A MODIFIED WHALE OPTIMIZATION ALGORITHM FOR IMPROVING DATA BALANCE BASED ON UNDERSAMPLING TECHNIQUES

JAKKRIT POLROB

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Applied Mathematics

Suranaree University of Technology

Academic Year 2021

ขั้นตอนวิธีการหาค่าเหมาะที่สุดแบบวาฬแปลงสำหรับปรับปรุงสมดุลข้อมูลอิงกลวิธีการเลือกตัวอย่างลด

นายจักรกฤษณ์ พลรบ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาคณิตศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปีการศึกษา 2564

A MODIFIED WHALE OPTIMIZATION ALGORITHM FOR IMPROVING DATA BALANCE BASED ON UNDERSAMPLING TECHNIQUES

Suranaree University of Technology has approved this thesis submitted in partial fulfillment of the requirements for a Master's Degree.

Thesis Examining Committee

Suthep Suantai

(Prof. Dr. Suthep Suantai)
Chairperson

Benjawan Rodjanadid

(Asst. Prof. Dr. Benjawan Rodjanadid)
Member (Thesis Advisor)

J.Tanthanuch

(Asst. Prof. Dr. Jessada Tanthanuch)
Member (Thesis Co-advisor)

[illegible]

(Assoc. Prof. Dr. Eckart Schulz)
Member (Thesis Co-advisor)

Panu

(Asst. Prof. Dr. Panu Yimmaung)
Member

Tidarut Areerak

(Asst. Prof. Dr. Tidarut Areerak)
Member

Amornrat Suriyawichitseranee

(Dr. Amornrat Suriyawichitseranee)
Member

[illegible]

(Assoc. Prof. Dr. Chatchai Jothityangkoon)
Vice Rector for Academic Affairs
and Quality Assurance

[illegible]

(Prof. Dr. Santi Maensiri)
Dean of Institute of Science

จักรกฤษณ์ พลรบ : ขั้นตอนวิธีการหาค่าเหมาะที่สุดแบบวาฬแปลงสำหรับปรับปรุงสมดุลข้อมูลอิงกลวิธีการเลือกตัวอย่างลด (A MODIFIED WHALE OPTIMIZATION ALGORITHM FOR IMPROVING DATA BALANCE BASED ON UNDERSAMPLING TECHNIQUES)
อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.เบญจวรรณ โรจนดิษฐ์, 112 หน้า.

คำสำคัญ: ข้อมูลไม่สมดุล/การเลือกตัวอย่างลด/ขั้นตอนวิธีการหาค่าเหมาะที่สุดแบบวาฬ/ขั้นตอนวิธีการหาค่าเหมาะที่สุดแบบไบนารี่วาฬ

การวิจัยครั้งนี้มีวัตถุประสงค์เพื่อสร้างขั้นตอนวิธีการเลือกตัวอย่างลดแบบใหม่ โดยประยุกต์ใช้ขั้นตอนวิธีการหาค่าเหมาะสมที่สุดแบบวาฬและไบนารี่วาฬร่วมกับเทคนิคเพื่อนบ้านใกล้ที่สุด เพื่อแก้ปัญหาข้อมูลไม่สมดุล ซึ่งมี 2 รูปแบบ คือ ตรึงค่าพารามิเตอร์เคเท่ากับหนึ่งและปรับค่าพารามิเตอร์เคได้ ทั้งนี้ได้เลือกชุดข้อมูลทดสอบจำนวน 10 ชุดจาก KEEL และ imbalanced-learn ซึ่งชุดข้อมูลจะมีอัตราส่วนความไม่สมดุลอยู่ในช่วง 1.82 ถึง 42.01 ขั้นตอนการทำงานเริ่มจากนำชุดข้อมูลมาแบ่งเป็นสองชุด คือ ชุดข้อมูลฝึกสอนและชุดข้อมูลทดสอบสำหรับข้อมูลชุดฝึกสอน ชั้นข้อมูลกลุ่มน้อยยังคงเก็บไว้เหมือนเดิม ในขณะที่ชั้นข้อมูลกลุ่มมากจะถูกวิเคราะห์เพื่อดึงชุดข้อมูลย่อยที่เป็นตัวแทนที่ดีที่สุดด้วยขั้นตอนวิธีที่นำเสนอ โดยประเมินประสิทธิภาพด้วยเทคนิคดังนี้ เทคนิคซัพพอร์ตเวกเตอร์แมทชีน เทคนิคป่าสุ่ม และ เทคนิคต้นไม้ตัดสินใจ ผลการวิจัยพบว่าขั้นตอนวิธีที่นำเสนอแบบปรับค่าพารามิเตอร์เคได้ร่วมกับแบบจำลองเทคนิคป่าสุ่มมีประสิทธิภาพการทำงานโดยรวมสูงที่สุด โดยมีผลการวัดประสิทธิภาพเฉลี่ยดังนี้ Accuracy = 0.8387, F1 score = 0.5783, G-mean = 0.8794, AUROC = 0.9212, AUPRC = 0.6457, Sensitivity = 0.9399, Precision = 0.4703, MCC = 0.5705, และ Kappa = 0.5123 โดยรวมแล้วขั้นตอนวิธีที่นำเสนอมีประสิทธิภาพดีกว่าสามขั้นตอนวิธีการเลือกตัวอย่างลดดังต่อไปนี้ คือ การลดจำนวนตัวอย่างข้อมูลแบบสุ่ม คลัสเตอร์เซนทรอยด์ และ เนียร์มิส

สาขาวิชาคณิตศาสตร์	ลายมือชื่อนักศึกษา จักรกฤษณ์ พลรบ
ปีการศึกษา 2564	ลายมือชื่ออาจารย์ที่ปรึกษา Benjawan Rodjanadid
	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม J.Tanthanuch
	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม [illegible]

JAKKRIT POLROB : A MODIFIED WHALE OPTIMIZATION ALGORITHM FOR IMPROVING DATA BALANCE BASED ON UNDERSAMPLING TECHNIQUES. THESIS ADVISOR : ASST. PROF. BENJAWAN RODJANADID, Ph.D. 112 PP.

Keyword: IMBALANCED DATA, UNDERSAMPLING TECHNIQUE, WHALE OPTIMIZATION ALGORITHM, BINARY WHALE OPTIMIZATION ALGORITHM

This thesis is aimed at developing a novel undersampling algorithm, by combining the idea of whale and binary whale optimization algorithms with K-nearest neighbor classification for solving imbalanced data problems. There are two versions, one with fixed-parameter $K = 1$, and the other with adjustable parameter K. Ten datasets of varying imbalance ratios ranging from 1.82 to 42.01 were selected from the Knowledge Extraction based on Evolutionary Learning (KEEL) and imbalanced-learn repositories to be used in the evaluation of the novel algorithms. The work started by splitting each dataset into two parts, the training set and the testing set. Whereas the minority class of each training set remained untouched, the majority class was analyzed by the proposed algorithms to extract an optimal representative subset. Then, the support vector machine, random forest, and decision tree classifiers were trained with the new training set for assessing performance. It was found that the proposed algorithms applied to the random forest model had the highest overall performance, with average efficiency measurement results as follows: Accuracy = 0.8387, F1 score = 0.5783, G-mean = 0.8794, AUROC = 0.9212, AUPRC = 0.6457, Sensitivity = 0.9399, Precision = 0.4703, MCC = 0.5705, and Kappa = 0.5123. In particular, the proposed algorithm performed better overall than three common undersampling algorithms, namely random undersampling, cluster centroid, and near-miss algorithms.

School of Mathematics	Student's Signature [illegible]
Academic Year 2021	Advisor's Signature [illegible]
	Co-advisor's Signature [illegible]
	Co-advisor's Signature [illegible]

ACKNOWLEDGEMENTS

First of all, I would like to thank all advisors who helped and advised on completing this thesis. My thesis advisor is Asst. Prof. Benjawan Rodjanadid, who helped me with everything, whether how to presentation, how to write a good paper, and always listened to my new ideas, I'm grateful for her support. Next, my first co-advisor is Asst. Prof. Jessada tanthanuch, who always suggests how to write a thesis form and support the study, I would not have done this thesis without him. The other co-advisor is Assoc. Prof. Eckart Schulz, who helped me how to write the correct English language and spent much of his time editing my thesis, I'm so grateful for his help. I appreciate all teachers in mathematics at the Institute of Science, Suranaree University of Technology (SUT), for teaching a lot of knowledge. I am very glad to have friends to study on the way together.

Finally, I am grateful for acknowledging the receipt of the Kittibandit scholarship from Suranaree University of Technology for my master's degree study. And finally, really thank you very much to everyone who made me successful in writing my thesis.

Jakkrit Polrob

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

CHAPTER I

INTRODUCTION

Data imbalance on a classification problem means that there is a disproportionate number of samples for each class. This situation can be encountered in many fields and applications, such as cancer diagnosis, where the number of cases of such disease is few in the entire population (Fotouhi, 2019), fraud detection in card transactions where the number of legitimate transactions is higher than the number of defrauders (Mqadi, 2021), classification of diabetic patients for whom the disease is rare compared to the total population (Kesornsit, 2018), and many others.

One of the most common problems with imbalanced data is that it may render a predictive model using conventional machine learning algorithms to become inaccurate and biased. This happens because machine learning algorithms are often designed to improve accuracy by reducing errors. Thus, the algorithm may maximize accuracy for elements of the majority, neglecting elements of the minority class.

Resampling techniques represent methods that can rebalance the data. Once the data is balanced, one can use this new data set to create a machine learning model. It will result in better performance by correctly predicting elements of the minority classes and reducing bias. Resampling techniques can be typically categorized into three groups: undersampling methods, oversampling methods, and hybrid methods (Fernández et al., 2018). One of the effective methods is undersampling, as it can reduce the size of majority classes to the corresponding size of the minority classes, even though one may lose some beneficial information. However In this way, it reduces the processing time of the machine learning model and can reduce overfitting.

At present, numerous nature-inspired algorithms have been applied to imbalanced data problems such as ant colony optimization algorithms (Yu, Ni, and Zhao, 2013), evolutionary algorithms (López, Triguero, Carmona, García, and Herrera, 2014), genetic algorithms (Kim, Jo, and Shin, 2016), and adaptive swarm balancing algorithms (Li et al., 2017). Recently, another two nature-inspired algorithms have become popular and have been applied to many fields of work, namely the whale optimization algorithm (Mirjalili, and Lewis, 2016) and the binary whale optimization algorithm (Kumar and Kumar, 2020). Examples of the application of these two algorithms are the feature selection problem (Mafarja and Mirjalili, 2017), electrical engineer problem (Kumar and Kumar, 2020), and parameter optimization problem, which can be applied to these tasks effectively. These algorithms are extremely interesting if applied to the imbalanced data problem.

Therefore, in this work, we present a novel algorithm that uses a combination of whale and binary whale optimization algorithms based for undersampling. We evaluate the performance of the proposed algorithm by comparing it with some of the most popular and widely used other techniques, which are Random Undersampling (Mishra, 2017), ClusterCentroid (imbalancedlearn, 2022), and Near-Miss (Mani and Zhang, 2003). There are various in prediction and classification methods, such as K-Nearest Neighbors (K-NN) (Aha, Kibler, and Albert, 1991), Support Vector Machine (SVM) (Cortes et al., 1995), Decision Tree (Breiman, Friedman, Olshen, and Stone, 1984), and Random Forest. We will employ the latter three methods to evaluate the performance of our undersampling method. Several performance metrics, such as Accuracy, Sensitivity, G-mean, F-measure, Area under the curves of Receiver operating characteristic (AUROC), and Cohen’s Kappa Statistics (Kappa), etc. will be used for this evaluation.

1.1 Research objective

To develop a novel technique for solving imbalanced data problems based on under-sampling using whale and binary whale optimization algorithms.

1.2 Scope and limitations

1. The datasets are selected from the Knowledge Extraction based on Evolutionary Learning (KEEL) and Imbalanced-learn repositories.
2. Using whale and binary whale algorithms to solve the imbalanced data problem.

3. The techniques for solving the classification problem in this study consist of the Support Vector Machine, Decision Tree, Random Forest, and K-nearest neighbors.

4. We use the Python programming language version 3.7.5, to develop the proposed algorithms.

1.3 Research procedure

The research work proceed as follows:

1. Study classification problems with imbalanced data.

2. Study data balancing techniques.

3. Study the whale and binary whale optimization algorithms.

4. Study classification algorithms in data mining and performance metrics for imbalanced data.

5. Create an algorithm to solve the problem of imbalanced data.

6. Analyze and construct the model for each algorithm.

7. Compare the performance of the algorithms that have been created with other algorithms.

1.4 Expected result

Our new algorithm is developed from a combination of the whale and binary whale optimization algorithms. It can effectively solve the problem of imbalanced data.

CHAPTER II

LITERATURE REVIEW

This section presents the knowledge of basic mathematics and machine learning related to imbalanced data problems. In particular, it reviews the main idea of the whale and binary whale optimization algorithms used in this study.

2.1 Classification Problem in Imbalanced Data

In classification tasks, one may encounter situations where the target class label is skew distributed across various classes. Such conditions are termed as imbalanced target classes. Modeling an imbalanced dataset is a major challenge faced by data scientists, as due to the presence of an imbalance in the data the model becomes biased towards the majority class prediction. Hence, handling the imbalance in the dataset is essential before model training. There are various things to keep in mind while working with imbalanced data.

In two-class problems, the minority class is usually referred to as the positive class, whereas the majority class is considered to be the negative one. The conventional way of referring to the degree of imbalance of two-class problems is the imbalance ratio (IR) (Orriols-Puig and Bernadó-Mansilla, 2009). The IR is defined as the number of negative class examples divided by the number of positive class examples and can be used to sort different datasets depending on their IR. One must take into account that the IR does not always give a good estimation of the difficulty of the dataset.

Figure 2.1 shows an example of a dataset of two-class problems with an imbalanced ratio of 100. However, if we model this data with a standard classifier, it can cause poor prediction of the minority class, because standard classifiers tend to be highly biased in their predictions; they aim at high overall accuracy. For example, from Figure 2.1, we might think that we are faced with a medical application where we should differ-

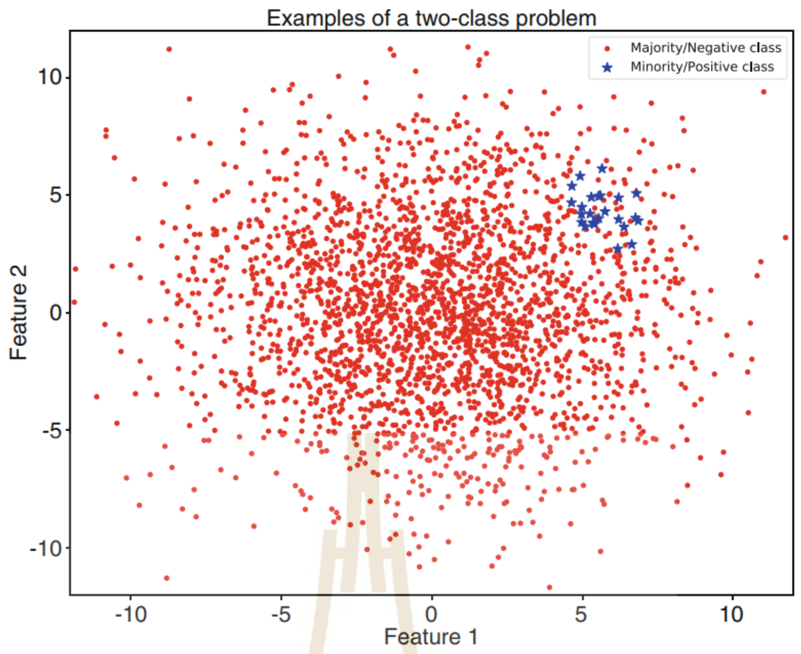


Figure 2.1 Example of a two-class problem (Fernández et al., 2018).

entiate between benign and malignant tumors. It uses two different features that were measured after the biopsy. In this case, the correct identification of malignant tumors was more important than benign tumors, because the consequences of an undetected malignant tumor can be fatal. Usually, the number of people with malignant tumors in real life is much lower than of those with benign tumors. The direct modeling of this data set could potentially predict malignant tumors as benign tumors, which could result in delayed treatment and even death (Fernández et al., 2018).

Accuracy is no longer a proper measure in the imbalance scenario because it does not distinguish between the numbers of correctly classified examples of different classes. Hence, it may lead to erroneous conclusions that are inaccurate if it classifies all examples as negatives (majority classes). Therefore, more informative measures in this context are required to assess the quality of the models, for instance, geometric mean, F-measure, precision, recall, etc. These metrics will be discussed later.

2.2 Imbalance Data Techniques

This section will discuss various techniques to handle class imbalance when training a robust and well-fit machine learning model.

2.2.1 Oversampling Methods

Oversampling methods duplicate samples in the minority class or synthesize new samples from the samples in the minority class. This is also called upsampling. Oversampling is also divided into two types: Random Oversampling and Informative Oversampling (Sonak and Patankar, 2015). Random Oversampling is the method that balances the class distribution by replicating randomly chosen minority class samples. On the other hand, the Informative Oversampling method synthetically generates minority class samples based on a pre-specified criterion (Ramyachitra and Manikandan, 2014).

2.2.2 Undersampling Methods

Undersampling is a frequently used and efficient method for balancing data. This method uses a subset of the majority class to train the classifier. In undersampling, one deletes some samples of the majority class. Undersampling methods are also divided into Random Undersampling and Informative Undersampling. Random Undersampling is simple, it randomly eliminates samples from the majority class till the data set gets balanced. The Informative Undersampling method selects only the required majority class samples based on a pre-specified selection criterion to balance the data set (Sonak and Patankar, 2015).

Random Undersampling

Random undersampling randomly selectes examples from the majority class and deletes them from the training dataset. This method will keep the information of the minority class but will reduce the size of the majority class, until class balance. However, if vast quantities of data discarded, this can be highly problematic, as the loss of such data can make the decision boundary between minority and majority instances harder to learn, resulting in a loss in classification performance (He and Ma, 2013).

Cluster centroid

This is another method of undersampling the majority class by replacing a cluster of majority samples with the cluster centroid of a K-means algorithm. This algorithm separates the majority class into K clusters, and replaces the majority class with the centroids of these clusters.

The K-means algorithm or K-means clustering is one of the simplest and most popular unsupervised machine learning algorithms. To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. Given a set of observations $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$, the K-means algorithm aims to cluster the n observations into K $(\leqq n)$ sets $\mathbf{S} = \{S_1, S_2, S_3, ..., S_K\}$ in which each cluster has a centroid $\mathbf{c}_k$, where $\mathbf{c}_k \in \mathbb{R}^d$. An objective function for this clustering can be created by finding the minimum value of the total distance of the samples and the centroid of each cluster $\mathbf{c}_k$ follows:

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik} \|\mathbf{x}_i - \mathbf{c}_k\|^2, \tag{2.1}$$

where $r_{ik} \in \{0, 1\}$ is a variable that indicates the membership of the i-th sample in the k-th cluster. That is,

$$r_{ik} = \begin{cases} 1, & \text{if } k = \arg\min_j \|\mathbf{x}_i - \mathbf{c}_j\|^2, \\ 0, & \text{if } k \neq \arg\min_j \|\mathbf{x}_i - \mathbf{c}_j\|^2. \end{cases} \tag{2.2}$$

This means that the sum of the values r_{ik} is 1 or $\sum_{k=1}^{K} r_{ik} = 1$ for each $i = 1, ..., n$.

An optimal $\mathbf{c}_k$ can be obtained by setting the partial derivative of J concerning $\mathbf{c}_k$ is 0 as follows:

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{c}_k} &= 2\sum_{i=1}^{n} r_{ik}(\mathbf{x}_i - \mathbf{c}_k) = 0, \\ \sum_{i=1}^{n} r_{ik}\mathbf{c}_k &= \sum_{i=1}^{n} r_{ik}\mathbf{x}_i, \\ \mathbf{c}_k \sum_{i=1}^{n} r_{ik} &= \sum_{i=1}^{n} r_{ik}\mathbf{x}_i, \\ \mathbf{c}_k &= \frac{\sum_{i=1}^{n} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{n} r_{ik}}. \end{aligned} \tag{2.3}$$

It can be seen that the divisor or $\sum_{i=1}^{n} r_{ik}$ is the total number of samples assigned to the k-th cluster, and $\mathbf{c}_k$ is the mean of all samples assigned to the k-th cluster.

Near–Miss

Near-miss is an algorithm that can help in balancing an imbalanced dataset. It can be grouped under undersampling algorithms and is an efficient way to balance the data. The algorithm does this by looking at the class distribution and randomly eliminating samples from the majority class depending on their distance from elements of the minority class (Madhukar, 2020).

This algorithm mainly uses distance finding. The most common and easiest way to find distances is the Euclidean distance, which is defined as follows:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}, \tag{2.4}$$

where $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$ and $\mathbf{y} = (y_1, y_2, y_3, ..., y_n)$

There are 3 types of near-miss algorithms:

• Version 1: For each sample from the majority class, find the three closest samples in the minority class and compute the average distance from these three. Then sort the elements of the majority class by this average distance, and choose the N samples of smallest average distance as the new majority class. Here, N is the size of the minority class.

• Version 2: Similar to version 1, but find the three farthest samples from the minority class and compute the average distance from these three.

• Version 3: For each element of the minority class, find the K closest element of the majority class and compute the average distance, (fixed K), using only the majority samples and choose the N samples of the largest average distance as the new majority class (Mani and Zhang, 2003).

2.2.3 Hybrid Methods

The disadvantages of undersampling and oversampling are data loss and over-fitting, respectively. Therefore, a hybrid method has been developed combining under-sampling and oversampling for resolving the previously mentioned problem (He and Ma, 2013).

2.3 Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) (Mirjalili and Lewis, 2016) is a new meta-heuristic optimization algorithm which mimics the foraging of humpback whales. The whales hunt schools of krill or small fishes close to the surface by swimming around them within a shrinking circle and creating distinctive bubbles along a circle or '9'-shaped path (see Figure 2.2).

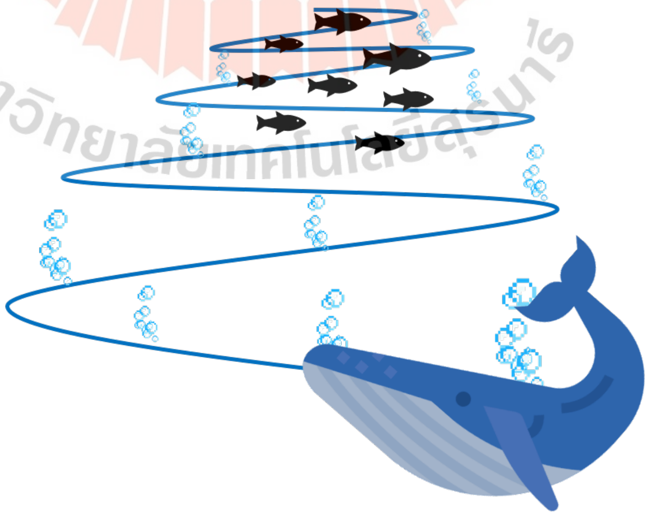


Figure 2.2 Bubble-net feeding of humpback whales.

The hunting consists of two phases, which may alternate. In the first phase, the exploration phase, they randomly search for prey. In the second phase, the exploitation phase, they circle around or close in on the prey in a spiralling motion. The circling and spiralling motions may alternate. As the whales approach their prey, the first phase, the exploration phase become less and less frequent.

The following subsections discuss the mathematics of each phase in our model in detail. Not all implementations of the algorithm are exactly as described, there can also be simplifications or modifications.

2.3.1 The mathematical model

Consider a function $f(\mathbf{x})$ defined on a bounded subset D of $\mathbb{R}^d$, called the fitness function. The goal is to find a maximizer (or minimizer) $\mathbf{x}_0$ of $f(\mathbf{x})$. One begins with a predetermined number of points $\mathbf{x}_1, ..., \mathbf{x}_m$ in D which represent the positions of the participating whales, more generally called search agents. Through an iterative process, these positions are updated in a way so that they approach the maximizer $\mathbf{x}_0$. *MaxIter* denotes the total number of iterations allowed, which is the stopping criterion.

As the m whales operate independently of another, we give the description for an individual whale. To be consistent with the notation in (Mirjalili and Lewis, 2016), let $\vec{X}_j(t)$ denote the position of the j-th whale at the t-th iteration. That is, $\vec{X}_j(0) = \mathbf{x}_j$ $(j = 1, ..., n)$. $\vec{X}^*(t)$ will indicate the best position obtained so far, i.e. the position derived from all m whales since the beginning until the current iteration that gives the largest value of the fitness function.

During the exploitation phase of the given whale, the switch between circling and spiralling is determined by random variable $p = p(t, j)$. The switch between the exploitation phase and the exploration phase is being determined by another random variable $A = A(t, j)$. The manner in which the circling phase, the spiralling phase and the exploration phase are stitched together can be seen from equation (2.12) below.

We now describe the progession from the t-th iteration to the t+1-th iteration.

Iteration step initialization. First we establish some variables that change with each iteration. Set

$$a = a(t) = 2\left(1 - \frac{t}{\textit{MaxIter}}\right). \tag{2.5}$$

This value decreases linearly from 2 to 0 with each iteration. Next let $p = p(t, j)$ and $r = r(t, j)$ be two uniformly distributed random variables in [0,1]. Set

$$A = A(t, j) = a(2r - 1) \quad \text{and} \quad C = C(t, j) = 2r.$$

These are two uniformly distributed random variables with values in the interval $[-a(t), a(t)]$, respectively $[0, 2]$.

