 29, 1996

Place of Birth Bangkok, Thailand

Educations B.Sc. (Mathematics) , Thammasat University, 2018

Publications

- **Polvimoltham, P.** and Sinapiromsaran, K. , 2021, October. Mass Ratio Variance Majority Undersampling and Minority Oversampling Technique for Class Imbalance. In *Fuzzy Systems and Data Mining VII* (p. 152-161). IOS Press.