Exploitation Phase – circle motion. To hunt a prey, humpback whales first encircle it. Equations (2.6) and (2.7) can be used to mathematically model this behaviour,

$$\vec{D} = \vec{D}(t, j) = |C\vec{X}^*(t) - \vec{X}_j(t)|, \tag{2.6}$$

$$\vec{X}_j(t+1) = \vec{X}^*(t) - A\vec{D}, \tag{2.7}$$

where $|\cdot|$ denotes the elementwise absolute value.

Equation (2.7) shows that the search agents (whales) update their positions to move around or come closer to the position of the so far best known solution (prey), as controlled by the values of A and C. The shinking-encircling behaviour is achieved by the decreasing value of the parameter a.

Exploitation Phase – bubble-net attacking spiralling motion. To simulate spiral-shaped motion, the above is modified to

$$\vec{D}' = \vec{D}'(t, j) = |\vec{X}^*(t) - \vec{X}_j(t)|, \tag{2.8}$$

$$\vec{X}_j(t+1) = \vec{X}^*(t) + \vec{D}' e^{bl} \cos(2\pi l), \tag{2.9}$$

where $l = l(t, j)$, l is a uniformly distributed random number in $[-1, 1]$, and b is a constant defining the shape of the spiral.

Exploration Phase. Instead of updating the positions of the search agents according to the so far best known position, a random search is performed. First one choses any of the other whales' position $\vec{X}_{rand}$ by random and sets

$$\vec{D}'' = \vec{D}''(t,j) = |C\vec{X}_{rand}(t) - \vec{X}_j(t)|, \tag{2.10}$$

$$\vec{X}_j(t+1) = \vec{X}_{rand}(t) - A\vec{D}''. \tag{2.11}$$

Stitching the phase together. The update of the position of the search agent now follows the rule

$$\vec{X}_j(t+1) = \begin{cases} \text{Shrinking encircling (equation (2.7))} & \text{if } p < 0.5 \text{ and } |A| < 1; \\ \text{bubble-net attacking (equation (2.9))} & \text{if } p \geq 0.5; \\ \text{exploration phase (equation (2.11))} & \text{if } p < 0.5 \text{ and } |A| \geq 1. \end{cases} \tag{2.12}$$

Exploration will no longer take place when more than half the maximal iterations have passed, as then $|A| < 1$. We observe that all operations in equations (2.6)–(2.11) are componentwise.

Figure 2.3 shows the workflow of the WOA algorithm. It may be seen that WOA creates a random, initial population, and evaluates it using a fitness function once the optimization process starts. After finding the best solution, the algorithm repeatedly executes the following steps until the end criterion is satisfied. Firstly, the main coefficients are updated. Secondly, a random value is generated. Based on this random value, the algorithm updates the position of a solution using either equations (2.7), (2.11) or (2.9). Thirdly, the solutions are prevented from going outside the search landscape. Finally, the algorithm returns the best solution obtained as an approximation of the global optimum.

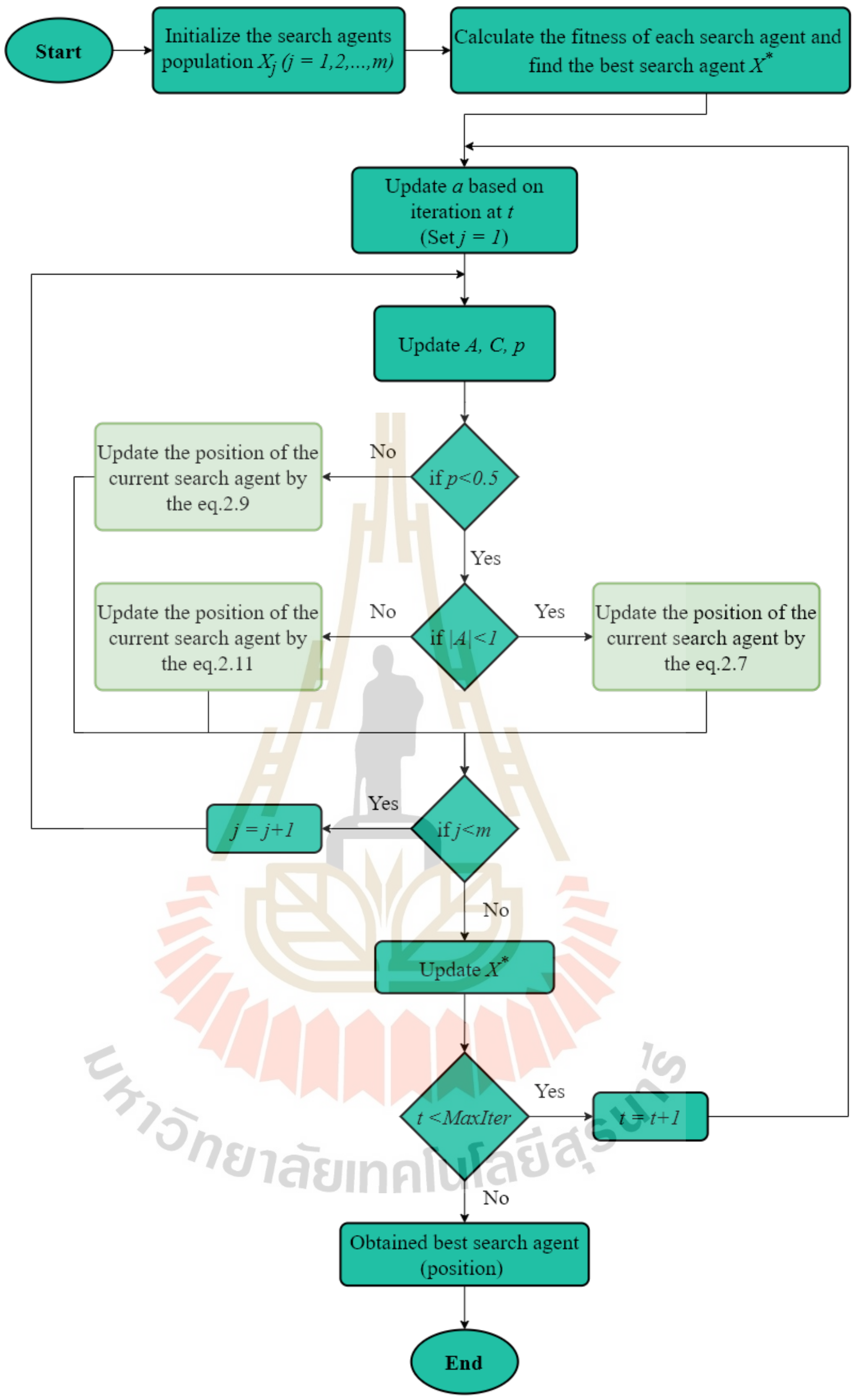

Start
Initialize the search agents population X_j ($j = 1,2,\ldots,m$)
Calculate the fitness of each search agent and find the best search agent X^*
Update a based on iteration at t (Set $j = 1$)
Update A, C, p
if $p<0.5$
No
Yes
Update the position of the current search agent by the eq.2.9
if $|A|<1$
No
Yes
Update the position of the current search agent by the eq.2.11
Update the position of the current search agent by the eq.2.7
if $j<m$
Yes
$j = j+1$
No
Update X^*
$t < MaxIter$
Yes
$t = t+1$
No
Obtained best search agent (position)
End

Figure 2.3 Flowchart of Whale Optimization Algorithm.

2.4 Binary Whale Optimization Algorithm

The binary whale optimization algorithm (BWOA) (Kumar and Kumar, 2020) was developed from the whale optimization algorithm (WOA) to be able to find solutions with only binary vectors. That is, the domain of the fitness function is no longer a subset of $\mathbb{R}^d$ but a space $\mathcal{X}$ of binary vectors,

$$\mathcal{X} = \{0,1\}^d = \prod_{i=1}^{d} \{0,1\}.$$

The fact that the vector components are only 0 and 1 has many applications, such as feature selection (Hussien, Hassanien, Houssein, Bhattacharyya and Amin, 2019) or unit commitment (Kumar and Kumar, 2020). The updating of a search agent's position in (2.7), (2.9) and (2.11) changes now to the toggling of individual bits. The toggling is decided on by first changing the distances $\vec{D}$, $\vec{D}'$, and $\vec{D}''$ to elements in $[0,1]$ through a sigmoid transfer function, and then comparing them with a random number. To be precise, the transfer function is

$$g(s) = \frac{1}{1 + e^{-10(s-0.5)}}.$$

Note that as s increases from -2 to 2, then $g(s)$ increases from $\frac{1}{1+e^{25}} \approx 0$ to $\frac{1}{1+e^{-15}} \approx 1$.

Let $\vec{X}_j(t,i)$ denote the i-th component of $\vec{X}_j(t)$. This component is now updated as follows: Let $r = r_j(t,i)$ be a random variable uniformly distributed in $[0,1]$. Then

$$\vec{X}_j(t+1,i) = \begin{cases} 1 - \vec{X}_j(t,i), & \text{if } r \leq g(A\vec{D}_0(i)); \\ \vec{X}_j(t,i), & \text{else;} \end{cases}$$

where

$$\vec{D}_0 = \begin{cases} \vec{D} & \text{if } p < 0.5 \text{ and } |A| < 1; \\ \vec{D}' & \text{if } p \geq 0.5; \\ \vec{D}'' & \text{if } p < 0.5 \text{ and } |A| \geq 1. \end{cases} \tag{2.13}$$

2.5 Machine Learning

Machine learning (ML) is the operation of a computer system that uses the data for learning by itself with the aim to detect relations within the data by computer. It uses programmed algorithms that receive and analyze input data to predict output values within an acceptable range. As new data is fed to these algorithms, they learn and optimize their operations to improve performance, developing 'intelligence' over time.

ML is separated into 4 categories which are supervised learning, unsupervised learning, semi-supervised, and reinforcement.

Supervised Learning is a popular method in machine learning. The operator provides the machine learning algorithm with a known dataset that includes desired inputs and outputs, and the algorithm must find a method to determine how to arrive at those inputs and outputs. While the operator knows the correct answers to the problem, the algorithm identifies patterns in data, learns from observations, and makes predictions. Incorrect predictions are corrected by the operator and this process continues until the algorithm achieves a high level of accuracy/performance. Supervised learning can solve regression, classification, and forecasting problems (Wakefield, 2022).

2.5.1 Model for Classification

In this section, the models for classification, which will all be supervised learning models, are discussed: Support Vector Machine, Decision Tree, Random Forest, and K-Nearest Neighbors.

Support Vector Machine

Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. It is a method for the classification of both linear and nonlinear data. If the data can not separated by a hyperplane, then one maps it into another vector space of large dimension where it can be separated. This leads to a kernel function, representing the inner product in the large vector space.

To explain the SVM, let's first look at the simplest case of two a class problems where the classes are linearly separable. Let the training data set D be given as $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), ..., (\mathbf{x}_l, y_l)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is the vector attribute of training dataset with associated class labels, y_i for $i = 1, 2, 3, , ..., l$. Each y_i can take one of two values, either 1 or -1. If all the examples in D can be separated exactly by the hyperplane $w \cdot x + b = 0$ and the distance from the nearest sample point of ecah class to the hyperplane is the maximum, we state that the data samples can be separated by the optimal hyperplane, which is also called the maximum margin hyperplane as shown in (Figure 2.4) (He and Ma, 2013).

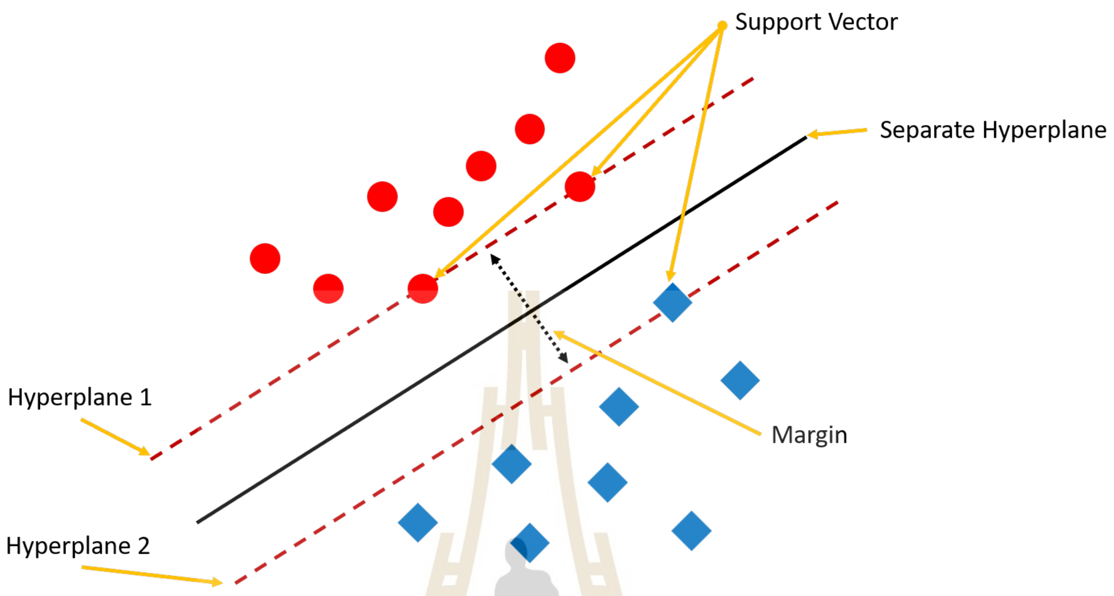


Figure 2.4 The margin of a hyperplane.

The problem of the optimal classification hyperplane is transformed into the following optimization problem by

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i;$$

$$y_i(w\cdot x + b) - 1 + \xi_i \geq 0;$$

$$\xi_i \geq 0, = 1, 2, 3, ..., l,$$

in which C is the penalty parameter, which controls the degree of penalty for misclassification samples. In addition, the greater the value of C, the greater the penalty for misclassification. The corresponding Lagrangian function is

$$L(w, b, \xi, \alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i(y_i(w\cdot\varphi(x_i)+b)-1+\xi_i) - \sum_{i=1}^{l}\beta_i\xi_i, \quad (2.14)$$

where α_i, β_i are Lagrangian multipliers and $\alpha_i > 0, \beta_i > 0$. We can obtain the following dual problem by

$$\min \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i,x_j) - \sum_{i=1}^{l}\alpha_i;$$

$$s.t. \sum_{i=1}^{l}\alpha_i y_i = 0;$$

$$0 \leq \alpha_i \leq C, i = 1,2,...,l, w \in \mathbb{R}^n, b \in \mathbb{R},$$

where $k(x_i, x_j)$ is the kernel function (Xie, Liang, Dong, Tan, and Zhang, 2019).

Decision Tree

Decision Tree is supervised learning suitable for solving regression and classification problems. In 1984, a group of statisticians published the book Classification and Regression Trees (CART) (Breiman et al., 1984), which described how a binary decision trees work. It can produce either classification or regression trees, depending on whether the dependent variable is a numeric or a category, respectively. It uses the Gini index and twoing criteria as an impurity measure for selecting attributes. In 1986, Iterative Dichiotomister (ID3) was proposed by Quinlan (Quinlan, 1986), which uses the entropy and information gain to choose the attribute in each node. In 1993, C4.5 was developed by Quinlan again (Quinlan, 1993), which is an extension from ID3 and became a benchmark to which newer supervised learning algorithms are frequently compared. Since a decision tree can handle noisy data and many independent variables using the If-Else rule, a decision tree is easy to interpret.

The components of a decision tree are nodes and branches. A branch represents the outcome of the node or the values of the attributes. The node on the top is called the root node, there is only one such root node, and there is a unique path from the root node to any other node. The remaining nodes are called the internal nodes, except for the leaf nodes, which represent the classes or the output of the model (Sá et al., 2011) shown in Figure 2.5.

From all of the above, it can be seen that there are different versions of the decision trees and each form will also use various splitting criteria. There are many measures of splitting that can be used to decide the best way to split the node. The splitting criteria

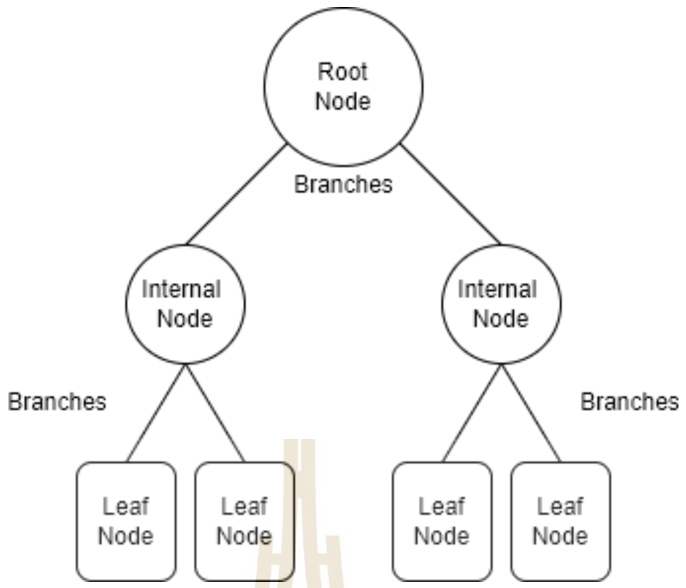


Figure 2.5 The components of a Decision Tree.

for the decision tree are as follows: Gini index, twoing criteria, entropy, information gain, and gain ratio (Singh and Gupta, 2014). However, in this study of decision trees, we be interested in the Gini index as the only splitting criterion.

Consider a multi-class having set D given by $(\mathbf{x}_1, C_1), (\mathbf{x}_2, C_2), (\mathbf{x}_3, C_3), ..., (\mathbf{x}_l,$ $C_l)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is the vector attributes and C_i the class label. Suppose there are q classes, $C_i \in \{1, ..., q\}$. Let d_j be the number of data samples in class j $(j = 1, ..., q)$ The nodes in the tree are constructed in the following order; Consider any attribute A. If A has k different values, then split the sample set by attribute A into k subset $A_1, A_2, A_3, ..., A_k$. If d_{ij} is the number of samples in A_i whose class is j, the Gini impurity is

$$\text{Gini impurity } (A_i) = 1 - \sum_{j=1}^{n} p_j^2 = 1 - \sum_{j=1}^{n} \left(\frac{d_{ij}}{|A_i|} \right)^2 \tag{2.15}$$

where p_j is the probability of class j. The Gini index measures the frequency at which any element of the dataset will be mislabeled when it is randomly labeled. In two class problems, the maximum value of the Gini impurity is 0.5 when the probability of two classes is the same (shown in equation (2.16)). Furthermore, its minimum value is 0 as shown in equation (2.17). It can occur when the node is pure, which means that all the contained components in the node are of one unique class. Therefore, this node can no longer be split (Aznar, 2020):

$$\text{Gini impurity}_{max} = 1 - (0.5^2 + 0.5^2) = 0.5, \tag{2.16}$$

$$\text{Gini impurity}_{min} = 1 - (1^2) = 0. \tag{2.17}$$

The Gini index is the weighted sum of Gini impurities based on the corresponding fraction of the category in the attribute. The formula is

$$\text{Gini index }(A) = \sum_{i=1}^{k} \frac{|A_i| \times \text{Gini impurity }(A_i)}{|A_1| + |A_2| + |A_3| + \cdots + |A_k|}. \tag{2.18}$$

The attribute A that has the lowest index value, is chosen as the next node in the tree.

Random Forest

Random forest is an ensemble learning method for regression, classification, and other tasks that operates by creating multiple decision trees at training time. For classification tasks, the output of class prediction will choose the class with the most votes (Majority Vote) and becomes our model's prediction (Wikipedia, 2022) (see figure 2.6).

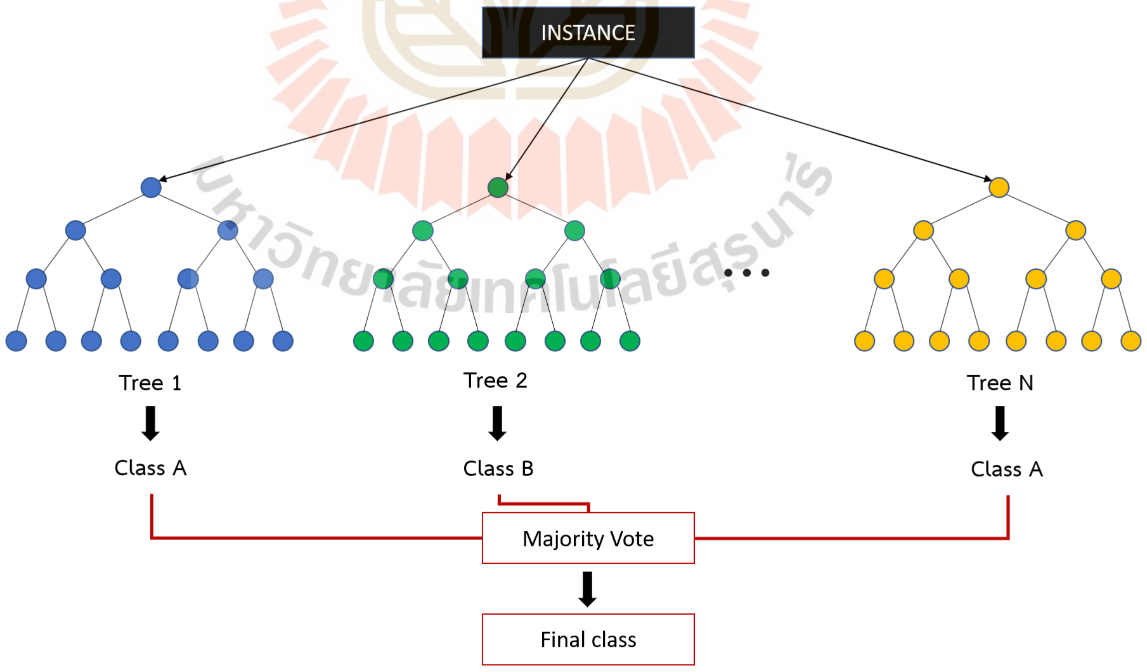


Figure 2.6 Majority vote of Random Forest.

K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised learning algorithm and a non-parametric classification algorithm. It is known for its simplicity and effectiveness (Taunk, De, Verma, and Swetapadma, 2019).

For a new instance, predictions are made by searching the entire training set for the K closest neighbors and summarizing the output variable for those K cases. The factors that affect the performance of KNN are the value of K, the distance metric chosen, and the normalization of the parameters. To understand the detailed working of the algorithm, the steps are as follows:

Given the training dataset: $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_l$ where $\mathbf{x}_i = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, ..., x_n^{(i)})$, l is the number of training data, n is number of features of each data sample.

Step1: Store the training set. Normalize and store the training data set.

Step2: For each new unlabeled data instance $\mathbf{y} = (y_1, y_2, y_3, ..., y_n)$,

2.1 Calculate the distance from all training data points using the formula:

$$\text{dist}(\mathbf{x}_j, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i^{(j)} - y_i|^p \right)^{1/p} . \tag{2.19}$$

Equation (2.19) is an equation in general form for finding distance, which is called Minkowski distance. Minkowski distance is typically used with p being 1 or 2, which correspond to the Manhattan distance and the Euclidean distance, respectively. However, the most popular distance used in the KNN method is the Euclidean distance.

2.2 Find the K - nearest neighbors

2.3 Assign the class containing the maximum number of the K-nearest neighbors. The result of the classification is sensitive to the value of K. The input variable K decides the number of neighbors that must be considered. The value of K affects the algorithm as using the K value we can build the boundaries of each class (Taunk et al, 2019).

Figure 2.7 shows a conceptual example of the KNN algorithm. From the figure, it can be seen that the yellow circle is the new data point. If K = 3, then this yellow circle will be labeled as a blue triangle because among the 3 closest neighbors, there are two blue triangles but only one red circle. Similarly, if K = 5, then this yellow circle will be labeled as a red circle.

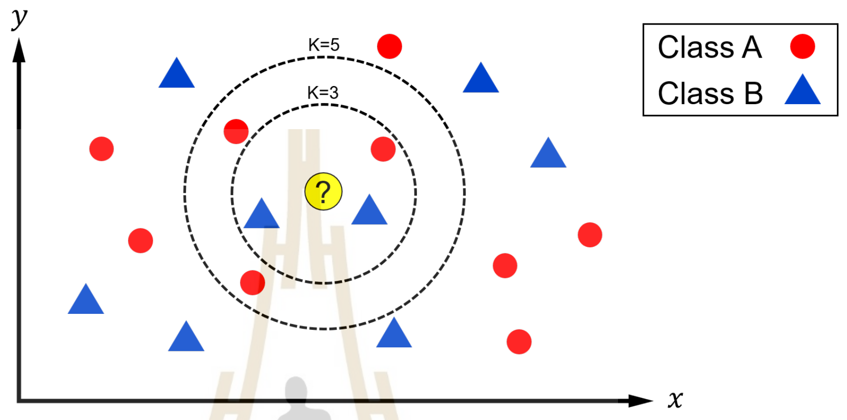


Figure 2.7 KNN algorithm example.

2.6 Performance Metrics

In this section, the performance measurement of the classification models is discussed. Various performance metrics will be considered, as follows.

2.6.1 Confusion Matrix

A confusion matrix is a table that visualizes and summarizes the performance of a classification algorithm (Fernández et al., 2018). A confusion matrix is shown in Figure 2.8.

The entries in the confusion matrix are defined as follows:

1. True positive (TP) is the number of elements in the positive class that are correctly predicted as positive;

2. True negative (TN) is the number of elements in the negative class that are correctly predicted as negative;

		Prediction	
		Negative (0)	Positive (1)
Actual	Negative (0)	True Negative (TN)	False Positive (FP)
	Positive (1)	False Negative (FN)	True Positive (TP)

Figure 2.8 Confusion matrix.

3. False positive (FP) is the number of elements in the negative class that are wrongly predicted as positive;

4. False negative (FN) is the number of elements in the positive class that are wrongly predicted as negative.

Common performance metrics of a classification algorithm are accuracy, precision, recall, and F1 score, which are calculated on the basis of the above-stated TP, TN, FP, and FN.

Accuracy

Accuracy is the ratio of correctly classified samples. It is calculated from the ratio of the correct predicted number to the total number,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.20}$$

Precision

Precision is the ratio of correct predictions for positive to the total number of positive predictions only,

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{2.21}$$

Recall

Recall, also called sensitivity, is defined as the ratio of the total number of correctly classified positive examples divided by the total number of positive examples,

$$\text{Recall (or Sensitivity)} = \frac{TP}{TP+FN}. \tag{2.22}$$

Specificity

Specificity is similar to Recall but focuses only on negative classes. Specificity is defined as the ratio of the total number of correctly classified negative samples divided by the total number of negative samples,

$$\text{Specificity} = \frac{TN}{TN+FP}. \tag{2.23}$$

F1 score

F1 score or F measure is also a measure of the test's accuracy. It is defined as a weighted mean of precision and recall. It has its maximum value at 1 and worst at 0 (Fernández et al., 2018),

$$\text{F1 score} = \frac{2 \times (Precision \times Recall)}{Precision + Recall}. \tag{2.24}$$

G-mean

The Geometric Mean (G-Mean) is a metric that measures the balance between classification performances on both the majority and minority classes. A low G-Mean is an indication of poor performance in the classification of the positive cases even if the negative cases are correctly classified as such. This measure is important in the avoidance of overfitting the negative class and underfitting the positive class (Akosa, 2017),

$$\text{G-mean} = \sqrt{Sensitivity \times Specificity}. \tag{2.25}$$

2.6.2 Mathew's Coefficient

The Matthew's correlation coefficient (MCC) is least influenced by imbalanced data. It is a correlation coefficient between the observed and predicted classifications. The value ranges from -1 to +1 with a value of +1 representing a perfect prediction, 0 as no better than a random prediction, and -1 the worst possible prediction. The calculation formula is shown in Equation (2.26) (Chicco and Jurman, 2020),

$$\text{MCC} = \frac{(TN \times TP) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \quad (2.26)$$

2.6.3 Cohen’s Kappa Coefficient

Cohen’s kappa is a metric often used to assess the agreement between two raters. It can also be used to assess the performance of a classification model. The calculation formula is shown in Equation (2.27) (Cohen, 1960),

$$k = \frac{p_a - p_e}{1 - p_e} = 1 - \frac{1 - p_a}{1 - p_e}. \quad (2.27)$$

where p_a is the relative observed agreement among raters shown in Equation (2.28) and p_e is the hypothetical probability of chance agreement shown in equation (2.29),

$$p_a = \frac{TP + TN}{TP + TN + FP + FN}, \quad (2.28)$$

(which is accuracy), and

$$p_e = \frac{(TN + FP)(TN + FN) + (FN + TP)(FP + TP)}{(TP + TN + FP + FN)^2}. \quad (2.29)$$

In a similar fashion to the MCC, kappa takes on values from -1 to +1, with a value of 1 indicating perfect concordance of the model prediction and the actual classes. A value of −1 indicates total disagreement between prediction and the actual classes, and a value of 0 meaning there is no agreement between the actual and classified classes (Akosa, 2017).

2.6.4 Receiver operating characteristic curve (ROC Curve)

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. A ROC graph is a plot of False Positive Rate (FPR) on the x-axis, and True Positive Rate (TPR) on the y-axis shown in Figure 2.9 (Fawcett, 2006).

True Positive Rate (TPR) is defined as follows:

$$\text{True Positive Rate (TPR or Sensitivity)} = \frac{TP}{TP + FN}; \quad (2.30)$$

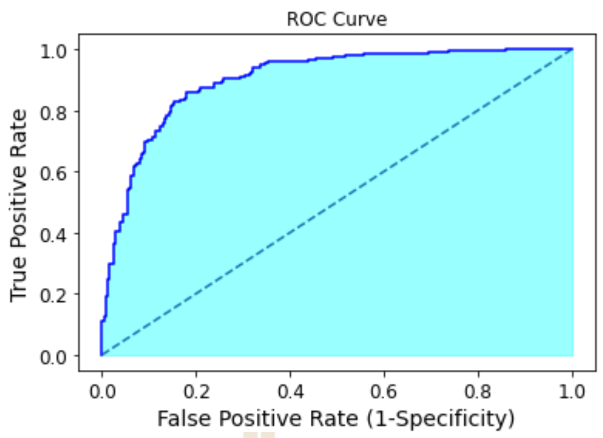


Figure 2.9 ROC Curve.

False Positive Rate (FPR) is defined as follows:

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}; \tag{2.31}$$

A good classifier should reach as close to the top left corner as possible. This corner corresponds to perfect classification. The upward diagonal indicates random performance. Ideally, all points in the ROC curve should lie above this diagonal, as points below the diagonal indicate performance worse than random. The lower-right corner corresponds to a classifier that always predicts the wrong class. The lower-left corner (origin) corresponds to always predicting the negative class, while the top right corner corresponds to always predicting the positive class. Figure 2.10 shows the different levels of model performance shown through a ROC graph.

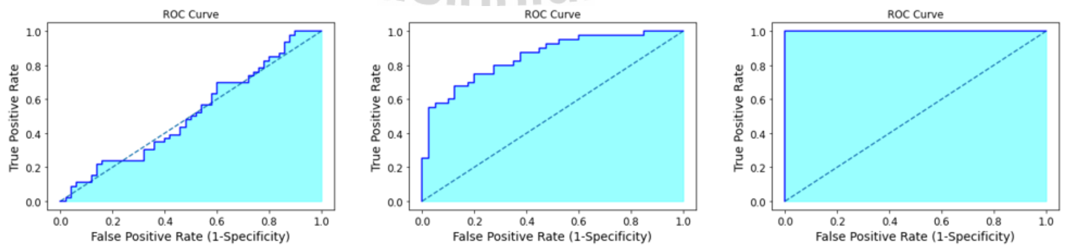


Figure 2.10 Performance of Model (left: Bad, middle: Good, right: Perfect).

However, if we look at the performance of the model through the graph with the naked eye, it can be inconvenient to measure it. Therefore, there is one way to quantify how well the classification model does at classifying data is to calculate the area under the ROC curve.

The value for Area Under the ROC Curve (AUROC) ranges from 0 to 1. A model that has an AUROC of 1 is able to perfectly classify observations into classes while a model that has an AUROC of 0.5 does no better than a model that performs random guessing.

2.6.5 Precision-Recall curve (PR Curve)

The precision-recall curve shows the tradeoff between precision and recall for different thresholds. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall). The process of drawing the PR Curve is similar to ROC Curve but uses in the x-axis the recall, and precision in the y-axis is shown in Figure 2.11. PR curves are often used in information retrieval, and focus only on the positive class (scikit-learn, 2022).

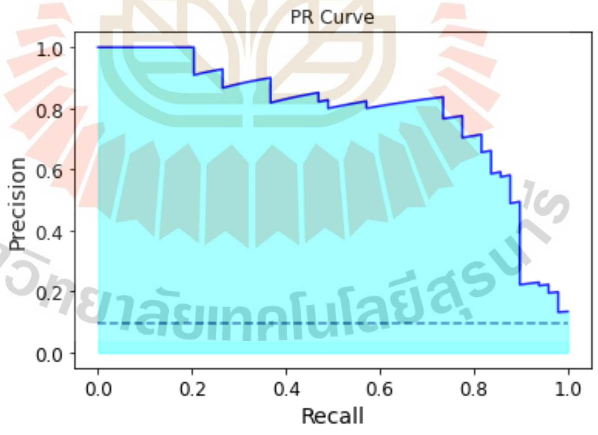


Figure 2.11 PR Curve.

The interpretation of the PR Curve is slightly different from the ROC Curve. Good classifiers should be as close as possible to the top right, as this corner represents the best trade-off between precision and recall. The baseline of the PR Curve is determined by the ratio of positive examples in the dataset. The value for Area Under the PR Curve

(AUPRC) ranges from 0 to 1 (Sofaer, Hoeting and Jarnevich, 2019).

2.6.6 K-fold Cross-Validation

K-fold Cross-validation is a very popular technique for machine learning models. The workflow is to divide the sample data into k partitions (or folds), then use $k-1$ of the partitions for training and the k-th for testing. After that, this procedure is repeated $k-1$ times, rotating the test set. The expected performance metrics (accuracy, recall, precision, or other appropriate metrics) are determined based on the results across the iterations.

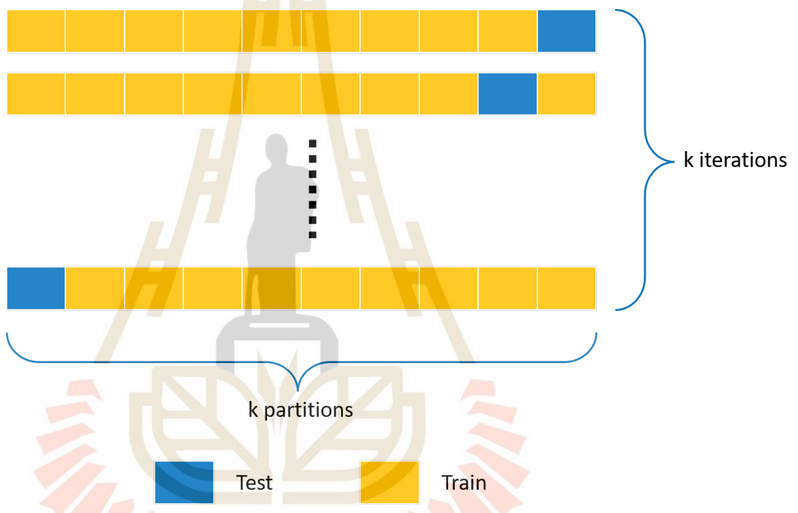


Figure 2.12 K-fold Cross-validation.

2.6.7 Standard competition ranking (”1224” ranking)

In competition ranking, items that compare equally receive the same ranking number, and then a gap is left in the ranking numbers. The number of ranking numbers that are left out in this gap is one less than the number of items that are compared equally. Equivalently, each item’s ranking number is one plus the number of items ranked above it. This ranking strategy is frequently adopted for competitions, as it means that if two (or more) competitors tie for a position in the ranking, the position of all those ranked below them is unaffected (i.e., a competitor only comes second if exactly one person scores better than them, third if exactly two people score better than them, fourth if exactly three people score better than them, etc.) (Wikipedia, 2021). Therefore, in this thesis, standard competition rankings are used to compare the efficiency of each algorithm, and determine which is the most efficient.

To compare the performance of different resampling methods, we use the ranking of resampling methods on each dataset and compute the average rank of each resampling method R_j as follows (Huang, Zhao, Zhu, Chen, and Broucke, 2020):

$$R_j = \frac{1}{m}\sum_{i=1}^{m} r_{ij}, \tag{2.32}$$

where m is the total number of datasets, r_{ij} is the rank of the j-th resampling method on the i-th dataset.

2.7 Related Researches

Mafarja and Mirjalili (2017) presented two hybridization models used to design different feature selection techniques based on Whale Optimization Algorithm (WOA). In the first model, a simulated Annealing (SA) algorithm is embedded in the WOA algorithm, while it is used to improve the best solution found after each iteration of WOA algorithm in the second model. The performance of the proposed approaches is evaluated on 18 standard benchmark datasets from the UCI repository and compared with three well-known wrapper feature selection methods in the literature. The experimental results confirm the efficiency of the proposed approaches in improving the classification accuracy compared to other wrapper-base algorithms, which ensures the ability of WOA algorithm in searching the feature space and selecting the most informative attributes for classification tasks.

Hussien, Hassanien, Houssein, Bhattacharyya and Amin (2019) present a novel binary version of the Whale Optimization Algorithm (BWOA), to select the optimal feature subset for dimensionality reduction and classification problems. The new approach is based on a sigmoid transfer function (S-shape). By dealing with the feature selection problem, a free position of the whale must be transformed to the corresponding binary solutions. KNN classifier is applied to ensure the selected features are the relevant ones. A set of criteria are used to evaluate and compare the proposed BWOA-S with the native one over 11 different datasets. The results showed that the new algorithm has a significant performance in finding the optimal feature.

Kumar and Kumar (2020) modified the WOA to the BWOA, by binarizing the solution vectors and using a sigmoidal transfer function is to update the position of whales. The performance of the proposed algorithm is evaluated on 29 benchmark functions. Furthermore, an unpaired t-test is carried out to illustrate its statistical significance. The experimental results depict that the proposed algorithm outperforms others in respect of benchmark test functions. The proposed approach is applied to an electrical engineering problem, a real-life application, named ‘‘unit commitment.’’ Experimental results reveal that the proposed approach is superior to other algorithms in terms of lower production costs.

Sayed, Darwish, and Hassanien (2020) presented a hybrid intelligence model that uses cluster analysis algorithms with bio-inspired algorithms as feature selection for analyzing clinical breast cancer data. A binary version of both moth flame optimization and WOA is proposed. Two evaluation criteria are adopted to evaluate the proposed algorithms: clustering-based measurements and statistics-based measurements. The experimental results positively demonstrate the capability of the proposed bio-inspired feature selection algorithms to produce both meaningful data partitions and significant feature subsets.

Hussien, Hassanien, Houssein, Amin, and Azar (2020) improved the original version of the WOA for handling binary optimization problems. For this purpose, two transfer functions (S-shaped and V-shaped) are presented to map a continuous search space to a binary one. To illustrate the functionality and performance of the proposed BWOA, its results when applied on 22 benchmark functions, 3 engineering optimization problems, and a real-world traveling salesman problem are found. Furthermore, the proposed BWOA is compared with five well-known metaheuristic algorithms. The experimental results show its superiority in comparison with other state-of-the-art metaheuristics in terms of accuracy and speed.

Yu, Ni and Zhao (2013) proposed ACOSampling which is a novel undersampling method based on the idea of ant colony optimization (ACO) to address this problem. First, the original training data set is randomly and repeatedly divided into two groups: training data set and validation data set. Then, for each partition, ACOSampling is performed to find the subset of the corresponding optimal majority class examples. They evaluated the method on four benchmarks skewed DNA microarray datasets by support vector machine (SVM) classifier, showing that the proposed method outperforms many other sampling approaches, which indicates its superiority. The fitness function used in ACOsampling is:

$$fitness = (\alpha \times F\ measure) + (\beta \times G\ mean) + (\gamma \times AUC),$$

where $\alpha + \beta + \gamma = 1$

López, Triguero, Carmona, García and Herrera (2014) proposed the usage of the Iterative Instance Adjustment for Imbalanced Domains (IPADE-ID) algorithm. It is an evolutionary framework, which uses an instance generation technique, designed to face the existing imbalance modifying the original training set. The method iteratively learns the appropriate number of examples that represent the classes and their particular positioning. The learning process contains three key operations in its design: a customized initialization procedure, an evolutionary optimization of the positioning of the examples, and a selection of the most representative examples for each class. An experimental analysis is carried out with a wide range of highly imbalanced datasets over the proposal and recognized solutions to the problem. The results obtained, which have been contrasted through nonparametric statistical tests, show that their proposal outperforms previously proposed methods. The fitness fuction used in IPADE-ID corresponding fitness value is measured as the AUCRC.

Kim, Jo and Shin (2016) suggested an optimization approach of cluster-based undersampling to select appropriate instances. This approach can solve the data imbalance problem. They examined the effectiveness of a hybrid method using a clustering technique and genetic algorithms based on the artificial neural networks model to balance the proportion between the minority class and the majority class. The proposed method is successfully applied to the bankruptcy prediction problem using financial data for which the proportion of small and medium-sized bankruptcy firms in the manufacturing industry is extremely small compared to that of non-bankruptcy firms. They use the G-Mean as the fitness function in GA for data balancing.

Li et al. (2017) presented Adaptive Swarm Balancing Algorithms, which lead to significant efficiency and effectiveness improvements on large datasets. They also find it more consistent with the practice of the typical large imbalanced medical datasets. The proposed methods lead to more credible performances of the classifier and shorten the run time compared to the brute-force method. The fitness function of this work involves accuracy and kappa.

CHAPTER III

RESEARCH METHODOLOGY

In this study, we created two versions of the proposed undersampling method: in the first version is fixed-parameter $K = 1$ which we called WBWOA 1NN, and the second version can adjustable parameter K we called WBWOA KNN. In this section, we will explain the methodology of this thesis. Its content includes the tools, datasets, the WBWOA 1NN algorithm, the WBWOA KNN algorithm, and the work procedures.

3.1 Tools

The computer program used in this research is the Python language version 3.7.5 to develop the algorithms by using the library sklearn, pandas, matplotlib, imblearn, optuna, and numpy packages.

This research used a laptop, CPU version i5-9300H 2.40GHz, 16 GB memory operating system Windows 11 Home 64 bit.

3.2 Datasets

We have selected 10 datasets from KEEL and imbalanced-learn that represent a variety of imbalance ratios, as detailed in Table 3.1, in order to compare our proposed undersampling method with the random undersampling, cluster centroid, and near-miss methods.

In this research, we split each data set into two sets, the training and the testing datasets, at a ratio of $80 : 20$. After splitting both, the training and the testing set still had a similar imbalance ratio as the original datasets. The details of the training and testing datasets are shown in Tables 3.2 and 3.3, respectively.

Table 3.1 Detail of datasets.

Dataset name	Attributes	Size	Minority size	Majority size	IR
glass1	9	214	76	138	1.82
iris0	4	150	50	100	2.00
glass-0-1-2-3_vs_4-5-6	9	214	51	163	3.20
ecoli2	7	336	52	284	5.46
ecoli	7	336	35	301	8.60
abalone	10	4177	391	3786	9.68
libras_move	90	360	24	336	14.00
solar_flare_m0	32	1389	68	1321	19.43
yeast_m2	8	1484	51	1433	28.10
mammography	6	11183	260	10923	42.01

Table 3.2 Training datasets.

Dataset name	Attributes	Size	Minority size	Majority size	IR
glass1	9	171	61	110	1.80
iris0	4	120	40	80	2.00
glass-0-1-2-3_vs_4-5-6	9	171	41	130	3.17
ecoli2	7	268	41	227	5.54
ecoli	7	268	28	240	8.57
abalone	10	3341	313	3028	9.67
libras_move	90	288	19	269	14.16
solar_flare_m0	32	1111	54	1057	19.57
yeast_m2	8	1187	41	1146	27.95
mammography	6	8946	208	8738	42.01

Table 3.3 Testing datasets.

Dataset name	Attributes	Size	Minority size	Majority size	IR
glass1	9	43	15	28	1.87
iris0	4	30	10	20	2.00
glass-0-1-2-3_vs_4-5-6	9	43	10	33	3.30
ecoli2	7	68	11	57	5.18
ecoli	7	68	7	61	8.71
abalone	10	836	78	758	9.72
libras_move	90	72	5	67	13.40
solar_flare_m0	32	278	14	264	18.86
yeast_m2	8	297	10	287	28.70
mammography	6	2237	52	2185	42.02

3.3 Range of Optimized Parameter

The ranges of the parameters to be optimized in each of the 3 classifiers, the support vector machine, decision tree, and random forest are shown in Table 3.4.

Table 3.4 Range of parameters to be optimized

Model	Parameter	Type/Interval
Decision tree	criterion	Gini index
	max_depth	$[1, 100]$
	min_samples_split	$[2, 100]$
Random forest	criterion	Gini index
	n_estimators	$[2, 500]$
	max_depth	$[1, 500]$
Support vector machine	kernel	radial basis function (rbf)
	C	$[1, 70]$
	gamma	$[1 \times 10^{-6}, 1]$

3.4 WBWOA 1NN Algorithm

We now describe the proposed WBWOA 1NN algorithm in mathematical terms. Let D be a given dataset. Split D into the majority class D^- and the minority class D^+, and let d and n^+ denote the number of samples in each class: $d = |D^-|$ and $n^+ = |D^+|$. When the data is highly unbalanced, then $d \gg n^+$.

The objective of our undersampling algorithm is to find a subset D^-_{red} of the majority class D^- with $|D^-_{red}| \approx |D^+|$ while at the same time giving best performance for a chosen classifier, when $D^-_{red} \cup D^+$ is the training data.

The performance metric which we choose is of the form

$$f = f(A) := (1 - \text{F1 score})^2 + (1 - \text{AUROC})^2 + (1 - \text{sensitivity})^2 + \beta(n^- - n^+)^2 \quad (3.1)$$

where $A \subseteq D^-$ is a given subset of the majority class, $n^- = n^-(A) = |A|$ is the number of samples in A, β is a non-negative parameter, and F1 score, sensitivity and AUROC are obtained through 10-fold cross-validation of the chosen classifier using the dataset $A \cup D^+$. The parameter β influences how well the two datasets should be balanced. In this manner we obtain a function

$$f : 2^{D^-} \to [0, \infty)$$

defined on the power set 2^{D^-} of D^- which is to be minimized.

Observe that after fixing a labeling of the samples in the majority class, $D^- = \{\mathsf{x}_1, \ldots, \mathsf{x}_d\}$, then there is a natural bijection

$$\Phi : \mathcal{X} \to 2^{D^-}$$

define $\mathcal{X}$ is

$$\mathcal{X} := \{0, 1\}^d$$

given by

$$\Phi(\vec{X}) = \{\mathsf{x}_i \in D^- : \vec{X}(i) = 1\} \quad (i = 1, ..., d)$$

That is, every binary vector $\vec{X}$ of length d uniqely determines a subset of the majority class according to the vector components which are equal to one. Composition thus gives a function

$$f \circ \Phi : \mathcal{X} \to [0, \infty)$$

to be minimized. Since the domain of this function is a space of binary vectors, the BWOA is a natural candidate for finding a minimizer of $f \circ \Phi$ as fitness function, in particular, since this algorithm has shown to be fairly efficient in applications. Furterhermore, in order to keep computation time low, we choose the K-nearest-neighbors method with $K = 1$ as a simple classifier.

3.5 WBWOA KNN Algorithm

This is a modification to the WBWOA 1NN algorithm. The difference is that instead of the one-nearest neighbor classifier now a K-nearest neighbor classifier is used where K itself is a parameter to be optimized. The parameter K is also chosen by the WOA. The optimal K value is an integer in the interval of 1 to 30, while the WOA deals with real numbers. Therefore, before applying a K-nearest neighbor algorithm, the value of K in the WOA must change to an integer. Therefore, the fitness function also depends on K,

$$f = f(A, K) := (1 - \text{F1 score})^2 + (1 - \text{AUROC})^2 + (1 - \text{sensitivity})^2 + \beta(n^- - n^+)^2 \quad (3.2)$$

where K is a given parameter of K-nearest neighbor. We thus obtain a function

$$f : 2^{D^-} \times \{1, ..., 30\} \to [0, \infty)$$

Composition thus gives a function

$$f \circ (\Phi, r_d) : \mathcal{X} \times [1, 30] \to [0, \infty)$$

to be minimized, where r_d denotes the rounding to an integer function.

3.6 Model Evaluation

To evaluate the performance of the various classification for models we used the following 9 performance metrics: Accuracy, F1 score, G-mean, Area under the ROC curve (AUROC), Area Under the PR Curve (AUPRC), Sensitivity, Precision, Mathew's Coefficient (MCC), and Cohen's Kappa Coefficient (kappa).

3.7 Work procedure

All tests proceeded as follows:

1. First split the given data set into the training and testing datasets at a ratio of $80 : 20$.
2. Next split the training data set further into minority class D^+ and majority class D^-.
3. Obtain a reduced majority class D^-_{red} using any of the five undersampling methods, while the minority class remains D^+.
   (a) In case of the WBWOA 1NN, we find a minimizing binary vector $\vec{X}^*$ of $f \circ \Phi$ and the new reduced majority class is then $D^-_{red} = \{\mathbf{x}_i \in D^- : \vec{X}^*(i) = 1\}$.
   (b) In case of the WBWOA KNN, we find a minimizing vector $\vec{X}^* = (x_1, x_2, x_3, ...x_d, \mathcal{K})$ of $f \circ (\Phi, r_d)$. The BWOA is used to update the position $x_1, x_2, x_3, ...x_d$, and the new reduced majority class is then $D^-_{red} = \{\mathbf{x}_i \in D^- : \vec{X}^*(i) = 1\} \quad (i = 1, ..., d)$. The WOA is used to update the position of $\mathcal{K}$, and the new parameter $\mathcal{K} \in \{1, ..., 30\}$.

   We used 20 whales (search agents) and 1000 iterations. We also choose $\beta = 100$ to obtain balanced datasets. The criterion to stopping process of WBOA 1NN and WBOA KNN are

   - If the fitness value is zero.
   - If the best fitness value remains unchanged for 350 iterations.
   - If the 1000 iterations have been completed maximum iteration.
4. Train decision tree, support vector machine, and random forest models with $D^-_{red} \cup D^+$ as data, using 10-fold validation for parameter optimization, and F1 score as performance metric.
5. Evaluate performance using the testing data. An outline of the workflow is shown in Figure 3.1

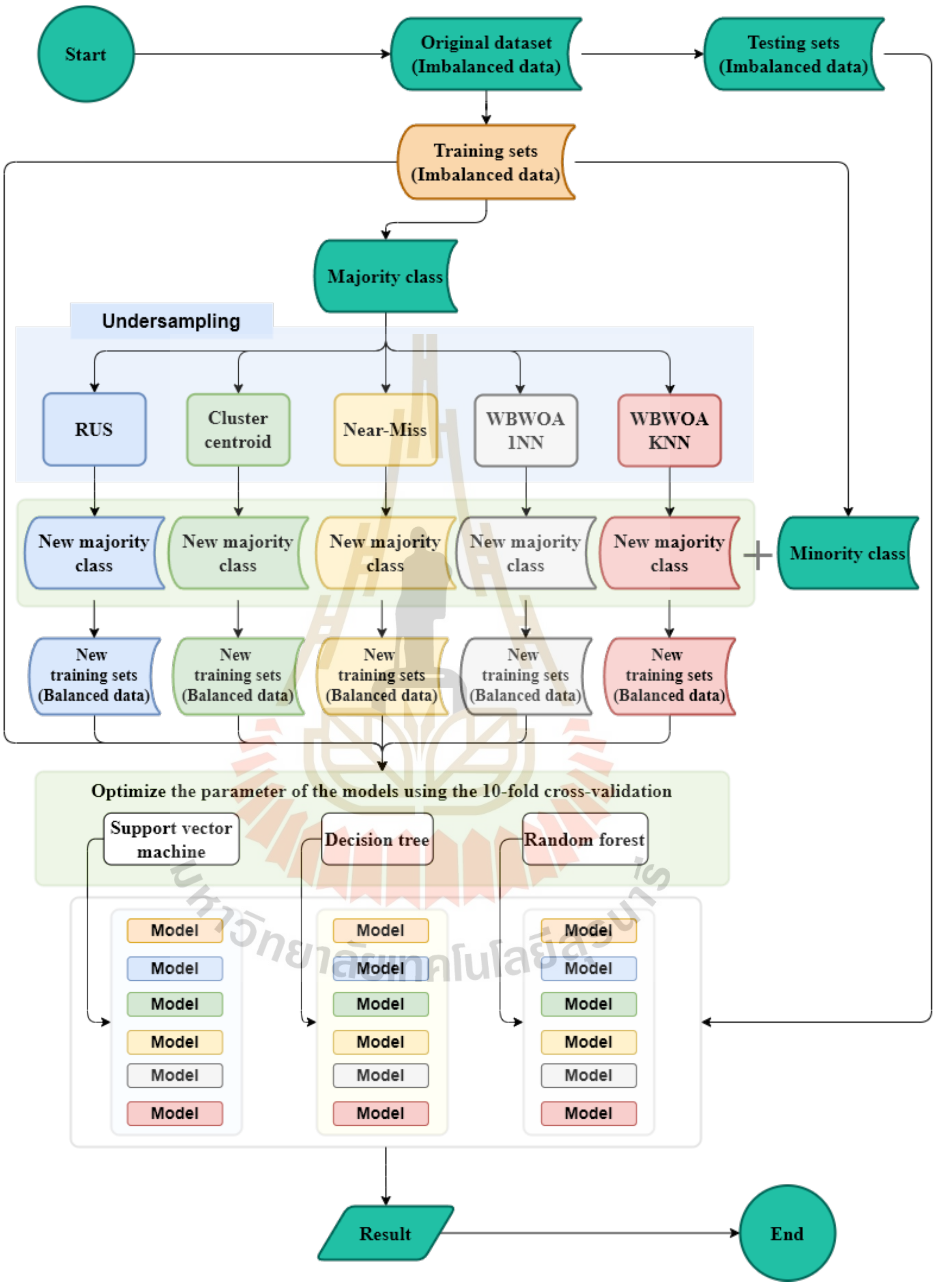

Start
Original dataset (Imbalanced data)
Testing sets (Imbalanced data)
Training sets (Imbalanced data)
Majority class
Undersampling
RUS
Cluster centroid
Near-Miss
WBWOA 1NN
WBWOA KNN
New majority class
Minority class
New training sets (Balanced data)
Optimize the parameter of the models using the 10-fold cross-validation
Support vector machine
Decision tree
Random forest
Model
Result
End

Figure 3.1 Outline of the workflow.

CHAPTER IV

RESULTS AND DISCUSSION

In this section, we would present the performance of 5 different undersampling methods. We test the performance of undersampling methods by ten datasets. We use three models to measure their performance: decision tree, random forest, and support vector machine. There are 9 performance metrics listed: Accuracy, F1 score, G-mean, Area under the ROC curve (AUROC), Area Under the PR Curve (AUPRC), Sensitivity, Precision, Mathew's Coefficient (MCC), and Cohen's Kappa Coefficient (kappa). However, we also test the original dataset without using any undersampling methods. In the discussion section, we will carefully discuss the performance of undersampling.

4.1 Results

The results of finding the best subsets of the majority class samples for WBWOA 1NN and WBWOA KNN can be expressed in terms of fitness values. If fitness values are closer to zero, it means that the subset is a good representation of the majority class samples. The fitness values and fitness graphs can be found in Appendix D.

The performance measurements of the decision tree, random forest, and support vector machine models using the testing data are displayed in Table A.1, Table A.2, and Table A.3, respectively and can be found in the Appendix A.

The ranking score results of the decision tree, random forest, and support vector machine models are shown in Table B.1, Table B.2, and Table B.3, respectively and can be found in Appendix B. This ranking score is based on standard competition ranking.

Furthermore, the optimized parameter of each model and the best F1 score of each model are shown in Appendix C.

4.2 Discussion

To simplify the analysis, we will first use the ranking score of tables B.1-B.3 (appendix B) to calculate the average ranking score. The average ranking scores by undersampling method for each, the decision tree, random forest, and support vector machine, are shown in tables 4.1, 4.2, and 4.3, respectively.

Remark: The symbols ***, **, and * that appear in tables 4.1-4.3 mean that the ranking score comes first place, second place, and third place, respectively.

Table 4.1 Average ranking score of each undersampling method in the decision tree model.

Metric	Undersampling Method					
	None	Cluster centriod	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
Accuracy	1.3***	3.9	4.4	3.0**	3.2*	3.0**
F1 score	3.2	3.5	4.4	2.4***	2.5**	2.9*
G-mean	4.0	3.1	4.7	2.3**	2.0***	3.0*
AUROC	3.1*	3.2	4.8	2.7**	2.1***	3.4
AUPRC	3.2*	3.4	3.9	3.2*	2.7***	2.9**
Sensitivity	4.5	2.3*	2.5	2.2**	2.0***	2.3*
Precision	1.2***	3.9	4.7	3.1*	3.0**	3.2
MCC	3.2**	3.4	4.4	2.4***	2.4***	3.3*
kappa	2.2***	3.6	4.6	2.7*	2.6**	3.4

Table 4.1 shows that the WBWOA 1NN undersampling method, under the performance metrics G-mean, AUROC, AUPRC, MCC, and sensitivity, has the best (lowest) ranking. Although precision and kappa are not the best rankings, they are still better than the ranking of the random undersampling method. The F1 score and accuracy metrics of random undersampling and the WBWOA 1NN are not much different because they have nearly ranking scores. However, accuracy is usually high in imbalanced datasets, which causes precision also to be high, they are inappropriate metrics. It is therefore not that

surprising accuracy and precision have decreased after undersampling.

Therefore, the WBWOA 1NN undersampling method obtains excellent performance when constructing the decision tree model compared with random undersampling, cluster centroid, near-miss, and WBWOA KNN undersampling methods.

Table 4.2 Average ranking score of each undersampling method in the random forest model.

Metric	Undersampling Method					
	None	Cluster centriod	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
Accuracy	1.2***	3.5	5.2	2.8*	3.3	2.5**
F1 score	2.2***	3.2	5.2	2.8*	3.2	2.5**
G-mean	4.7	2.9	4.8	2.0**	2.8*	1.9***
AUROC	2.9**	3.3	4.5	2.5***	2.9**	3.2*
AUPRC	2.2***	3.6	4.7	2.8**	2.8**	3.4*
Sensitivity	5.2	2.6*	2.3**	2.3**	2.3**	1.6***
Precision	1.2***	3.6	5.4	2.7**	3.4	2.8*
MCC	2.0***	3.2	5.1	2.8*	3.4	2.6**
kappa	2.0***	3.1	5.2	2.8*	3.4	2.6**

The results of table 4.2 show not using any undersampling technique gains excellent results in several performance metrics like accuracy, F1 score, AUPRC, precision, MCC, and kappa. However, if we did not use any technique for imbalanced data, the model would be unable to classify minority class samples at all, causing a high (poor) ranking score in sensitivity and G-mean. Thus, we will only compare the performance of the undersampling methods. Observing the results of the average ranking score found that WBWOA KNN obtains excellent performance more often than random undersampling, i.e. accuracy, F1 score, G-mean, sensitivity, MCC, and kappa. Although AUROC and AUPRC scores of random undersampling are better than of WBWOA KNN, we also select the WBWOA KNN undersampling method for the random forest model.

Therefore, the WBWOA KNN undersampling method obtains excellent overall performance when constructing the random forest model compared with random undersampling, cluster centroid, near-miss, and WBWOA 1NN undersampling methods.

Table 4.3 Average ranking score of each undersampling method in the support vector machine model.

Metric	Undersampling Method					
	None	Cluster centriod	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
Accuracy	1.5***	3.7	5.1	2.5**	2.7*	3.4
F1 score	2.7**	3.4	4.9	2.7**	2.4***	3.0*
G-mean	4.0	3.1	4.8	2.6*	2.1***	2.5**
AUROC	4.0	3.4	4.9	1.7***	2.1**	2.9*
AUPRC	2.5***	3.5	4.7	2.5***	2.7**	3.3*
Sensitivity	4.4	2.1*	2.5	2.2	1.5***	2.0**
Precision	2.5***	3.4*	5.0	2.5***	2.5***	3.2**
MCC	2.6**	3.3	5.0	2.7*	2.4***	3.1
kappa	2.4***	3.4	5.0	2.8**	2.4***	3.1*

The results of table 4.3 show that the WBWOA 1NN undersampling method, under the performance metrics F1 score, G-mean, sensitivity, precision, MCC, and kappa metrics, has the best ranking. The accuracy, AUROC, and AUPRC metrics of random undersampling and the WBWOA 1NN are not considerably different because of close ranking scores.

Therefore, for the support vector machine model, WBWOA 1NN undersampling method provides excellent performance when compared with random undersampling, cluster centroid, near-miss, and WBWOA KNN undersampling methods.

The results from tables 4.1, 4.2, and 4.3 show that each classification model has a different best undersampling method. Next, we will analyze which classification model could give the highest performance when using the best-suited undersampling method for that model. The best undersampling methods for decision tree, random forest, and support vector machine are WBWOA 1NN, WBWOA KNN, and WBWOA 1NN, respectively. Table 4.4 lists the average ranking scores for each model. We can see that the random forest model with WBWOA KNN undersampling method has the highest overall performance, followed by the support vector machine model with WBWOA 1NN undersampling, and the last one is the decision tree model with WBWOA 1NN undersampling.

Table 4.4 Average ranking score of 3 classification models, using the best undersampling method for each model.

Measurement	WBWOA 1NN (decision tree)	WBWOA KNN (random forest)	WBWOA 1NN (SVM)
Accuracy	2.1	1.6	1.8
F1 score	2.3	1.5	1.8
G-mean	2.4	1.5	1.7
AUROC	2.7	1.5	1.5
AUPRC	2.0	1.9	1.8
Sensitivity	2.2	1.4	1.2
Precision	2.1	1.6	1.9
MCC	2.2	1.6	1.8
kappa	2.2	1.6	1.8

Even though the WBWOA KNN algorithm combined with random forest has the highest average ranking score, when averaging all nine performance metrics as shown in Table 4.5, we found that the WBWOA 1NN algorithm combined with support vector machine had a higher average performance metrics score than WBWOA KNN algorithm combined with random forest for all the metrics. This is because the metric value of the WBWOA 1NN algorithm may be higher than the WBWOA KNN algorithm in some measurements. However, if measured from the average ranking score of the WBWOA KNN algorithm by random forest is still the best overall performant undersampling method.

Table 4.5 Average performance of 3 classification models, using the best undersampling method for each model.

Measurement	WBWOA 1NN (decision tree)	WBWOA KNN (random forest)	WBWOA 1NN (SVM)
Accuracy	0.8113	0.8387	0.8504
F1 score	0.5140	0.5783	0.5960
G-mean	0.8419	0.8794	0.8908
AUROC	0.8667	0.9212	0.9312
AUPRC	0.6385	0.6457	0.6499
Sensitivity	0.8776	0.9399	0.9499
Precision	0.4323	0.4703	0.4927
MCC	0.5039	0.5705	0.5958
kappa	0.4430	0.5123	0.5391

In addition, although the algorithm that we have developed has excellent overall performance, its effective implementation may require consideration of the suitability of the dataset. In fact, this algorithm is compute-intensive and may require long computation time for large datasets.

CHAPTER V

CONCLUSION

This thesis has studied how to solve imbalanced data problems based on the undersampling method. We developed a novel undersampling method that applied the whale and the binary whale optimization algorithms to cooperate with the K-nearest neighbor algorithm. In this study, we created two versions of the proposed undersampling method: in the first version is fixed-parameter $K = 1$ which we called WBWOA 1NN, and the second version can adjustable parameter K we called WBWOA KNN. We selected ten datasets from the KEEL and imbalanced-learn repositories to evaluate the performance of the proposed algorithm. These datasets have varying imbalance ratios ranging from 1.82 to 42.01, and have binary classes. We choose other undersampling methods to compare with our proposed undersampling method namely the random undersampling, cluster centroid, and near-miss methods. When data had been balanced by several undersampling methods, it was used to traine a decision tree, random forest, and support vector machine model using 10-fold validation for parameter optimization, and using the F1 score as the performance metric. And it was tested for performance with a testing dataset with nine performance metrics: accuracy, F1 score, G-mean, AUROC, AUPRC, sensitivity, precision, MCC, and kappa.

The results of this thesis found that the WBWOA KNN algorithm applied to the random forest model has the highest overall performance, followed by the WBWOA 1NN by support vector machine model, and the last one is WBWOA 1NN by a decision tree model. The efficiency average measurement results of WBWOA KNN by random forest were as follows: Accuracy = 0.8387, F1 score = 0.5783, G-mean = 0.8794, AUROC = 0.9212, AUPRC = 0.6457, Sensitivity = 0.9399, Precision = 0.4703, MCC = 0.5705, and Kappa = 0.5123. This shows that our proposed undersampling method, is effective in dealing with an imbalanced data problem. The highlight of the proposed undersampling method is its high sensitivity, which is suitable for predicting the minority classes. However, its effective implementation may require consideration of the suitability of the dataset, and trial and error is also an important process for analyzing data.

In the future, the proposed undersampling method can be further developed in many ways. For example, one may develop a parallel process between undersampling and feature selection, the second adapt to a hybrid approach between the undersampling and oversampling methods. Finally, we hope this thesis will be useful to reseachers.

REFERENCES

REFERENCES

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37-66.

Akosa, J. (2017). *Predictive accuracy: A misleading performance measure for highly imbalanced data*. Paper presented at the Proceedings of the SAS Global Forum.

Aznar, P. (2020, 02/12/2020). *Decision Trees : Gini vs Entropy*. https://quantdare.com/decision-trees-gini-vs-entropy/

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and regression trees. Belmont, CA: Wadsworth. *International Group, 432*, 151-166.

Chicco, D., and Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics, 21*(1), 1-13.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*(1), 37-46.

Cortes, C., and Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297.

Deng, N., Tian, Y., and Zhang, C. (2012). *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*: CRC press.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861-874.

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from Imbalanced Data Sets* (Vol. 10): Springer.

Fotouhi, S., Asadi, S., and Kattan, M. W. (2019). A comprehensive data level analysis for cancer diagnosis on imbalanced data. *Journal of Biomedical Informatics, 90*, 103089.

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences, 180*(10), 2044-2064.

He, H., and Ma, Y. (2013). *Imbalanced learning: Foundations, Algorithms, and Applications*.

Huang, L., Zhao, J., Zhu, B., Chen, H., and Broucke, S. V. (2020). An experimental investigation of calibration techniques for imbalanced data. *IEEE Access, 8*, 127343-127352.

Hussien, A. G., Hassanien, A. E., Houssein, E. H., Amin, M., and Azar, A. T. (2020). New binary whale optimization algorithm for discrete optimization problems. *Engineering Optimization, 52*(6), 945-959.

Hussien, A. G., Hassanien, A. E., Houssein, E. H., Bhattacharyya, S., and Amin, M. (2019). *S-shaped binary whale optimization algorithm for feature selection*. In Recent Trends in Signal and Image Processing (pp. 79-87): Springer.

imbalancedlearn. (2020). *fetch_datasets*. https://imbalanced-learn.org/stable/references/generated/imblearn.datasets.fetch_datasets.html

imbalancedlearn. (2022). *ClusterCentroids*. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.ClusterCentroids.html

KEEL. *Imbalanced data sets*. http://www.keel.es/

Kesornsit, W., Lorchirachoonkul, V., and Jitthavech, J. (2018). Imbalanced data problem solving in classification of diabetes patients. *Khon Kaen University Research Journal, 18*(3), 11-21.

Kim, H.-J., Jo, N.-O., and Shin, K.-S. (2016). Optimization of cluster-based evolutionary undersampling for the artificial neural networks in corporate bankruptcy prediction. *Expert Systems with Applications, 59*, 226-234.

Kumar, V., and Kumar, D. (2020). Binary whale optimization algorithm and its application to unit commitment problem. *Neural Computing and Applications, 32*(7), 2095-2123.

Li, J., Liu, L.-s., Fong, S., Wong, R. K., Mohammed, S., Fiaidhi, J., . . . Wong, K. K. (2017). Adaptive swarm balancing algorithms for rare-event prediction in imbalanced healthcare data. *PloS One, 12*(7), e0180830.

López, V., Triguero, I., Carmona, C. J., García, S., and Herrera, F. (2014). Addressing imbalanced classification with instance generation techniques: IPADE-ID. *Neurocomputing, 126*, 15-28.

Madhukar, B. (2020). *Using Near-Miss Algorithm For Imbalanced Datasets*. https://analyticsindiamag.com/using-near-miss-algorithm-for-imbalanced-datasets/

Mafarja, M. M., and Mirjalili, S. (2017). Hybrid whale optimization algorithm with simulated annealing for feature selection. *Neurocomputing, 260*, 302-312.

Mani, I., and Zhang, I. (2003). *kNN approach to unbalanced data distributions: a case study involving information extraction*. Paper presented at the Proceedings of workshop on learning from imbalanced datasets.

Mirjalili, S., and Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software, 95*, 51-67.

Mishra, S. (2017). Handling imbalanced data: SMOTE vs. random undersampling. *International Journal of Managing Information Technology, 4*(8), 317-320.

Mqadi, N. M., Naicker, N., and Adeliyi, T. (2021). Solving Misclassification of the Credit Card Imbalance Problem Using Near Miss. *Mathematical Problems in Engineering, 2021*.

Orriols-Puig, A., and Bernadó-Mansilla, E. (2009). Evolutionary rule-based systems for imbalanced data sets. *Soft Computing, 13*(3), 213-225.

Quinlan, J. (1986). Indroduction of Decision Trees Machine Learning. *Boston (NL): Kluwer Acad. Publ, 1*(86-106), 650.

Quinlan, J. R. (1993). *C4. 5: Programs for Machine Learning*: Morgan Kaufmann Publishers Inc.

Ramyachitra, D., and Manikandan, P. (2014). Imbalanced dataset classification and solutions: a review. *International Journal of Computing and Business Research (IJCBR), 5*(4), 1-29.

Sá, A., Almeida, A., Rocha, B., Mota, M., Souza, J., and Dentel, L. (2011). *Lightning forecast using data mining techniques on hourly evolution of the convective available potential energy*. Paper presented at the Brazilian Congress on Computational Intelligence, Fortaleza, November.

Sayed, G. I., Darwish, A., and Hassanien, A. E. (2020). Binary whale optimization algorithm and binary moth flame optimization with clustering algorithms for clinical breast cancer diagnoses. *Journal of Classification, 37*(1), 66-96.

scikit-learn. (2022). *Precision-Recall*. https://scikit-learn.org/stable/auto-examples/model_selection/plot_precision_recall.html

Singh, S., and Gupta, P. (2014). Comparative study ID3, cart and C4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST), 27*(27), 97-103.

Sofaer, H. R., Hoeting, J. A., and Jarnevich, C. S. (2019). The area under the precision-recall curve as a performance metric for rare binary events. *Methods in Ecology and Evolution, 10*(4), 565-577.

Sonak, A., and Patankar, R. (2015). A survey on methods to handle imbalance dataset. *International Journal of Computer Science and Mobile Computing, 4*(11), 338-343.

Taunk, K., De, S., Verma, S., and Swetapadma, A. (2019). *A brief review of nearest neighbor* *algorithm for learning and classification*. Paper presented at the 2019 International Conference on Intelligent Computing and Control Systems (ICCS).

Wakefield, K. (2022). *A guide to the types of machine learning algorithms and their applications*. https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html

Wikipedia. (2021). *Ranking*. https://en.wikipedia.org/wiki/Ranking

Wikipedia. (2022). *Random forest*. https://en.wikipedia.org/wiki/Random_forest

Yu, H., Ni, J., and Zhao, J. (2013). ACOSampling: An ant colony optimization-based under-sampling method for classifying imbalanced DNA microarray data. *Neurocomputing, 101*, 309-318.

APPENDICES

APPENDIX A

REPORT OF PERFORMANCE MEASUREMENTS

Table A.1 The various performance measurements with the decision tree model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
	Accuracy	0.7674	0.7442	0.5349	0.7209	0.8372	0.6977
	F1 score	0.6154	0.6667	0.5652	0.6842	0.7407	0.6667
	G-mean	0.6901	0.7416	0.5563	0.7464	0.7868	0.7254
	AUROC	0.7881	0.7655	0.6357	0.7512	0.7976	0.7369
glass1	AUPRC	0.7294	0.6937	0.6292	0.5070	0.8081	0.7274
	Sensitivity	0.5333	0.7333	0.8667	0.8667	0.6667	0.8667
	Precision	0.7273	0.6111	0.4194	0.5652	0.8333	0.5417
	MCC	0.4655	0.4669	0.2378	0.4869	0.6325	0.4547
	kappa	0.4543	0.4619	0.1794	0.4534	0.6242	0.4159
	Accuracy	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	F1 score	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	G-mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AUROC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
iris0	AUPRC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Sensitivity	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Precision	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	MCC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	kappa	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Accuracy	0.8605	0.8372	0.8372	0.8605	0.8372	0.8605
	F1 score	0.7273	0.6667	0.6957	0.7273	0.6957	0.7273
	G-mean	0.8385	0.7843	0.8239	0.8385	0.8239	0.8385
	AUROC	0.8394	0.8288	0.8242	0.8333	0.8242	0.8394
glass-0-1-2-3_vs_4-5-6	AUPRC	0.7566	0.7201	0.7309	0.7278	0.7309	0.7566
	Sensitivity	0.8000	0.7000	0.8000	0.8000	0.8000	0.8000
	Precision	0.6667	0.6364	0.6154	0.6667	0.6154	0.6667
	MCC	0.6393	0.5604	0.5965	0.6393	0.5965	0.6393
	kappa	0.6346	0.5593	0.5871	0.6346	0.5871	0.6346
	Accuracy	0.9265	0.9412	0.6618	0.8676	0.9265	0.8235
	F1 score	0.7619	0.8462	0.4651	0.7097	0.8148	0.6471
	G-mean	0.8377	0.9643	0.7471	0.9177	0.9551	0.8885
	AUROC	0.9609	0.9737	0.6013	0.9211	0.9561	0.8596
ecoli2	AUPRC	0.8338	0.8929	0.1566	0.7750	0.8438	0.2391
	Sensitivity	0.7273	1.0000	0.9091	1.0000	1.0000	1.0000
	Precision	0.8000	0.7333	0.3125	0.5500	0.6875	0.4783
	MCC	0.7197	0.8258	0.3859	0.6806	0.7920	0.6145
	kappa	0.7185	0.8108	0.2955	0.6331	0.7709	0.5482

Continuation of Table A.1							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
ecoli	Accuracy	0.9118	0.7941	0.9265	0.8824	0.8235	0.8676
	F1 score	0.5714	0.5000	0.6667	0.6000	0.5385	0.5714
	G-mean	0.7371	0.8778	0.8241	0.8711	0.8963	0.8630
	AUROC	0.9262	0.8852	0.8080	0.9520	0.9016	0.8396
	AUPRC	0.3576	0.6667	0.7551	0.7690	0.6842	0.6322
	Sensitivity	0.5714	1.0000	0.7143	0.8571	1.0000	0.8571
	Precision	0.5714	0.3333	0.6250	0.4615	0.3684	0.4286
	MCC	0.5222	0.5068	0.6273	0.5737	0.5440	0.5456
	kappa	0.5222	0.4087	0.6256	0.5382	0.4567	0.5032
abalone	Accuracy	0.8565	0.7081	0.1567	0.7033	0.7022	0.7069
	F1 score	0.2308	0.3646	0.1132	0.3575	0.3532	0.3570
	G-mean	0.4610	0.7861	0.2558	0.7783	0.7726	0.7756
	AUROC	0.5771	0.8540	0.2290	0.8113	0.8571	0.8120
	AUPRC	0.2618	0.3781	0.0933	0.5636	0.5266	0.5588
	Sensitivity	0.2308	0.8974	0.5769	0.8846	0.8718	0.8718
	Precision	0.2308	0.2288	0.0628	0.2240	0.2215	0.2244
	MCC	0.1516	0.3539	-0.2577	0.3433	0.3358	0.3399
	kappa	0.1516	0.2536	-0.0662	0.2451	0.2402	0.2449
libras_move	Accuracy	0.9583	0.5972	0.7500	0.7361	0.7778	0.8194
	F1 score	0.7273	0.1714	0.3077	0.2963	0.3333	0.3810
	G-mean	0.8810	0.5985	0.7727	0.7649	0.7880	0.8104
	AUROC	0.8851	0.5985	0.7731	0.7657	0.7881	0.8104
	AUPRC	0.7403	0.3639	0.5022	0.4979	0.5122	0.5319
	Sensitivity	0.8000	0.6000	0.8000	0.8000	0.8000	0.8000
	Precision	0.6667	0.1000	0.1905	0.1818	0.2105	0.2500
	MCC	0.7084	0.1016	0.3055	0.2932	0.3323	0.3797
	kappa	0.7049	0.0595	0.2202	0.2065	0.2510	0.3077
solar_flare_m0	Accuracy	0.9353	0.4065	0.3489	0.7806	0.6115	0.6259
	F1 score	0.1000	0.1270	0.1084	0.2078	0.1563	0.1333
	G-mean	0.2647	0.5726	0.5059	0.6726	0.6580	0.5994
	AUROC	0.5764	0.6199	0.6673	0.7055	0.6692	0.6546
	AUPRC	0.1129	0.4665	0.3503	0.3229	0.4084	0.3406
	Sensitivity	0.0714	0.8571	0.7857	0.5714	0.7143	0.5714
	Precision	0.1667	0.0686	0.0582	0.1270	0.0877	0.0755
	MCC	0.0790	0.1086	0.0523	0.1897	0.1424	0.0901
	kappa	0.0720	0.0372	0.0161	0.1366	0.0731	0.0487
	Accuracy	0.9293	0.6734	0.6869	0.7037	0.6835	0.7340
	F1 score	0.1600	0.1709	0.1622	0.1698	0.1754	0.1684

Continuation of Table A.1

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	G-mean	0.4370	0.8136	0.7820	0.7919	0.8200	0.7651
	AUROC	0.5774	0.8310	0.7645	0.8685	0.9206	0.7659
	AUPRC	0.1801	0.5467	0.2534	0.5094	0.3914	0.4504
	Sensitivity	0.2000	1.0000	0.9000	0.9000	1.0000	0.8000
	Precision	0.1333	0.0935	0.0891	0.0938	0.0962	0.0941
	MCC	0.1274	0.2487	0.2206	0.2302	0.2543	0.2122
	kappa	0.1246	0.1165	0.1075	0.1159	0.1215	0.1151
mammography	Accuracy	0.9844	0.4390	0.3317	0.9097	0.9137	0.8534
	F1 score	0.6237	0.0752	0.0639	0.3176	0.3322	0.2264
	G-mean	0.7447	0.6464	0.5569	0.9068	0.9183	0.8867
	AUROC	0.8999	0.9074	0.6485	0.9494	0.9525	0.9013
	AUPRC	0.6441	0.3890	0.5071	0.5554	0.4796	0.5243
	Sensitivity	0.5577	0.9808	0.9808	0.9038	0.9231	0.9231
	Precision	0.7073	0.0391	0.0330	0.1926	0.2025	0.1290
	MCC	0.6203	0.1244	0.0968	0.3933	0.4096	0.3135
	kappa	0.6158	0.0319	0.0198	0.2904	0.3057	0.1935

End of Table

Table A.2 The various performance measurements with the random forest model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	Accuracy	0.8372	0.8372	0.6279	0.7907	0.8140	0.7907
	F1 score	0.7407	0.7586	0.6190	0.7273	0.7647	0.7429
	G-mean	0.7868	0.8092	0.6583	0.7928	0.8252	0.8062
	AUROC	0.8798	0.8917	0.7690	0.8798	0.8929	0.8607
	AUPRC	0.7988	0.8109	0.6692	0.7416	0.8201	0.7505
	Sensitivity	0.6667	0.7333	0.8667	0.8000	0.8667	0.8667
	Precision	0.8333	0.7857	0.4815	0.6667	0.6842	0.6500
	MCC	0.6325	0.6369	0.3615	0.5659	0.6261	0.5892
	kappa	0.6242	0.6360	0.3092	0.5597	0.6143	0.5724
iris0	Accuracy	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	F1score	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	G-mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AUROC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AUPRC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Sensitivity	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Precision	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	MCC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	kappa	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
glass-0-1-2-3_vs_4-5-6	Accuracy	0.8605	0.8605	0.8605	0.8837	0.8372	0.9070
	F1 score	0.7273	0.7273	0.7500	0.7826	0.6957	0.8333
	G-mean	0.8385	0.8385	0.8739	0.8893	0.8239	0.9374
	AUROC	0.9606	0.9667	0.9621	0.9697	0.9636	0.9606
	AUPRC	0.8900	0.9038	0.8987	0.9172	0.9073	0.8860
	Sensitivity	0.8000	0.8000	0.9000	0.9000	0.8000	1.0000
	Precision	0.6667	0.6667	0.6429	0.6923	0.6154	0.7143
	MCC	0.6393	0.6393	0.6748	0.7164	0.5965	0.7923
	kappa	0.6346	0.6346	0.6569	0.7051	0.5871	0.7713
ecoli2	Accuracy	0.9706	0.8382	0.7794	0.9412	0.8676	0.8971
	F1 score	0.9000	0.6207	0.5946	0.8462	0.7097	0.7586
	G-mean	0.9045	0.8301	0.8584	0.9643	0.9177	0.9366
	AUROC	0.9968	0.9226	0.9841	0.9952	1.0000	0.9761
	AUPRC	0.9854	0.7941	0.9257	0.9798	1.0000	0.8927
	Sensitivity	0.8182	0.8182	1.0000	1.0000	1.0000	1.0000
	Precision	1.0000	0.5000	0.4231	0.7333	0.5500	0.6111
	MCC	0.8891	0.5511	0.5583	0.8258	0.6806	0.7322
	kappa	0.8830	0.5254	0.4753	0.8108	0.6331	0.6980

Continuation of Table A.2

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
ecoli	Accuracy	0.9412	0.9118	0.5000	0.8529	0.7647	0.8235
	F1 score	0.6667	0.7000	0.2609	0.5833	0.4667	0.5385
	G-mean	0.7497	0.9495	0.6273	0.9144	0.8589	0.8963
	AUROC	0.9684	0.9742	0.7588	0.9637	0.9403	0.9133
	AUPRC	0.8150	0.7983	0.3138	0.5933	0.5588	0.3692
	Sensitivity	0.5714	1.0000	0.8571	1.0000	1.0000	1.0000
	Precision	0.8000	0.5385	0.1538	0.4118	0.3043	0.3684
	MCCt	0.6462	0.6968	0.1943	0.5867	0.4738	0.5440
	kappa	0.6354	0.6537	0.1046	0.5122	0.3667	0.4567
abalone	Accuracy	0.8720	0.6998	0.1256	0.7057	0.7069	0.7069
	F1 score	0.2190	0.3613	0.0874	0.3594	0.3603	0.3603
	G-mean	0.4256	0.7856	0.2036	0.7798	0.7805	0.7805
	AUROC	0.6821	0.8793	0.1527	0.8750	0.8836	0.8807
	AUPRC	0.2525	0.3617	0.0716	0.5143	0.5311	0.5184
	Sensitivity	0.1923	0.9103	0.4487	0.8846	0.8846	0.8846
	Precision	0.2542	0.2254	0.0484	0.2255	0.2262	0.2262
	MCC	0.1525	0.3531	-0.3904	0.3453	0.3464	0.3464
	kappa	0.1507	0.2490	-0.0975	0.2475	0.2487	0.2487
libras_move	Accuracy	0.9722	0.9306	0.6944	0.8889	0.8889	0.9722
	F1 score	0.7500	0.6154	0.3125	0.5000	0.5000	0.8333
	G-mean	0.7746	0.8673	0.8195	0.8464	0.8464	0.9850
	AUROC	0.9866	0.9478	0.9164	0.9134	0.8910	0.9940
	AUPRC	0.8717	0.8408	0.6033	0.7556	0.7497	0.9381
	Sensitivity	0.6000	0.8000	1.0000	0.8000	0.8000	1.0000
	Precision	1.0000	0.5000	0.1852	0.3636	0.3636	0.7143
	MCC	0.7633	0.5988	0.3527	0.4914	0.4914	0.8324
	kappa	0.7363	0.5794	0.2212	0.4472	0.4472	0.8186
solar_flare_m0	Accuracy	0.9496	0.4281	0.3705	0.6835	0.6115	0.6115
	F1 score	0.3000	0.1405	0.1206	0.1852	0.1692	0.1692
	G-mean	0.4603	0.6106	0.5436	0.6979	0.6879	0.6879
	AUROC	0.6836	0.7091	0.7216	0.7482	0.7041	0.7508
	AUPRC	0.2819	0.1120	0.1909	0.1623	0.1245	0.2216
	Sensitivity	0.2143	0.9286	0.8571	0.7143	0.7857	0.7857
	Precision	0.5000	0.0760	0.0649	0.1064	0.0948	0.0948
	MCC	0.3054	0.1484	0.0936	0.1831	0.1721	0.1721
	kappa	0.2782	0.0523	0.0298	0.1069	0.0872	0.0872
	Accuracy	0.9697	0.7037	0.6936	0.7609	0.7609	0.7643
	F1 score	0.3077	0.1538	0.1495	0.2198	0.2022	0.2045

Continuation of Table A.2

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	G-mean	0.4464	0.7485	0.7429	0.8675	0.8249	0.8268
	AUROC	0.9146	0.8577	0.8650	0.9324	0.9057	0.8972
	AUPRC	0.4901	0.3017	0.1508	0.3922	0.2656	0.1817
	Sensitivity	0.2000	0.8000	0.8000	1.0000	0.9000	0.9000
	Precision	0.6667	0.0851	0.0825	0.1235	0.1139	0.1154
	MCC	0.3545	0.1940	0.1884	0.3048	0.2678	0.2704
	kappa	0.2968	0.0990	0.0942	0.1700	0.1515	0.1541
mammography	Accuracy	0.9866	0.5834	0.3299	0.9204	0.9280	0.9142
	F1 score	0.6341	0.0986	0.0637	0.3597	0.3784	0.3425
	G-mean	0.7065	0.7503	0.5553	0.9403	0.9350	0.9370
	AUROC	0.9827	0.6971	0.6915	0.9796	0.9770	0.9781
	AUPRC	0.7657	0.1110	0.5002	0.7262	0.6823	0.6990
	Sensitivity	0.5000	0.9808	0.9808	0.9615	0.9423	0.9615
	Precision	0.8667	0.0519	0.0329	0.2212	0.2367	0.2083
	MCC	0.6526	0.1684	0.0964	0.4405	0.4524	0.4258
	kappa	0.6278	0.0570	0.0196	0.3346	0.3544	0.3163

End of Table

Table A.3 The various performance measurements with the support vector machine model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	Accuracy	0.8140	0.6512	0.5581	0.7907	0.8837	0.7674
	F1 score	0.7143	0.6154	0.5366	0.7429	0.8485	0.7222
	G-mean	0.7715	0.6761	0.5835	0.8062	0.8944	0.7868
	AUROC	0.8357	0.7500	0.7833	0.8810	0.9429	0.7952
	AUPRC	0.8424	0.6341	0.7820	0.8212	0.9149	0.7173
	Sensitivity	0.6667	0.8000	0.7333	0.8667	0.9333	0.8667
	Precision	0.7692	0.5000	0.4231	0.6500	0.7778	0.6190
	MCC	0.5806	0.3565	0.1926	0.5892	0.7637	0.5539
	kappa	0.5774	0.3260	0.1689	0.5724	0.7554	0.5316
iris0	Accuracy	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	F1 score	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	G-mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AUROC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AUPRC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Sensitivity	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Precision	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	MCC	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	kappa	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
glass-0-1-2-3_vs_4-5-6	Accuracy	0.8372	0.8372	0.8837	0.8837	0.8837	0.9070
	F1 score	0.6667	0.6667	0.8000	0.7826	0.8000	0.8333
	G-mean	0.7843	0.7843	0.9211	0.8893	0.9211	0.9374
	AUROC	0.9273	0.9152	0.9545	0.9576	0.9333	0.9576
	AUPRC	0.8362	0.8197	0.8761	0.8798	0.7376	0.8798
	Sensitivity	0.7000	0.7000	1.0000	0.9000	1.0000	1.0000
	Precision	0.6364	0.6364	0.6667	0.6923	0.6667	0.7143
	MCC	0.5604	0.5604	0.7521	0.7164	0.7521	0.7923
	kappa	0.5593	0.5593	0.7226	0.7051	0.7226	0.7713
ecoli2	Accuracy	0.9853	0.8676	0.7059	0.8676	0.9706	0.8529
	F1 score	0.9565	0.7097	0.5238	0.7097	0.9167	0.6875
	G-mean	0.9912	0.9177	0.8057	0.9177	0.9823	0.9081
	AUROC	1.0000	0.9920	0.9537	0.9904	1.0000	0.9920
	AUPRC	1.0000	0.9639	0.8395	0.9591	1.0000	0.9662
	Sensitivity	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Precision	0.9167	0.5500	0.3548	0.5500	0.8462	0.5238
	MCC	0.9490	0.6806	0.4799	0.6806	0.9036	0.6572

Continuation of Table A.3							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
	kappa	0.9477	0.6331	0.3744	0.6331	0.8990	0.6033
ecoli	Accuracy	0.9412	0.8382	0.7059	0.7941	0.7794	0.7647
	F1 score	0.7500	0.5217	0.3750	0.4615	0.4444	0.4667
	G-mean	0.9028	0.8465	0.7682	0.8213	0.8127	0.8589
	AUROC	0.9321	0.8993	0.8478	0.9625	0.9555	0.9625
	AUPRC	0.5679	0.7967	0.4195	0.8984	0.8583	0.8984
	Sensitivity	0.8571	0.8571	0.8571	0.8571	0.8571	1.0000
	Precision	0.6667	0.3750	0.2400	0.3158	0.3000	0.3043
	MCC	0.7245	0.4966	0.3439	0.4362	0.4186	0.4738
	kappa	0.7173	0.4418	0.2552	0.3662	0.3445	0.3667
abalone	Accuracy	0.9067	0.6675	0.1447	0.7380	0.7213	0.7033
	F1 score	0.0000	0.3505	0.0822	0.3831	0.3916	0.3706
	G-mean	0.0000	0.7827	0.2195	0.7946	0.8184	0.7974
	AUROC	0.6220	0.8791	0.1626	0.8829	0.8763	0.8717
	AUPRC	0.1234	0.3690	0.0533	0.3308	0.3238	0.3087
	Sensitivity	0.0000	0.9615	0.4103	0.8718	0.9615	0.9359
	Precision	0.0000	0.2143	0.0456	0.2455	0.2459	0.2310
	MCC	0.0000	0.3530	-0.3733	0.3683	0.3976	0.3691
	kappa	0.0000	0.2335	-0.1031	0.2780	0.2855	0.2598
libras_move	Accuracy	0.9861	0.9722	0.9306	0.9444	0.9583	0.9583
	F1 score	0.8889	0.8333	0.6154	0.7143	0.7692	0.7692
	G-mean	0.8944	0.9850	0.8673	0.9697	0.9774	0.9774
	AUROC	0.9373	0.9881	0.8448	0.9910	0.9881	0.9851
	AUPRC	0.8352	0.8348	0.7239	0.8931	0.8648	0.8463
	Sensitivity	0.8000	1.0000	0.8000	1.0000	1.0000	1.0000
	Precision	1.0000	0.7143	0.5000	0.5556	0.6250	0.6250
	MCC	0.8878	0.8324	0.5988	0.7228	0.7727	0.7727
	kappa	0.8816	0.8186	0.5794	0.6863	0.7477	0.7477
solar_flare_m0	Accuracy	0.9424	0.5647	0.3273	0.6583	0.5432	0.6187
	F1 score	0.2000	0.1655	0.1137	0.1880	0.1477	0.1719
	G-mean	0.3751	0.6861	0.5065	0.7155	0.6455	0.6922
	AUROC	0.6130	0.6987	0.6427	0.7142	0.7077	0.6901
	AUPRC	0.1270	0.1231	0.0825	0.1106	0.0976	0.0843
	Sensitivity	0.1429	0.8571	0.8571	0.7857	0.7857	0.7857
	Precision	0.3333	0.0916	0.0609	0.1068	0.0815	0.0965
	MCC	0.1922	0.1780	0.0753	0.1980	0.1383	0.1759
	kappa	0.1751	0.0820	0.0218	0.1090	0.0621	0.0903
	Accuracy	0.9630	0.8215	0.8013	0.8653	0.8586	0.8316

Continuation of Table A.3

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	F1 score	0.0000	0.2740	0.2338	0.3103	0.3226	0.2647
	G-mean	0.0000	0.9030	0.8474	0.8819	0.9239	0.8639
	AUROC	0.7892	0.9394	0.9185	0.9345	0.9380	0.9251
	AUPRC	0.2534	0.2465	0.2282	0.2133	0.2145	0.1876
	Sensitivity	0.0000	1.0000	0.9000	0.9000	1.0000	0.9000
	Precision	0.0000	0.1587	0.1343	0.1875	0.1923	0.1552
	MCC	-0.0108	0.3597	0.3012	0.3744	0.4052	0.3318
	kappa	-0.0062	0.2292	0.1861	0.2696	0.2820	0.2199
mammography	Accuracy	0.9866	0.6343	0.3344	0.9276	0.9048	0.8869
	F1 score	0.6341	0.1070	0.0641	0.3864	0.3195	0.2792
	G-mean	0.7065	0.7687	0.5593	0.9532	0.9320	0.9135
	AUROC	0.9082	0.8363	0.6040	0.9768	0.9701	0.9607
	AUPRC	0.7269	0.3978	0.0263	0.4821	0.4878	0.4549
	Sensitivity	0.5000	0.9423	0.9808	0.9808	0.9615	0.9423
	Precision	0.8667	0.0567	0.0331	0.2406	0.1916	0.1639
	MCC	0.6526	0.1762	0.0975	0.4667	0.4060	0.3666
	kappa	0.6278	0.0660	0.0200	0.3626	0.2920	0.2495

End of Table

APPENDIX B

REPORT OF RANKING SCORE

Table B.1 Ranking scores for the decision tree model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	Accuracy	2	3	6	4	1	5
	F1 score	5	3	6	2	1	3
	G-mean	5	3	6	2	1	4
	AUROC	2	3	6	4	1	5
	AUPRC	2	4	5	6	1	3
	Sensitivity	6	4	1	1	5	1
	Precision	2	3	6	4	1	5
	MCC	4	3	6	2	1	5
	kappa	3	2	6	4	1	5
iris0	Accuracy	1	1	1	1	1	1
	F1 score	1	1	1	1	1	1
	G-mean	1	1	1	1	1	1
	AUROC	1	1	1	1	1	1
	AUPRC	1	1	1	1	1	1
	Sensitivity	1	1	1	1	1	1
	Precision	1	1	1	1	1	1
	MCC	1	1	1	1	1	1
	kappa	1	1	1	1	1	1
glass-0-1-2-3_vs_4-5-6	Accuracy	1	4	4	1	4	1
	F1 score	1	6	4	1	4	1
	G-mean	1	6	4	1	4	1
	AUROC	1	4	5	3	5	1
	AUPRC	1	6	3	5	3	1
	Sensitivity	1	6	1	1	1	1
	Precision	1	4	5	1	5	1
	MCC	1	6	4	1	4	1
	kappa	1	6	4	1	4	1
ecoli2	Accuracy	2	1	6	4	2	5
	F1 score	3	1	6	4	2	5
	G-mean	5	1	6	3	2	4
	AUROC	2	1	6	4	3	5
	AUPRC	3	1	6	4	2	5
	Sensitivity	6	1	5	1	1	1
	Precision	1	2	6	4	3	5
	MCC	3	1	6	4	2	5
	kappa	3	1	6	4	2	5

Continuation of Table B.1

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
ecoli	Accuracy	2	6	1	3	5	4
	F1 score	3	6	1	2	5	3
	G-mean	6	2	5	3	1	4
	AUROC	2	4	6	1	3	5
	AUPRC	6	4	2	1	3	5
	Sensitivity	6	1	5	3	1	3
	Precision	2	6	1	3	5	4
	MCC	5	6	1	2	4	3
	kappa	3	6	1	2	5	4
abalone	Accuracy	1	2	6	4	5	3
	F1 score	5	1	6	2	4	3
	G-mean	5	1	6	2	4	3
	AUROC	5	2	6	4	1	3
	AUPRC	5	4	6	1	3	2
	Sensitivity	6	1	5	2	3	3
	Precision	1	2	6	4	5	3
	MCC	5	1	6	2	4	3
	kappa	5	1	6	2	4	3
libras_move	Accuracy	1	6	4	5	3	2
	F1 score	1	6	4	5	3	2
	G-mean	1	6	4	5	3	2
	AUROC	1	6	4	5	3	2
	AUPRC	1	6	4	5	3	2
	Sensitivity	1	6	1	1	1	1
	Precision	1	6	4	5	3	2
	MCC	1	6	4	5	3	2
	kappa	1	6	4	5	3	2
solar_flare_m0	Accuracy	1	5	6	2	4	3
	F1 score	6	4	5	1	2	3
	G-mean	6	4	5	1	2	3
	AUROC	6	5	3	1	2	4
	AUPRC	6	1	3	5	2	4
	Sensitivity	6	1	2	4	3	4
	Precision	1	5	6	2	3	4
	MCC	5	3	6	1	2	4
	kappa	3	5	6	1	2	4
	Accuracy	1	6	4	3	5	2
	F1 score	6	2	5	3	1	4

Continuation of Table B.1

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	G-mean	6	2	4	3	1	5
	AUROC	6	3	5	2	1	4
	AUPRC	6	1	5	2	4	3
	Sensitivity	6	1	3	3	1	5
	Precision	1	5	6	4	2	3
	MCC	6	2	4	3	1	5
	kappa	1	3	6	4	2	5
mammography	Accuracy	1	5	6	3	2	4
	F1 score	1	5	6	3	2	4
	G-mean	4	5	6	2	1	3
	AUROC	5	3	6	2	1	4
	AUPRC	1	6	4	2	5	3
	Sensitivity	6	1	1	5	3	3
	Precision	1	5	6	3	2	4
	MCC	1	5	6	3	2	4
	kappa	1	5	6	3	2	4

End of Table

Table B.2 Ranking scores for the random forest model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
	Accuracy	1	1	6	4	3	4
	F1 score	4	2	6	5	1	3
	G-mean	5	2	6	4	1	3
	AUROC	3	2	6	3	1	5
glass1	AUPRC	3	2	6	5	1	4
	Sensitivity	6	5	1	4	1	1
	Precision	1	2	6	4	3	5
	MCC	2	1	6	5	3	4
	kappa	2	1	6	5	3	4
	Accuracy	1	1	1	1	1	1
	F1 score	1	1	1	1	1	1
	G-mean	1	1	1	1	1	1
	AUROC	1	1	1	1	1	1
iris0	AUPRC	1	1	1	1	1	1
	Sensitivity	1	1	1	1	1	1
	Precision	1	1	1	1	1	1
	MCC	1	1	1	1	1	1
	kappa	1	1	1	1	1	1
	Accuracy	3	3	3	2	6	1
	F1 score	4	4	3	2	6	1
	G-mean	4	4	3	2	6	1
	AUROC	5	2	4	1	3	5
glass-0-1-2-3_vs_4-5-6	AUPRC	5	3	4	1	2	6
	Sensitivity	4	4	2	2	4	1
	Precision	3	3	5	2	6	1
	MCC	4	4	3	2	6	1
	kappa	4	4	3	2	6	1
	Accuracy	1	5	6	2	4	3
	F1 score	1	5	6	2	4	3
	G-mean	4	6	5	1	3	2
	AUROC	2	6	4	3	1	5
ecoli2	AUPRC	2	6	4	3	1	5
	Sensitivity	5	5	1	1	1	1
	Precision	1	5	6	2	4	3
	MCC	1	6	5	2	4	3
	kappa	1	5	6	2	4	3

Continuation of Table B.2

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
ecoli	Accuracy	1	2	6	3	5	4
	F1 score	2	1	6	3	5	4
	G-mean	5	1	6	2	4	3
	AUROC	2	1	6	3	4	5
	AUPRC	1	2	6	3	4	5
	Sensitivity	6	1	5	1	1	1
	Precision	1	2	6	3	5	4
	MCC	2	1	6	3	5	4
	kappa	2	1	6	3	5	4
abalone	Accuracy	1	5	6	4	2	2
	F1 score	5	1	6	4	2	2
	G-mean	5	1	6	4	2	2
	AUROC	5	3	6	4	1	2
	AUPRC	5	4	6	3	1	2
	Sensitivity	6	1	5	2	2	2
	Precision	1	5	6	4	2	2
	MCC	5	1	6	4	2	2
	kappa	5	1	6	4	2	2
libras_move	Accuracy	1	3	6	4	4	1
	F1 score	2	3	6	4	4	1
	G-mean	6	2	5	3	3	1
	AUROC	2	3	4	5	6	1
	AUPRC	2	3	6	4	5	1
	Sensitivity	6	3	1	3	3	1
	Precision	1	3	6	4	4	2
	MCC	2	3	6	4	4	1
	kappa	2	3	6	4	4	1
solar_flare_m0	Accuracy	1	5	6	2	3	3
	F1 score	1	5	6	2	3	3
	G-mean	6	4	5	1	2	2
	AUROC	6	4	3	2	5	1
	AUPRC	1	6	3	4	5	2
	Sensitivity	6	1	2	5	3	3
	Precision	1	5	6	2	3	3
	MCC	1	5	6	2	3	3
	kappa	1	5	6	2	3	3
	Accuracy	1	5	6	3	3	2
	F1 score	1	5	6	2	4	3

Continuation of Table B.2

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	G-mean	6	4	5	1	3	2
	AUROC	2	6	5	1	3	4
	AUPRC	1	3	6	2	4	5
	Sensitivity	6	4	4	1	2	2
	Precision	1	5	6	2	4	3
	MCC	1	5	6	2	4	3
	kappa	1	5	6	2	4	3
mammography	Accuracy	1	5	6	3	2	4
	F1 score	1	5	6	3	2	4
	G-mean	5	4	6	1	3	2
	AUROC	1	5	6	2	4	3
	AUPRC	1	6	5	2	4	3
	Sensitivity	6	1	1	3	5	3
	Precision	1	5	6	3	2	4
	MCC	1	5	6	3	2	4
	kappa	1	5	6	3	2	4

End of Table

Table B.3 Ranking scores for the support vector machine model.

Begin of Table							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	Accuracy	2	5	6	3	1	4
	F1 score	4	5	6	2	1	3
	G-mean	4	5	6	2	1	3
	AUROC	3	6	5	2	1	4
	AUPRC	2	6	4	3	1	5
	Sensitivity	6	4	5	2	1	2
	Precision	2	5	6	3	1	4
	MCC	3	5	6	2	1	4
	kappa	2	5	6	3	1	4
iris0	Accuracy	1	1	1	1	1	1
	F1 score	1	1	1	1	1	1
	G-mean	1	1	1	1	1	1
	AUROC	1	1	1	1	1	1
	AUPRC	1	1	1	1	1	1
	Sensitivity	1	1	1	1	1	1
	Precision	1	1	1	1	1	1
	MCC	1	1	1	1	1	1
	kappa	1	1	1	1	1	1
glass-0-1-2-3_vs_4-5-6	Accuracy	5	5	2	2	2	1
	F1 score	5	5	2	4	2	1
	G-mean	5	5	2	4	2	1
	AUROC	5	6	3	1	4	1
	AUPRC	4	5	3	1	6	1
	Sensitivity	5	5	1	4	1	1
	Precision	5	5	3	2	3	1
	MCC	5	5	2	4	2	1
	kappa	5	5	2	4	2	1
ecoli2	Accuracy	1	3	6	3	2	5
	F1 score	1	3	6	3	2	5
	G-mean	1	3	6	3	2	5
	AUROC	1	3	6	5	1	3
	AUPRC	1	4	6	5	1	3
	Sensitivity	1	1	1	1	1	1
	Precision	1	3	6	3	2	5
	MCC	1	3	6	3	2	5
	kappa	1	3	6	3	2	5

Continuation of Table B.3

Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
ecoli	Accuracy	1	2	6	3	4	5
	F1 score	1	2	6	4	5	3
	G-mean	1	3	6	4	5	2
	AUROC	4	5	6	1	3	1
	AUPRC	5	4	6	1	3	1
	Sensitivity	2	2	2	2	2	1
	Precision	1	2	6	3	5	4
	MCC	1	2	6	4	5	3
	kappa	1	2	6	4	5	3
abalone	Accuracy	1	5	6	2	3	4
	F1 score	6	4	5	2	1	3
	G-mean	6	4	5	3	1	2
	AUROC	5	2	6	1	3	4
	AUPRC	5	1	6	2	3	4
	Sensitivity	6	1	5	4	1	3
	Precision	6	4	5	2	1	3
	MCC	5	4	6	3	1	2
	kappa	5	4	6	2	1	3
libras_move	Accuracy	1	2	6	5	3	3
	F1 score	1	2	6	5	3	3
	G-mean	5	1	6	4	2	2
	AUROC	5	2	6	1	2	4
	AUPRC	4	5	6	1	2	3
	Sensitivity	5	1	5	1	1	1
	Precision	1	2	6	5	3	3
	MCC	1	2	6	5	3	3
	kappa	1	2	6	5	3	3
solar_flare_m0	Accuracy	1	4	6	2	5	3
	F1 score	1	4	6	2	5	3
	G-mean	6	3	5	1	4	2
	AUROC	6	3	5	1	2	4
	AUPRC	1	2	6	3	4	5
	Sensitivity	6	1	1	3	3	3
	Precision	1	4	6	2	5	3
	MCC	2	3	6	1	5	4
	kappa	1	4	6	2	5	3
	Accuracy	1	5	6	2	3	4
	F1 score	6	3	5	2	1	4

Continuation of Table B.3							
Dataset name	Metric	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
yeast_me2	G-mean	6	2	5	3	1	4
	AUROC	6	1	5	3	2	4
	AUPRC	1	2	3	5	4	6
	Sensitivity	6	1	3	3	1	3
	Precision	6	3	5	2	1	4
	MCC	6	3	5	2	1	4
	kappa	6	3	5	2	1	4
mammography	Accuracy	1	5	6	2	3	4
	F1 score	1	5	6	2	3	4
	G-mean	5	4	6	1	2	3
	AUROC	4	5	6	1	2	3
	AUPRC	1	5	6	3	2	4
	Sensitivity	6	4	1	1	3	4
	Precision	1	5	6	2	3	4
	MCC	1	5	6	2	3	4
	kappa	1	5	6	2	3	4
End of Table							

APPENDIX C

REPORT OF OPTIMIZE PARAMETERS

Table C.1 Report of optimal parameters for the decision tree model.

Dataset name	Parameter	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	63	78	28	5	50	24
	min_samples_split	74	40	51	5	2	2
iris0	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	22	54	70	80	99	24
	min_samples_split	68	24	45	55	29	54
glass-0-1-2-3_vs_4-5-6	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	47	52	74	76	59	66
	min_samples_split	8	10	65	37	39	6
ecoli2	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	5	45	3	77	93	3
	min_samples_split	29	31	3	49	3	23
ecoli	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	47	32	8	47	55	22
	min_samples_split	60	49	16	15	17	17
abalone	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	57	68	27	2	4	2
	min_samples_split	3	70	100	86	80	4
libras_move	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	78	13	91	78	85	74
	min_samples_split	4	31	9	22	13	11
solar_flare_m0	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	74	35	4	62	67	30
	min_samples_split	4	2	2	5	2	5
yeast_me2	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	97	71	40	89	50	30
	min_samples_split	2	2	35	13	37	2
mammography	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	max_depth	59	36	48	76	78	59
	min_samples_split	33	43	55	7	33	5

Table C.2 Report of optimal parameters for the random forest model.

Dataset name	Parameter	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	70	14	362	137	36	442
	max_depth	20.33	157.27	102.86	37.83	17.32	12.50
iris0	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	373	106	98	217	449	143
	max_depth	298.55	119.62	7.69	1.78	2.84	6.94
glass-0-1-2-3_vs_4-5-6	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	46	396	85	198	239	197
	max_depth	109.61	49.87	421.21	210.20	116.79	4.60
ecoli2	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	375	3	403	261	81	28
	max_depth	387.66	9.41	2.47	2.39	12.26	4.17
ecoli	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	165	203	428	181	214	118
	max_depth	40.71	356.42	82.41	8.74	299.65	25.85
abalone	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	3	335	183	58	416	309
	max_depth	34.81	4.02	5.18	2.69	2.07	2.98
libras_move	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	30	300	19	21	126	417
	max_depth	10.52	361.57	7.06	23.73	44.96	457.79
solar_flare_m0	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	2	228	234	410	34	42
	max_depth	10.64	70.89	226.90	4.59	129.97	42.93
yeast_me2	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	7	181	31	149	195	105
	max_depth	132.72	45.42	6.22	33.10	72.83	72.14
mammography	criterion	gini index	gini index	gini index	gini index	gini index	gini index
	n_estimators	49	484	320	115	263	186
	max_depth	15.73	19.98	266.44	37.92	21.95	80.77

Table C.3 Report of optimal parameters for the support vector machine model.

Dataset name	Parameter	Undersampling Method					
		None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	C	45.00781	1.83155	14.31699	23.47540	34.93333	39.66526
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.70784	0.04841	0.91205	0.19230	0.36970	0.97996
iris0	C	43.25908	1.38988	11.87009	15.11733	2.26292	49.25986
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.00130	0.01871	0.00056	0.01661	0.20596	0.01910
glass-0-1-2-3_vs_4-5-6	C	6.95641	7.67253	3.78185	1.92010	21.96538	1.08254
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.02474	0.02416	0.07951	0.10213	0.29440	0.08892
ecoli2	C	34.41158	1.61272	4.66512	48.08350	19.85147	25.86596
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.90697	0.37697	0.04508	0.14903	0.88054	0.01718
ecoli	C	35.33428	27.80937	8.61512	19.09766	13.49229	20.74351
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.95417	0.58603	0.08665	0.18432	0.98430	0.73581
abalone	C	62.21637	48.58663	1.43683	52.47040	21.32463	61.82094
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.00002	0.35604	0.48739	0.98067	0.90163	0.70378
libras_move	C	68.41414	7.78533	26.94547	4.07597	6.19252	12.32828
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.03329	0.34089	0.04381	0.53077	0.16355	0.12579
solar_flare_m0	C	44.04344	32.83469	6.73459	1.01141	2.97452	46.47843
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.11902	0.14940	0.06824	0.21630	0.74315	0.55391
yeast_me2	C	38.82924	5.86669	40.32024	3.25104	3.31192	8.15110
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.84669	0.02614	0.01309	0.03772	0.19505	0.67308
mammography	C	39.73587	1.33356	1.05477	56.42831	3.97367	4.61503
	kernel	rbf	rbf	rbf	rbf	rbf	rbf
	gamma	0.15172	0.27988	0.19592	0.31152	0.55724	0.92099

Table C.4 Report of maximize F1 score for the decision tree model.

Dataset name	Undersampling Method					
	None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	0.6065	0.7429	0.8302	0.6983	0.7570	0.8275
iris0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
glass-0-1-2-3_vs_4-5-6	0.9099	0.9149	0.9413	0.9381	0.9746	0.9546
ecoli2	0.7452	0.8090	0.8090	0.8163	0.8760	0.8534
ecoli	0.5176	0.8600	0.6062	0.9024	0.9800	0.9357
abalone	0.2981	0.8670	0.6505	0.8215	0.8262	0.8490
libras_move	0.6533	0.8600	0.8967	0.7367	0.8800	0.8167
solar_flare_m0	0.1464	0.8643	0.7187	0.7103	0.8297	0.7632
yeast_me2	0.3794	0.9131	0.7182	0.8963	0.9235	0.8592
mammography	0.6218	0.8510	0.9850	0.8562	0.8528	0.8593

Table C.5 Report of maximize F1 score for the random forest model.

Dataset name	Undersampling Method					
	None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	0.7817	0.8001	0.8734	0.8256	0.7990	0.8226
iris0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
glass-0-1-2-3_vs_4-5-6	0.9496	0.9270	0.9667	0.9667	0.9889	0.9889
ecoli2	0.7940	0.8871	0.8770	0.8881	0.9238	0.9460
ecoli	0.6300	0.9064	0.6329	0.9371	0.9657	0.9800
abalone	0.2752	0.8818	0.6645	0.8237	0.8348	0.8571
libras_move	0.6000	0.9467	0.9133	0.9133	0.9467	0.9667
solar_flare_m0	0.1586	0.9422	0.7523	0.7247	0.8555	0.8504
yeast_me2	0.2783	0.9131	0.7395	0.8963	0.9635	0.9070
mammography	0.6658	0.9182	0.9850	0.9090	0.9067	0.9004

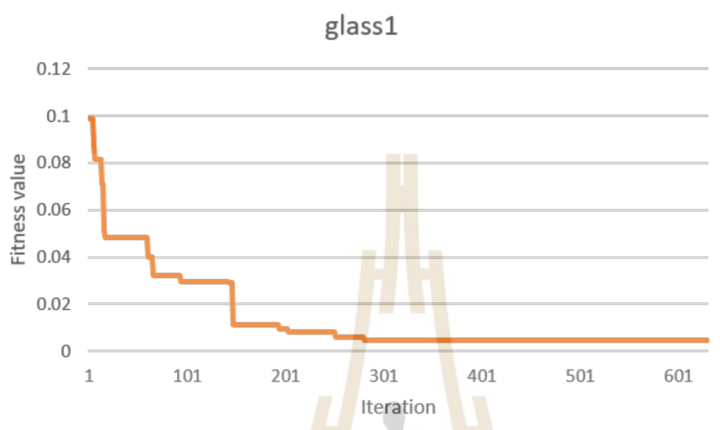
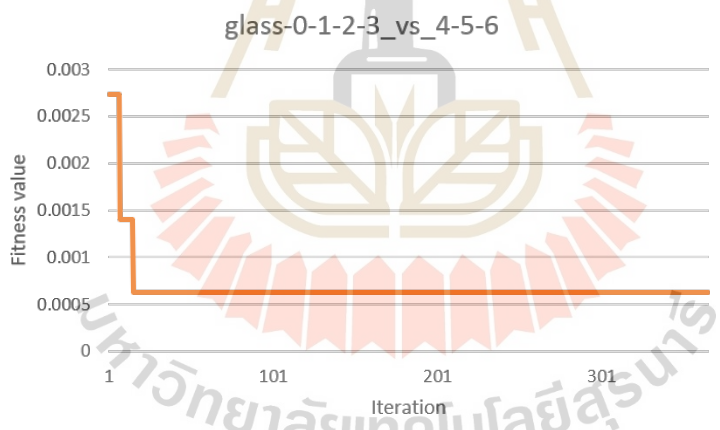
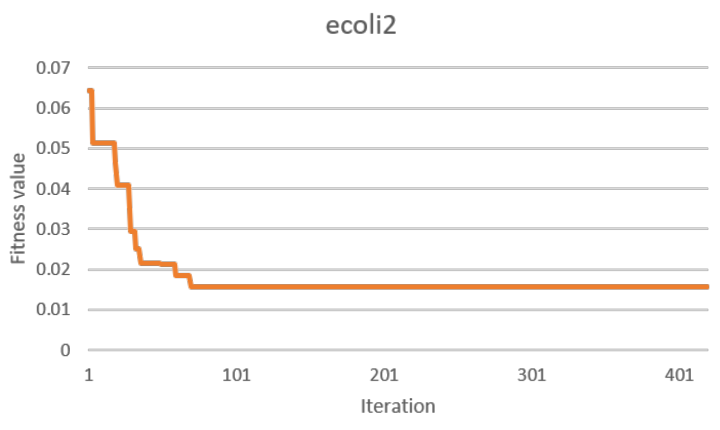
Table C.6 Report of maximize F1 score for the support vector machine model.

Dataset name	Undersampling Method					
	None	Cluster centroid	Near-Miss	RUS	WBWOA 1NN	WBWOA KNN
glass1	0.7287	0.8058	0.8425	0.7959	0.8541	0.8671
iris0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
glass-0-1-2-3_vs_4-5-6	0.9385	0.9524	0.9667	0.9578	0.9889	0.9889
ecoli2	0.8820	0.8798	0.8820	0.9210	0.9317	0.9353
ecoli	0.6524	0.8967	0.7295	0.9371	0.9800	0.9800
abalone	0.0000	0.8753	0.6395	0.8119	0.8474	0.8596
libras_move	0.9000	0.9667	0.9000	0.9467	1.0000	1.0000
solar_flare_m0	0.0786	0.8975	0.7661	0.7801	0.8739	0.8776
yeast_me2	0.0400	0.8237	0.7774	0.8567	0.8427	0.8210
mammography	0.6720	0.8902	0.9850	0.9114	0.9056	0.9011

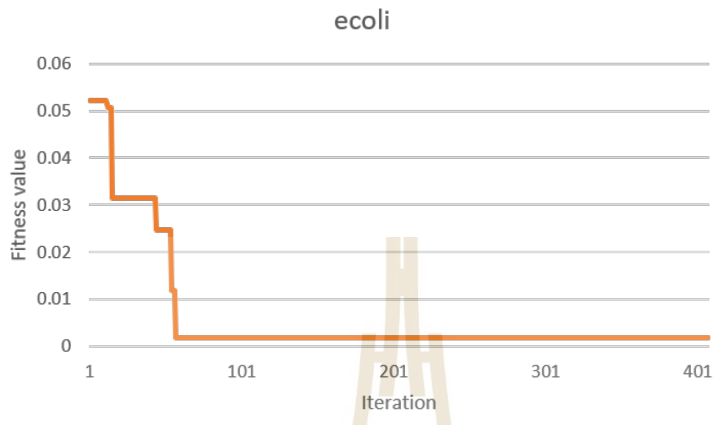
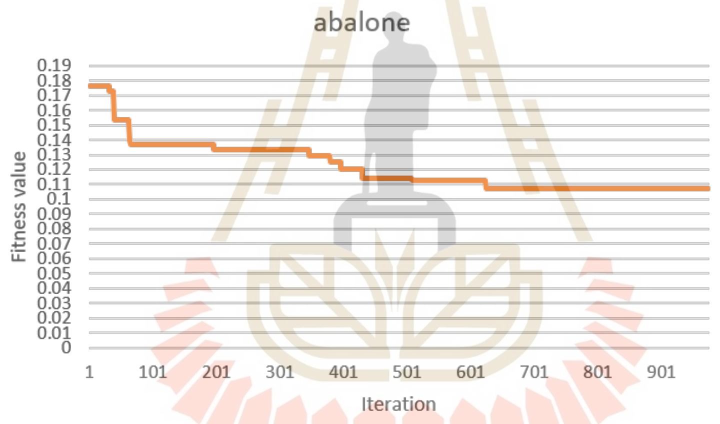
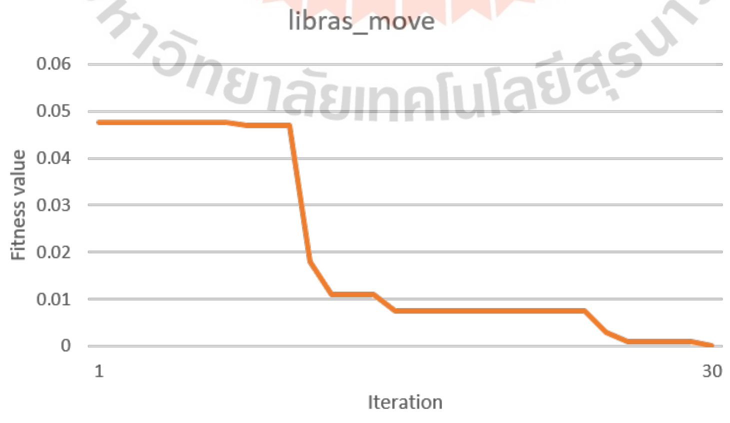
APPENDIX D

REPORT OF FITNESS VALUES

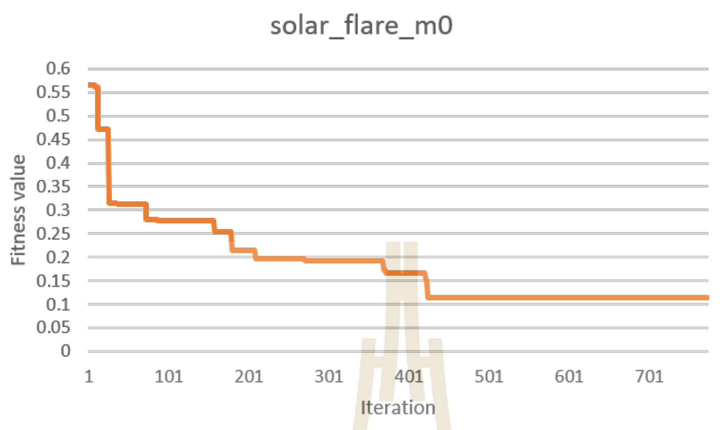
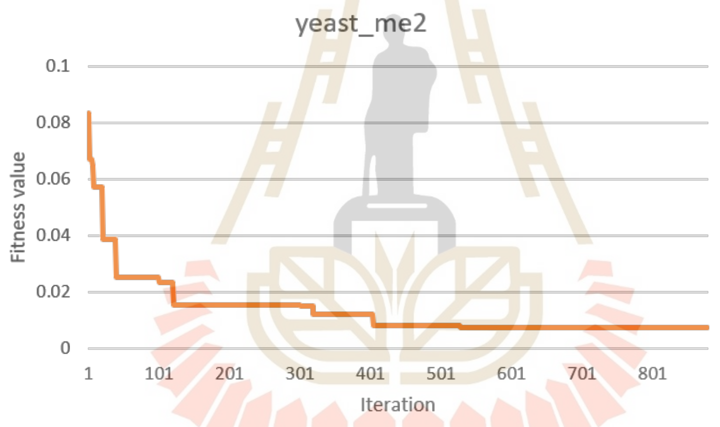
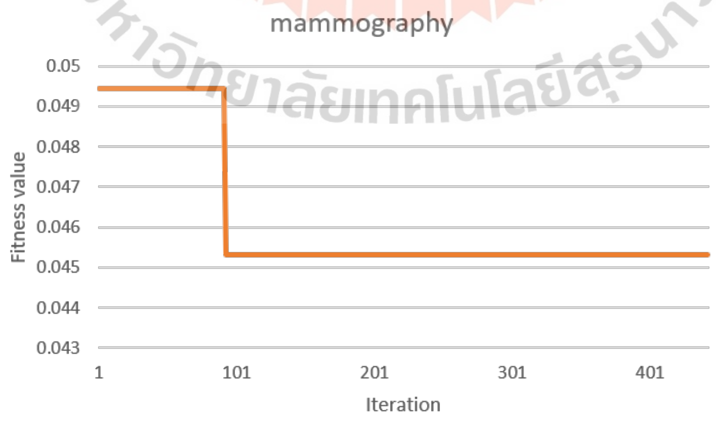
Table D.1 Fitness values of WBWOA 1NN algorithm.

Begin of table			
No.	**Graphs**	**Fitness value**	K
1		0.004756	1
2	iris0 has fitness value equal to 0 at the first iteration.	0.000000	1
3		0.000623	1
4		0.015765	1

Continuation of Table D.1

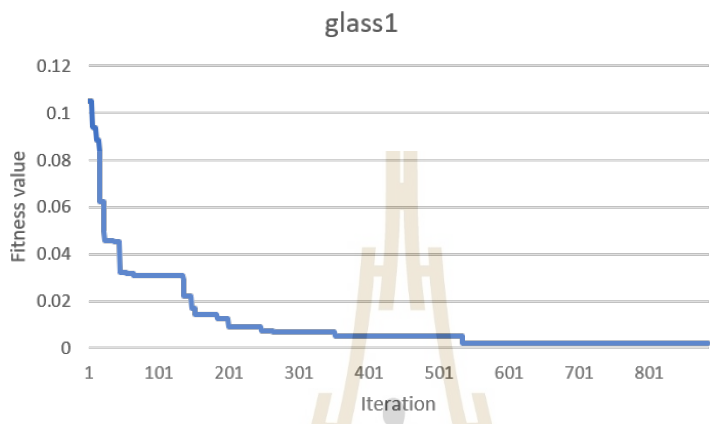
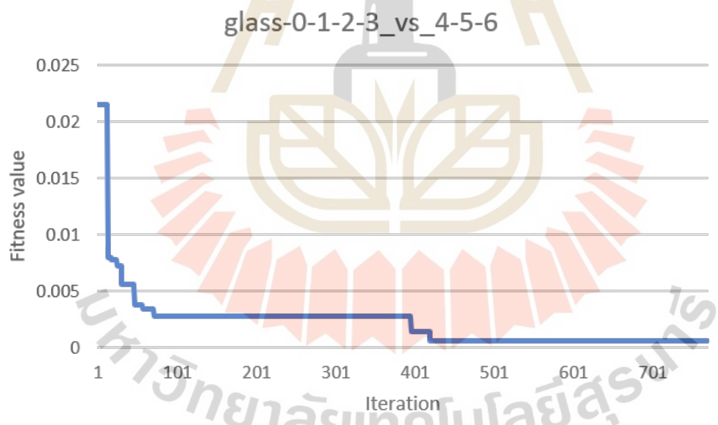
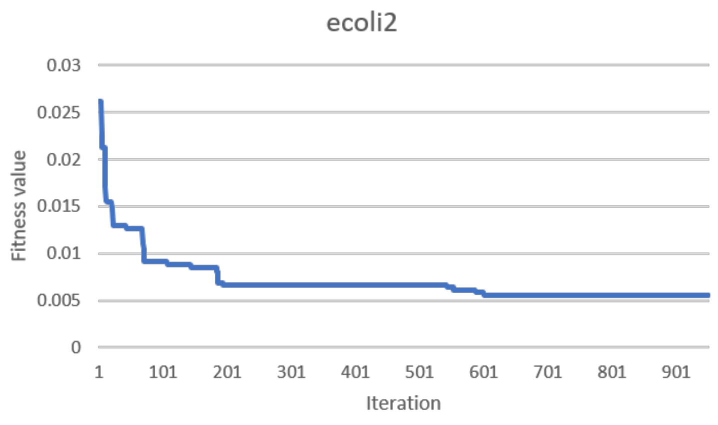
No.	Graphs	Fitness value	K
5		0.001789	1
6		0.106982	1
7		0.000000	1

Continuation of Table D.1

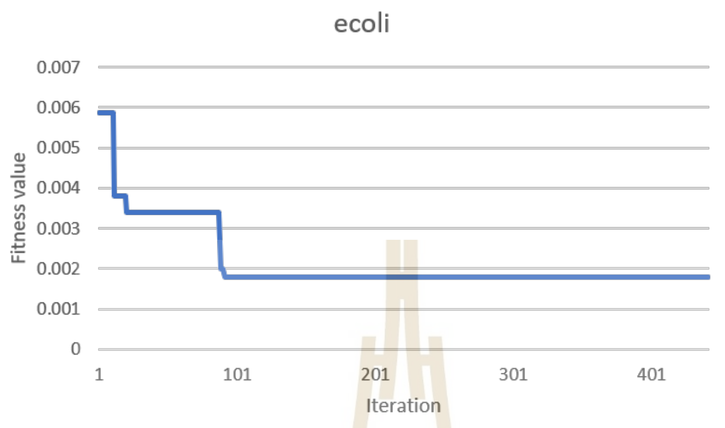
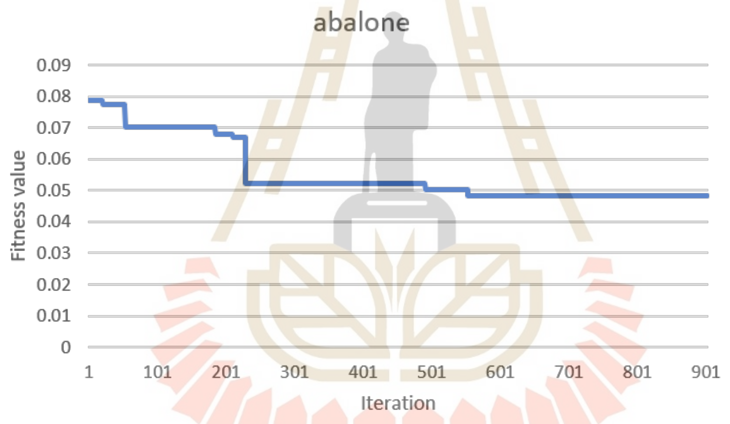
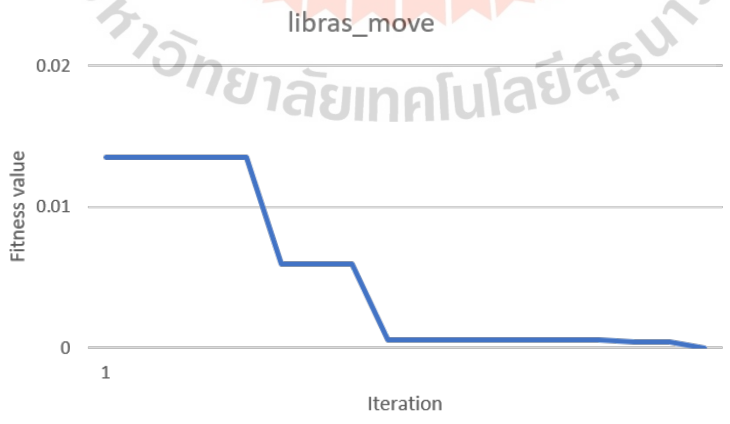
No.	Graphs	Fitness value	K
8		0.113525	1
9		0.007418	1
10		0.045317	1

End of Table

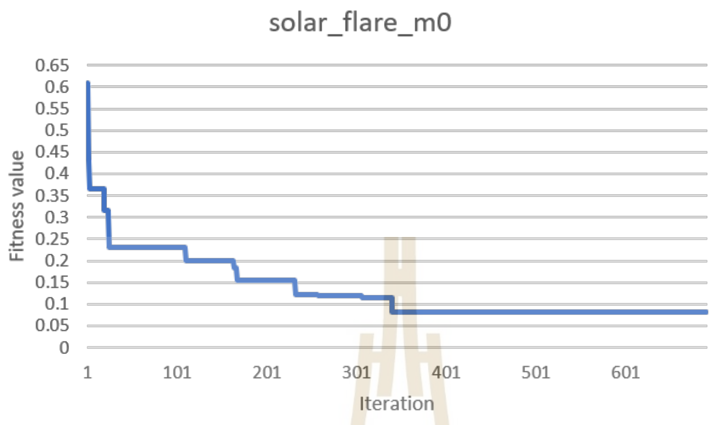
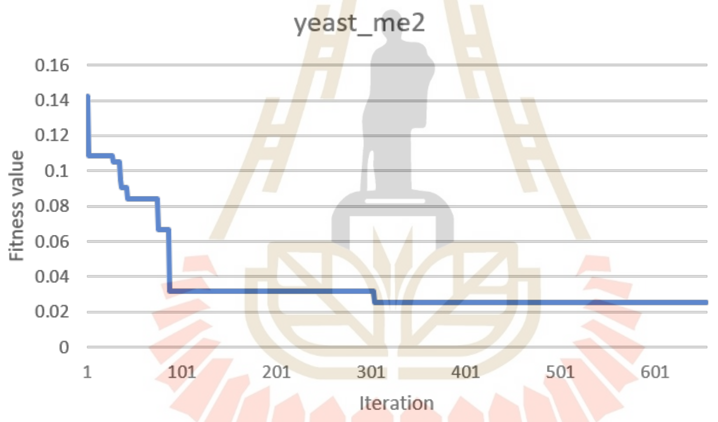
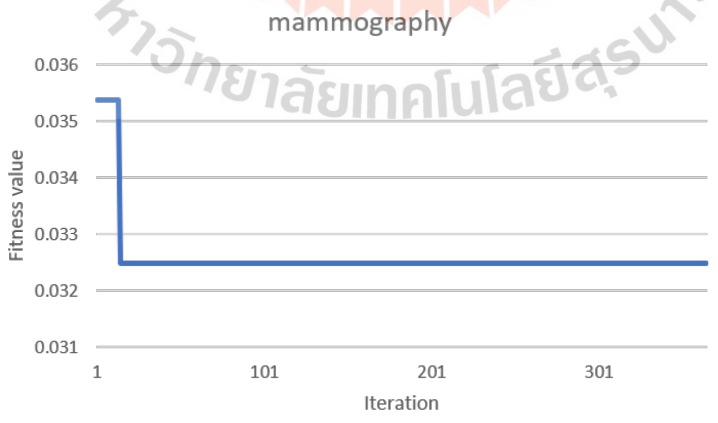
Table D.2 Fitness values of WBWOA KNN algorithm.

Begin of table			
No.	Graphs	Fitness value	K
1		0.002201	1
2	iris0 has fitness value equal to 0 at the first iteration.	0.000000	14
3		0.000623	1
4		0.005518	27

Continuation of Table D.2

No.	Graphs	Fitness value	K
5		0.001789	1
6		0.048363	22
7		0.000000	3

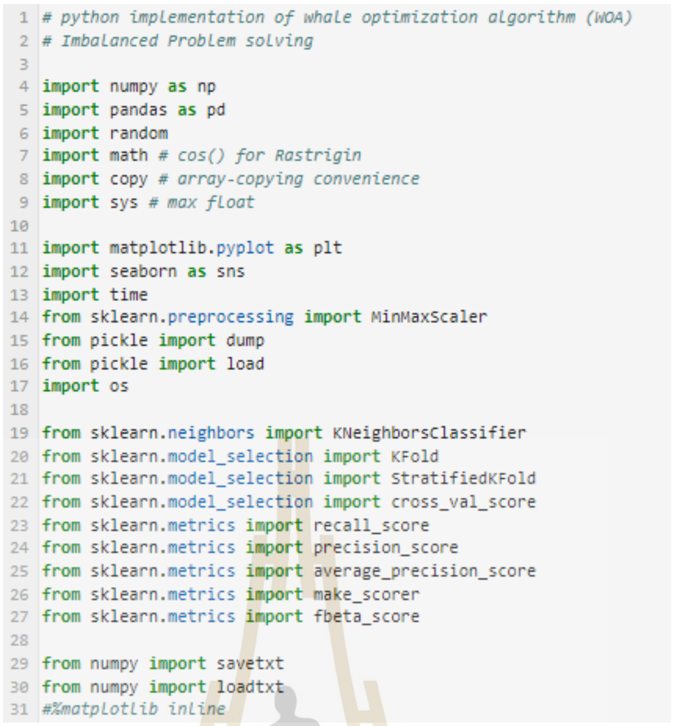
Continuation of Table D.2

No.	Graphs	Fitness value	K
8		0.081860	1
9		0.025287	3
10		0.032486	5

End of Table

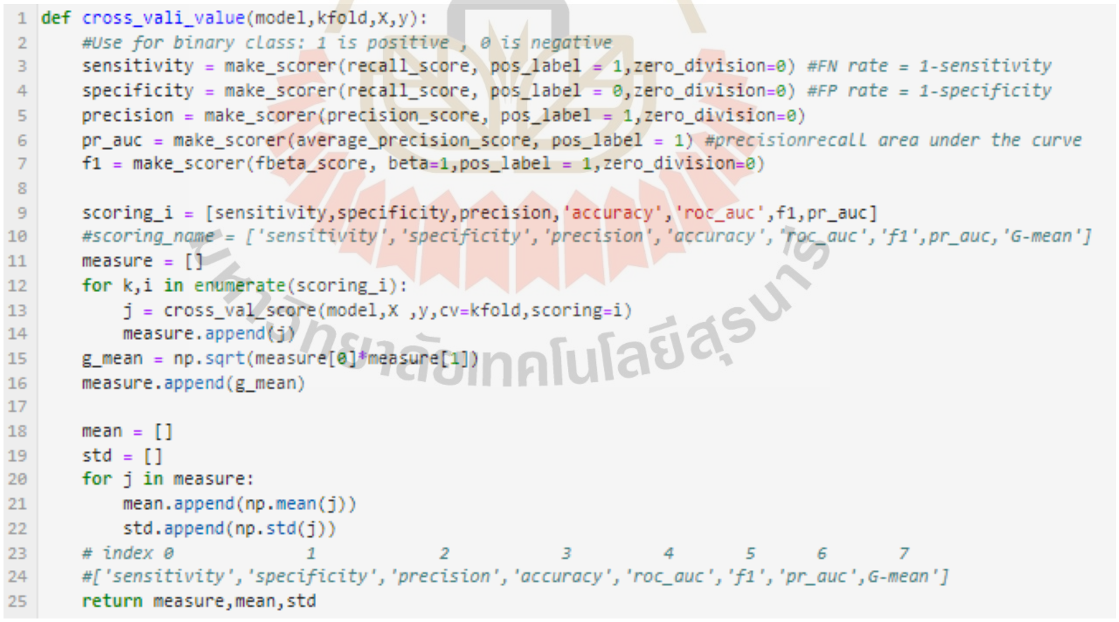
APPENDIX E

CODE OF WBWOA 1NN ALGORITHM



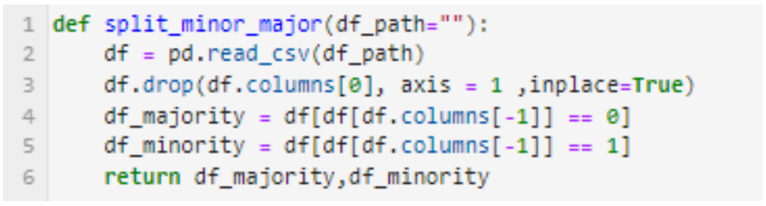
```python
# python implementation of whale optimization algorithm (WOA)
# Imbalanced Problem solving

import numpy as np
import pandas as pd
import random
import math # cos() for Rastrigin
import copy # array-copying convenience
import sys # max float

import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.preprocessing import MinMaxScaler
from pickle import dump
from pickle import load
import os

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import make_scorer
from sklearn.metrics import fbeta_score

from numpy import savetxt
from numpy import loadtxt
#%matplotlib inline
```

Figure E.1 Code for importing library for the prepossessing work.



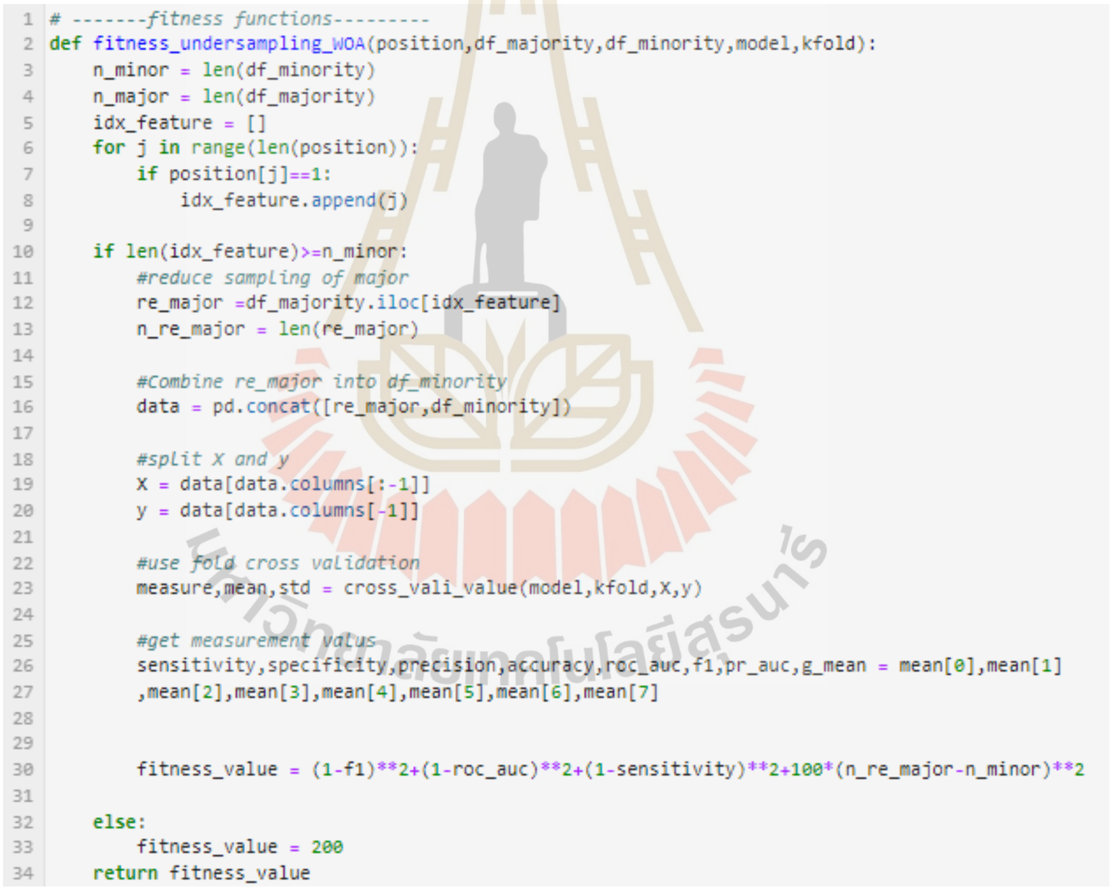
```python
def cross_vali_value(model,kfold,X,y):
    #Use for binary class: 1 is positive , 0 is negative
    sensitivity = make_scorer(recall_score, pos_label = 1,zero_division=0) #FN rate = 1-sensitivity
    specificity = make_scorer(recall_score, pos_label = 0,zero_division=0) #FP rate = 1-specificity
    precision = make_scorer(precision_score, pos_label = 1,zero_division=0)
    pr_auc = make_scorer(average_precision_score, pos_label = 1) #precisionrecall area under the curve
    f1 = make_scorer(fbeta_score, beta=1,pos_label = 1,zero_division=0)

    scoring_i = [sensitivity,specificity,precision,'accuracy','roc_auc',f1,pr_auc]
    #scoring_name = ['sensitivity','specificity','precision','accuracy','roc_auc','f1',pr_auc,'G-mean']
    measure = []
    for k,i in enumerate(scoring_i):
        j = cross_val_score(model,X ,y,cv=kfold,scoring=i)
        measure.append(j)
    g_mean = np.sqrt(measure[0]*measure[1])
    measure.append(g_mean)

    mean = []
    std = []
    for j in measure:
        mean.append(np.mean(j))
        std.append(np.std(j))
    # index 0          1             2            3           4          5      6         7
    #['sensitivity','specificity','precision','accuracy','roc_auc','f1','pr_auc',G-mean']
    return measure,mean,std
```

Figure E.2 Code for creating function of performance metrics.



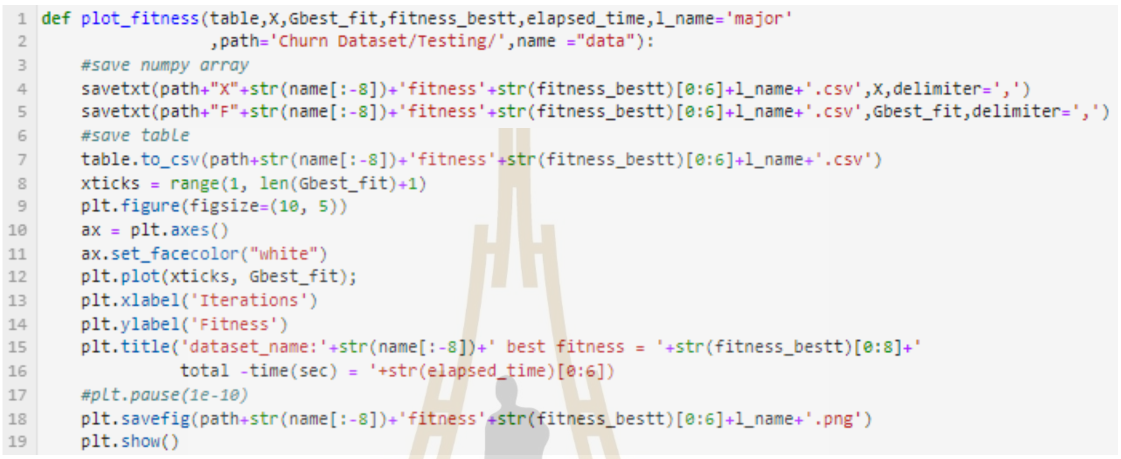
```python
def split_minor_major(df_path=""):
    df = pd.read_csv(df_path)
    df.drop(df.columns[0], axis = 1 ,inplace=True)
    df_majority = df[df[df.columns[-1]] == 0]
    df_minority = df[df[df.columns[-1]] == 1]
    return df_majority,df_minority
```

Figure E.3 Code for creating function for splitting majority and minority tables.



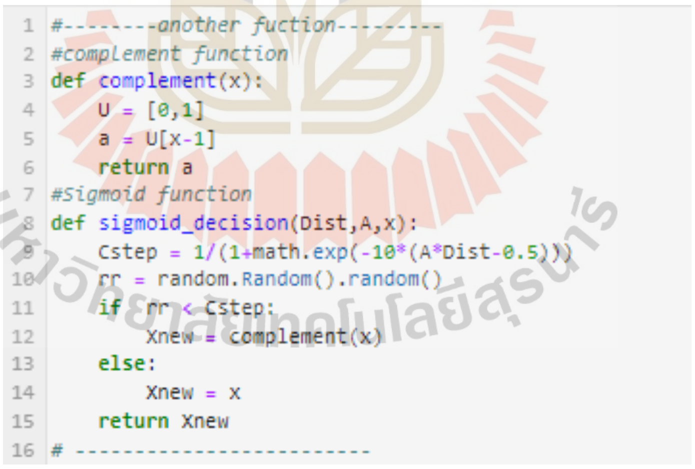
```python
# -------fitness functions---------
def fitness_undersampling_WOA(position,df_majority,df_minority,model,kfold):
    n_minor = len(df_minority)
    n_major = len(df_majority)
    idx_feature = []
    for j in range(len(position)):
        if position[j]==1:
            idx_feature.append(j)

    if len(idx_feature)>=n_minor:
        #reduce sampling of major
        re_major =df_majority.iloc[idx_feature]
        n_re_major = len(re_major)

        #Combine re_major into df_minority
        data = pd.concat([re_major,df_minority])

        #split X and y
        X = data[data.columns[:-1]]
        y = data[data.columns[-1]]

        #use fold cross validation
        measure,mean,std = cross_vali_value(model,kfold,X,y)

        #get measurement valus
        sensitivity,specificity,precision,accuracy,roc_auc,f1,pr_auc,g_mean = mean[0],mean[1]
        ,mean[2],mean[3],mean[4],mean[5],mean[6],mean[7]


        fitness_value = (1-f1)**2+(1-roc_auc)**2+(1-sensitivity)**2+100*(n_re_major-n_minor)**2

    else:
        fitness_value = 200
    return fitness_value
```

Figure E.4 Code for creating function for fitness function.



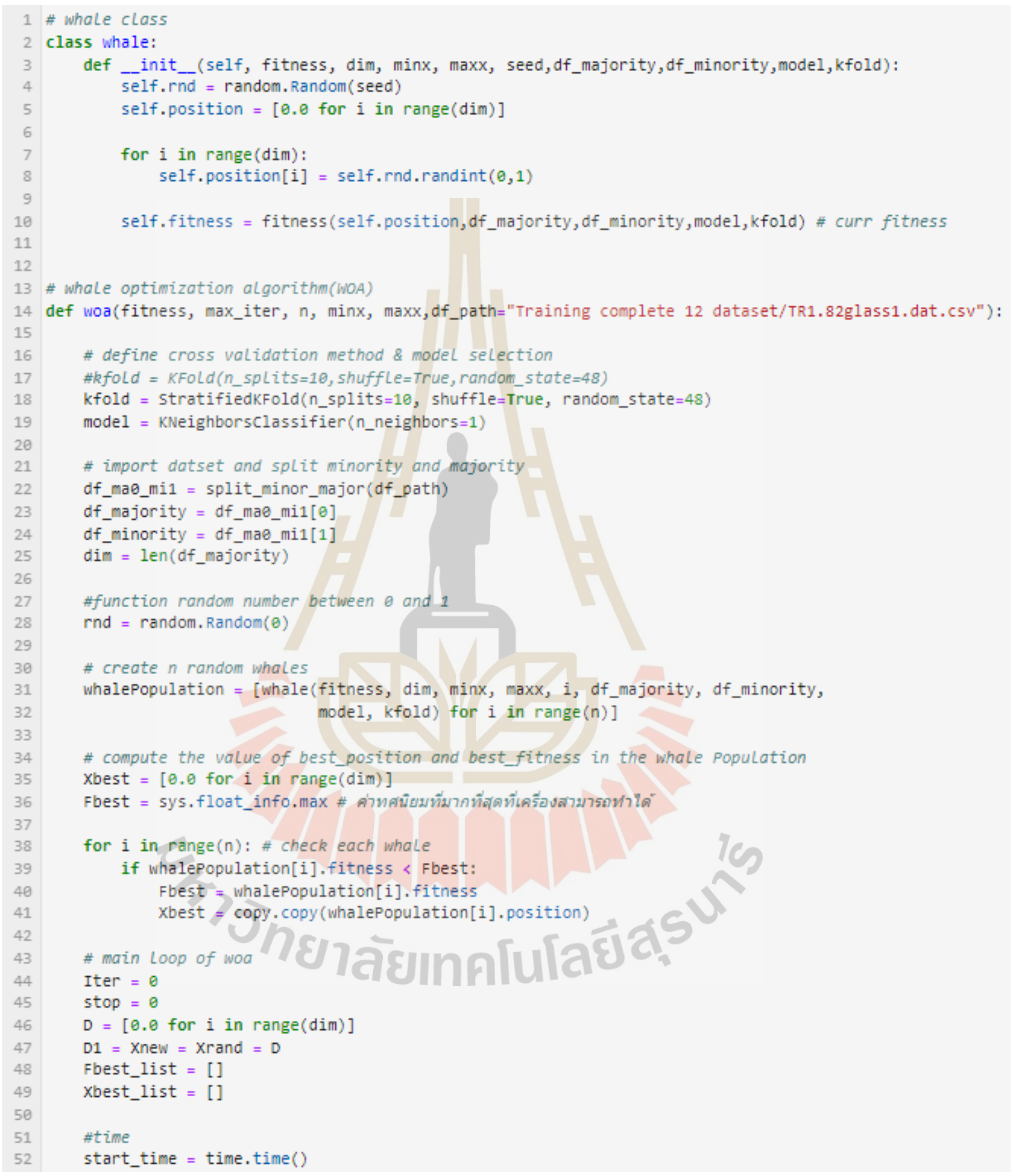
```python
def plot_fitness(table,X,Gbest_fit,fitness_bestt,elapsed_time,l_name='major'
                 ,path='Churn Dataset/Testing/',name ="data"):
    #save numpy array
    savetxt(path+"X"+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv',X,delimiter=',')
    savetxt(path+"F"+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv',Gbest_fit,delimiter=',')
    #save table
    table.to_csv(path+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv')
    xticks = range(1, len(Gbest_fit)+1)
    plt.figure(figsize=(10, 5))
    ax = plt.axes()
    ax.set_facecolor("white")
    plt.plot(xticks, Gbest_fit);
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title('dataset_name:'+str(name[:-8])+' best fitness = '+str(fitness_bestt)[0:8]+'
              total -time(sec) = '+str(elapsed_time)[0:6])
    #plt.pause(1e-10)
    plt.savefig(path+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.png')
    plt.show()
```

Figure E.5 Code for creating function for graph fitness.



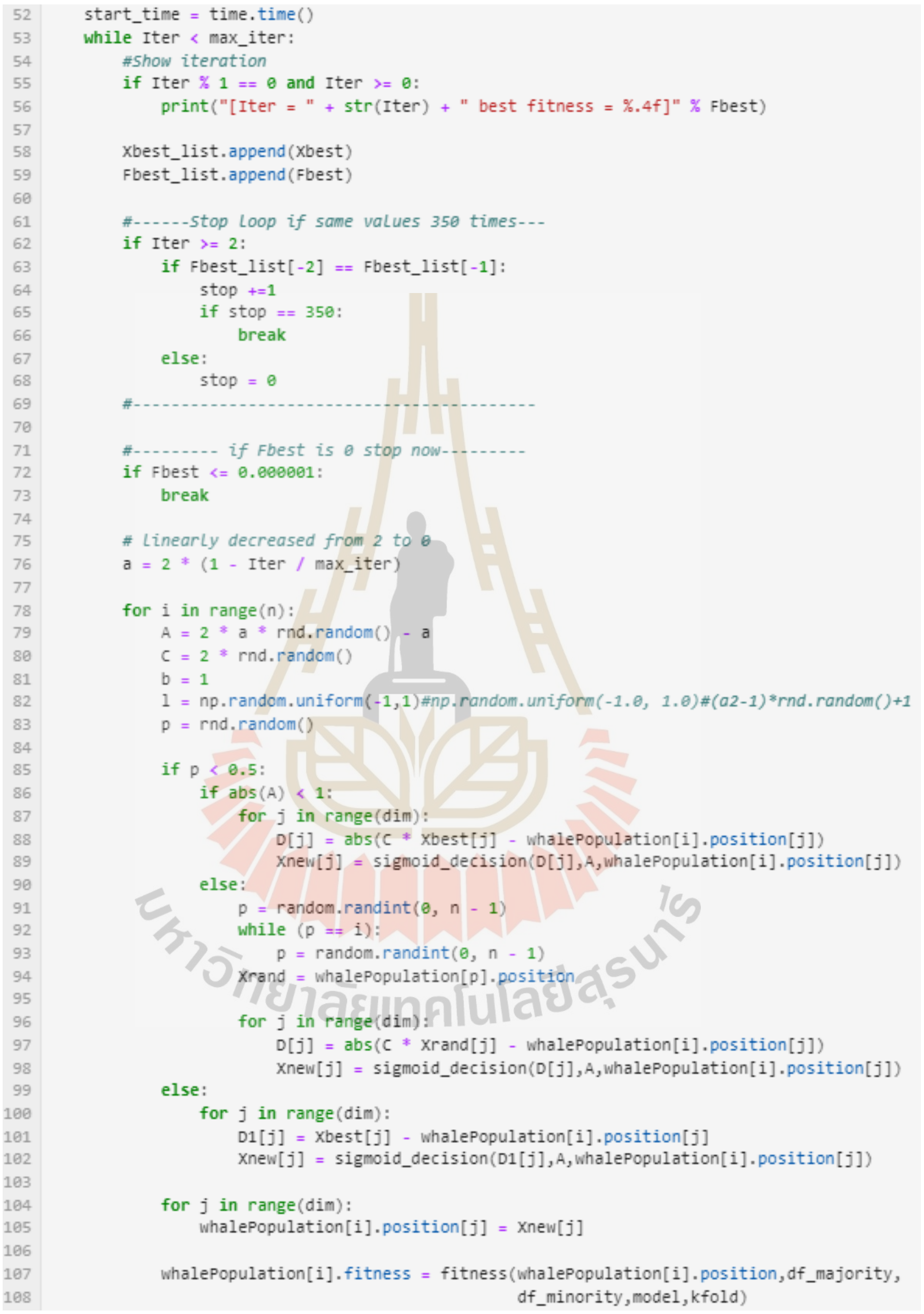
```python
#--------another fuction---------
#complement function
def complement(x):
    U = [0,1]
    a = U[x-1]
    return a
#Sigmoid function
def sigmoid_decision(Dist,A,x):
    Cstep = 1/(1+math.exp(-10*(A*Dist-0.5)))
    rr = random.Random().random()
    if rr < Cstep:
        Xnew = complement(x)
    else:
        Xnew = x
    return Xnew
# -------------------------
```

Figure E.6 Code for complement function.



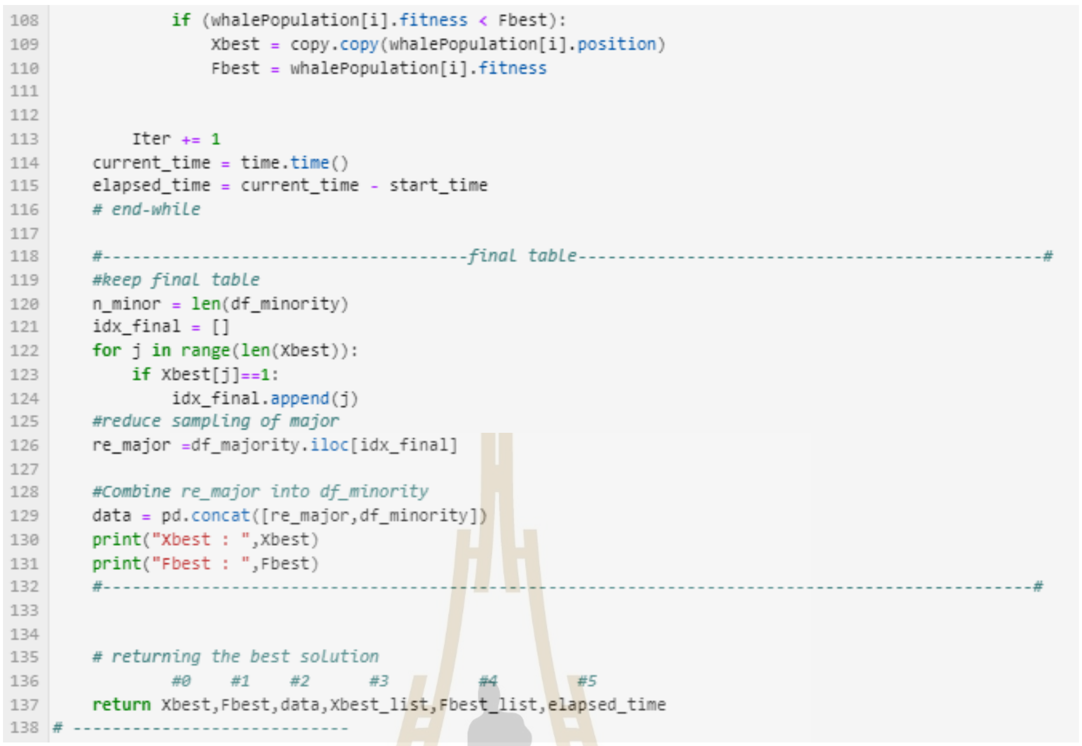
```python
# whale class
class whale:
    def __init__(self, fitness, dim, minx, maxx, seed,df_majority,df_minority,model,kfold):
        self.rnd = random.Random(seed)
        self.position = [0.0 for i in range(dim)]

        for i in range(dim):
            self.position[i] = self.rnd.randint(0,1)

        self.fitness = fitness(self.position,df_majority,df_minority,model,kfold) # curr fitness


# whale optimization algorithm(WOA)
def woa(fitness, max_iter, n, minx, maxx,df_path="Training complete 12 dataset/TR1.82glass1.dat.csv"):

    # define cross validation method & model selection
    #kfold = KFold(n_splits=10,shuffle=True,random_state=48)
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=48)
    model = KNeighborsClassifier(n_neighbors=1)

    # import datset and split minority and majority
    df_ma0_mi1 = split_minor_major(df_path)
    df_majority = df_ma0_mi1[0]
    df_minority = df_ma0_mi1[1]
    dim = len(df_majority)

    #function random number between 0 and 1
    rnd = random.Random(0)

    # create n random whales
    whalePopulation = [whale(fitness, dim, minx, maxx, i, df_majority, df_minority,
                             model, kfold) for i in range(n)]

    # compute the value of best_position and best_fitness in the whale Population
    Xbest = [0.0 for i in range(dim)]
    Fbest = sys.float_info.max # ค่าทศนิยมที่มากที่สุดที่เครื่องสามารถทำได้

    for i in range(n): # check each whale
        if whalePopulation[i].fitness < Fbest:
            Fbest = whalePopulation[i].fitness
            Xbest = copy.copy(whalePopulation[i].position)

    # main loop of woa
    Iter = 0
    stop = 0
    D = [0.0 for i in range(dim)]
    D1 = Xnew = Xrand = D
    Fbest_list = []
    Xbest_list = []

    #time
    start_time = time.time()
```

Figure E.7 Code for creating function for WBWOA 1NN.



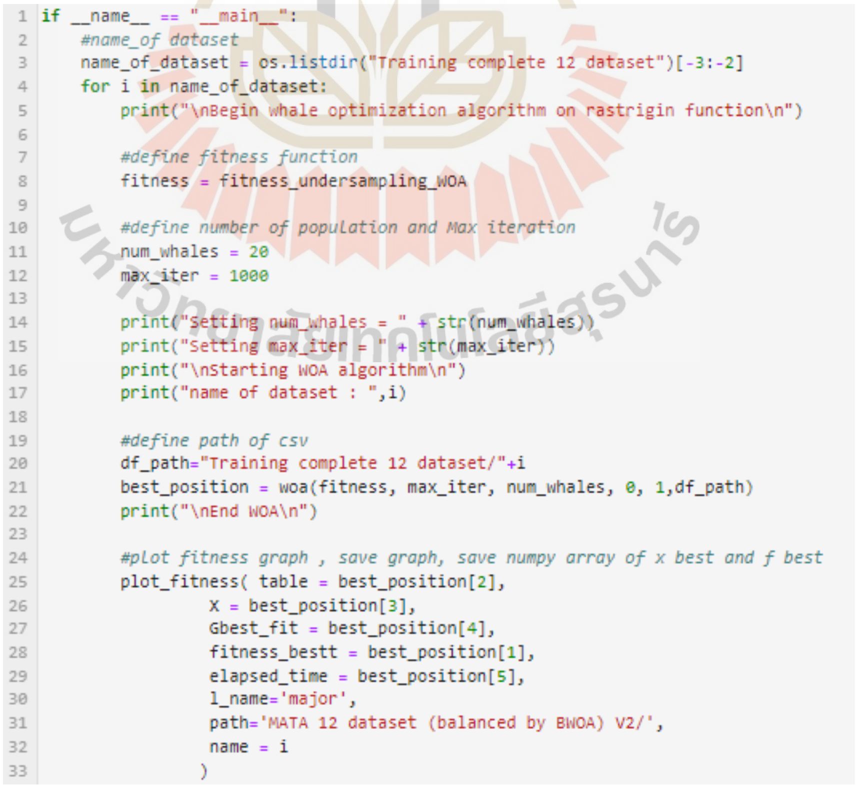
```
    start_time = time.time()
    while Iter < max_iter:
        #Show iteration
        if Iter % 1 == 0 and Iter >= 0:
            print("[Iter = " + str(Iter) + " best fitness = %.4f]" % Fbest)

        Xbest_list.append(Xbest)
        Fbest_list.append(Fbest)

        #------Stop loop if same values 350 times---
        if Iter >= 2:
            if Fbest_list[-2] == Fbest_list[-1]:
                stop +=1
                if stop == 350:
                    break
            else:
                stop = 0
        #-------------------------------------------

        #--------- if Fbest is 0 stop now---------
        if Fbest <= 0.000001:
            break

        # linearly decreased from 2 to 0
        a = 2 * (1 - Iter / max_iter)

        for i in range(n):
            A = 2 * a * rnd.random() - a
            C = 2 * rnd.random()
            b = 1
            l = np.random.uniform(-1,1)#np.random.uniform(-1.0, 1.0)#(a2-1)*rnd.random()+1
            p = rnd.random()

            if p < 0.5:
                if abs(A) < 1:
                    for j in range(dim):
                        D[j] = abs(C * Xbest[j] - whalePopulation[i].position[j])
                        Xnew[j] = sigmoid_decision(D[j],A,whalePopulation[i].position[j])
                else:
                    p = random.randint(0, n - 1)
                    while (p == i):
                        p = random.randint(0, n - 1)
                    Xrand = whalePopulation[p].position

                    for j in range(dim):
                        D[j] = abs(C * Xrand[j] - whalePopulation[i].position[j])
                        Xnew[j] = sigmoid_decision(D[j],A,whalePopulation[i].position[j])
            else:
                for j in range(dim):
                    D1[j] = Xbest[j] - whalePopulation[i].position[j]
                    Xnew[j] = sigmoid_decision(D1[j],A,whalePopulation[i].position[j])

            for j in range(dim):
                whalePopulation[i].position[j] = Xnew[j]

            whalePopulation[i].fitness = fitness(whalePopulation[i].position,df_majority,
                                                 df_minority,model,kfold)
```

Figure E.8 Code for creating function for WBWOA 1NN (Continued1).



```python
            if (whalePopulation[i].fitness < Fbest):
                Xbest = copy.copy(whalePopulation[i].position)
                Fbest = whalePopulation[i].fitness


        Iter += 1
    current_time = time.time()
    elapsed_time = current_time - start_time
    # end-while

    #------------------------------------final table----------------------------------------------#
    #keep final table
    n_minor = len(df_minority)
    idx_final = []
    for j in range(len(Xbest)):
        if Xbest[j]==1:
            idx_final.append(j)
    #reduce sampling of major
    re_major =df_majority.iloc[idx_final]

    #Combine re_major into df_minority
    data = pd.concat([re_major,df_minority])
    print("Xbest : ",Xbest)
    print("Fbest : ",Fbest)
    #-----------------------------------------------------------------------------------------------#


    # returning the best solution
            #0    #1    #2     #3        #4         #5
    return Xbest,Fbest,data,Xbest_list,Fbest_list,elapsed_time
# ----------------------------
```

Figure E.9 Code for creating function for WBWOA 1NN (Continued2).

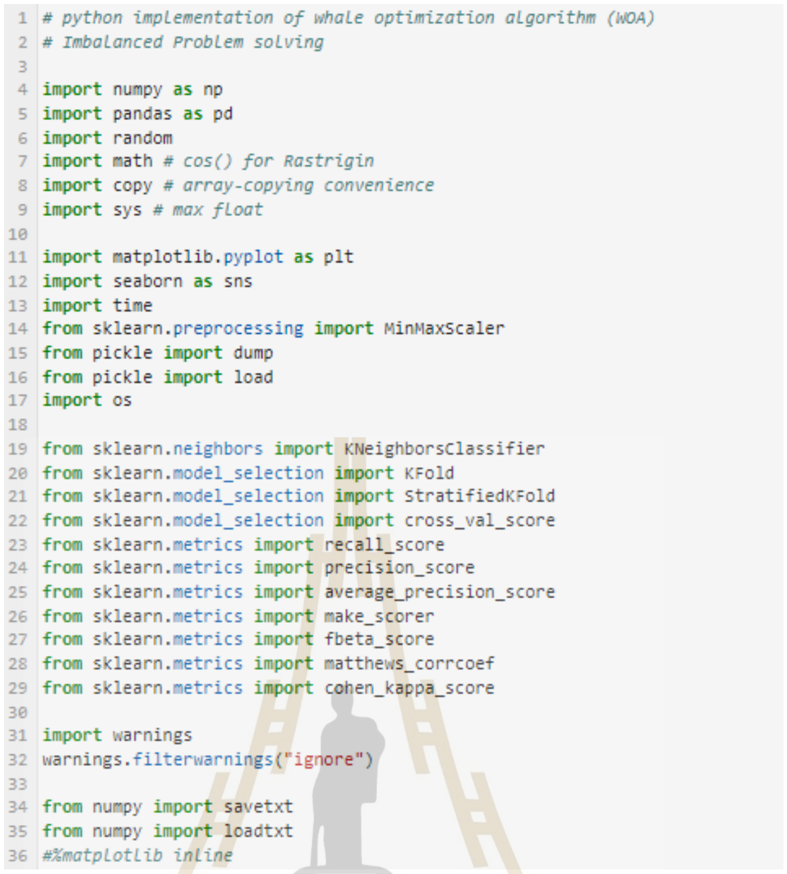


```python
if __name__ == "__main__":
    #name_of_dataset
    name_of_dataset = os.listdir("Training complete 12 dataset")[-3:-2]
    for i in name_of_dataset:
        print("\nBegin whale optimization algorithm on rastrigin function\n")

        #define fitness function
        fitness = fitness_undersampling_WOA

        #define number of population and Max iteration
        num_whales = 20
        max_iter = 1000

        print("Setting num_whales = " + str(num_whales))
        print("Setting max_iter = " + str(max_iter))
        print("\nStarting WOA algorithm\n")
        print("name of dataset : ",i)

        #define path of csv
        df_path="Training complete 12 dataset/"+i
        best_position = woa(fitness, max_iter, num_whales, 0, 1,df_path)
        print("\nEnd WOA\n")

        #plot fitness graph , save graph, save numpy array of x best and f best
        plot_fitness( table = best_position[2],
                      X = best_position[3],
                      Gbest_fit = best_position[4],
                      fitness_bestt = best_position[1],
                      elapsed_time = best_position[5],
                      l_name='major',
                      path='MATA 12 dataset (balanced by BWOA) V2/',
                      name = i
                     )
```

Figure E.10 Code for run the WBWOA 1NN algorithm.

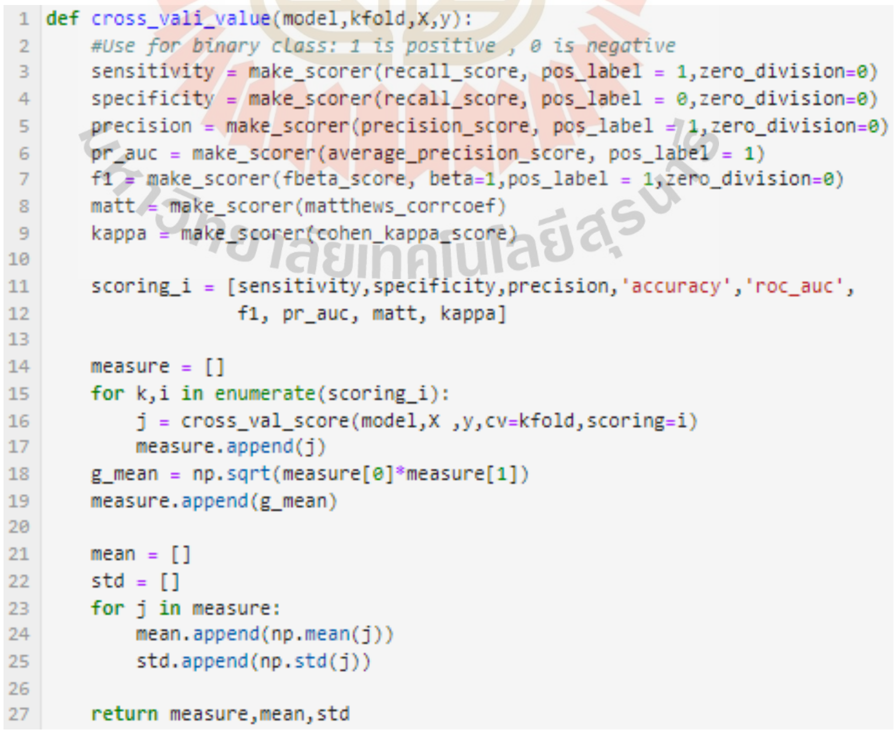
APPENDIX F

CODE OF WBWOA KNN ALGORITHM



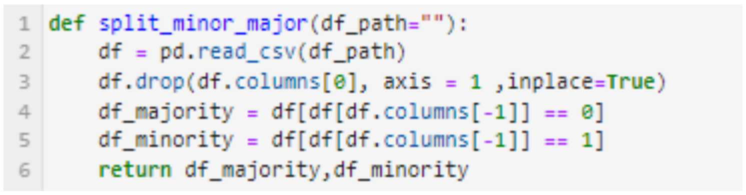
```python
# python implementation of whale optimization algorithm (WOA)
# Imbalanced Problem solving

import numpy as np
import pandas as pd
import random
import math # cos() for Rastrigin
import copy # array-copying convenience
import sys # max float

import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.preprocessing import MinMaxScaler
from pickle import dump
from pickle import load
import os

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import make_scorer
from sklearn.metrics import fbeta_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import cohen_kappa_score

import warnings
warnings.filterwarnings("ignore")

from numpy import savetxt
from numpy import loadtxt
#%matplotlib inline
```

Figure F.1 Code for importing library for the prepossessing work.



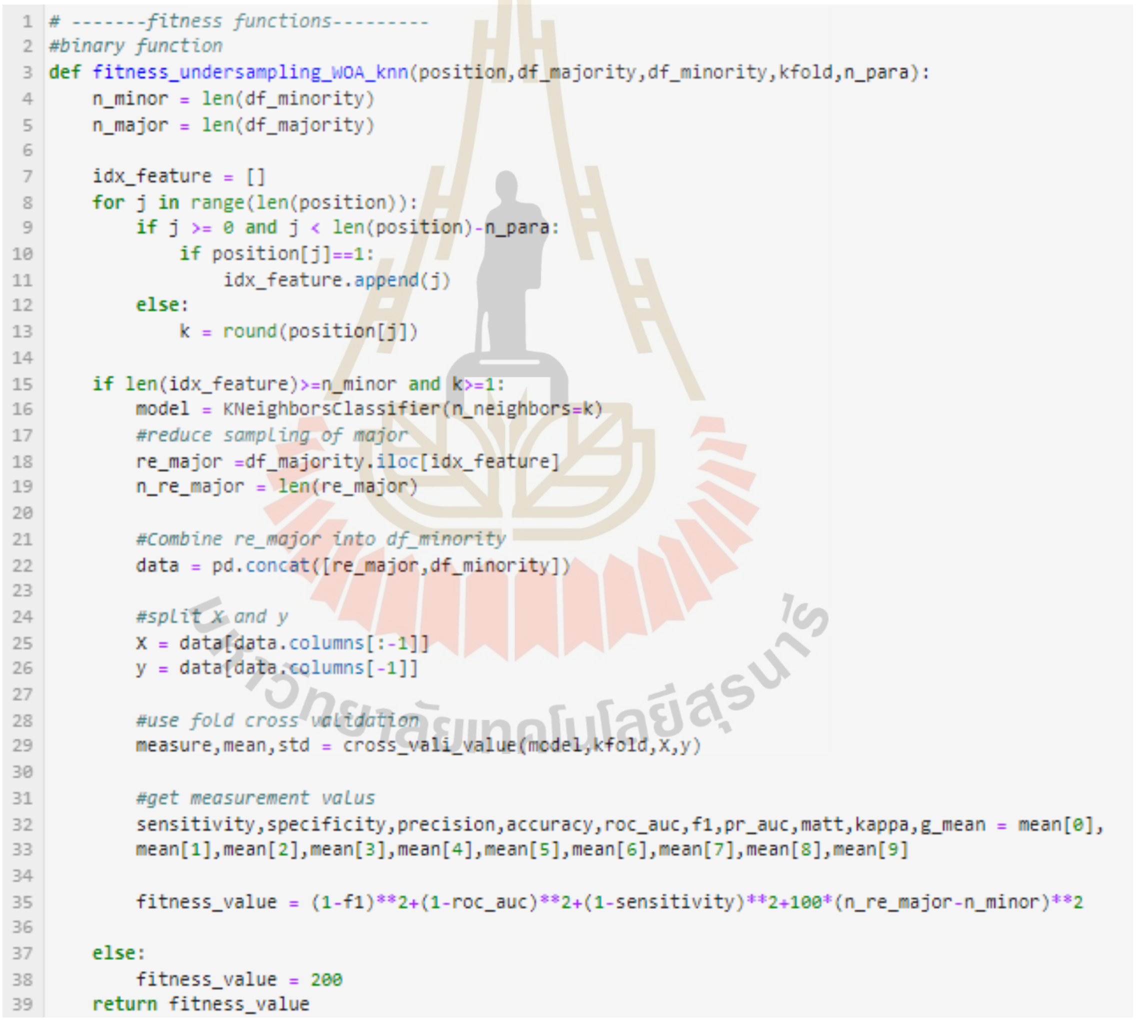
```python
def cross_vali_value(model,kfold,X,y):
    #Use for binary class: 1 is positive , 0 is negative
    sensitivity = make_scorer(recall_score, pos_label = 1,zero_division=0)
    specificity = make_scorer(recall_score, pos_label = 0,zero_division=0)
    precision = make_scorer(precision_score, pos_label = 1,zero_division=0)
    pr_auc = make_scorer(average_precision_score, pos_label = 1)
    f1 = make_scorer(fbeta_score, beta=1,pos_label = 1,zero_division=0)
    matt = make_scorer(matthews_corrcoef)
    kappa = make_scorer(cohen_kappa_score)

    scoring_i = [sensitivity,specificity,precision,'accuracy','roc_auc',
                 f1, pr_auc, matt, kappa]

    measure = []
    for k,i in enumerate(scoring_i):
        j = cross_val_score(model,X ,y,cv=kfold,scoring=i)
        measure.append(j)
    g_mean = np.sqrt(measure[0]*measure[1])
    measure.append(g_mean)

    mean = []
    std = []
    for j in measure:
        mean.append(np.mean(j))
        std.append(np.std(j))

    return measure,mean,std
```

Figure F.2 Code for creating function of performance metrics.



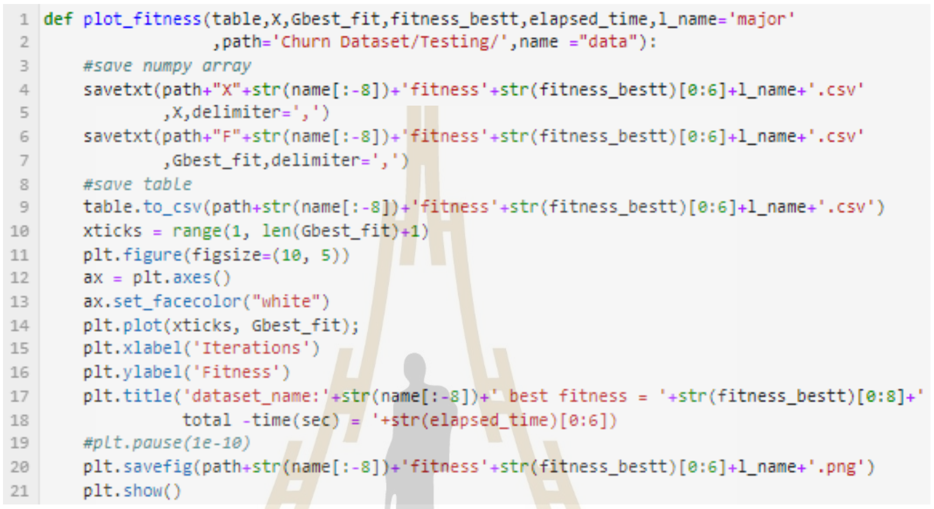
```python
def split_minor_major(df_path=""):
    df = pd.read_csv(df_path)
    df.drop(df.columns[0], axis = 1 ,inplace=True)
    df_majority = df[df[df.columns[-1]] == 0]
    df_minority = df[df[df.columns[-1]] == 1]
    return df_majority,df_minority
```

Figure F.3 Code for creating function for splitting majority and minority tables.



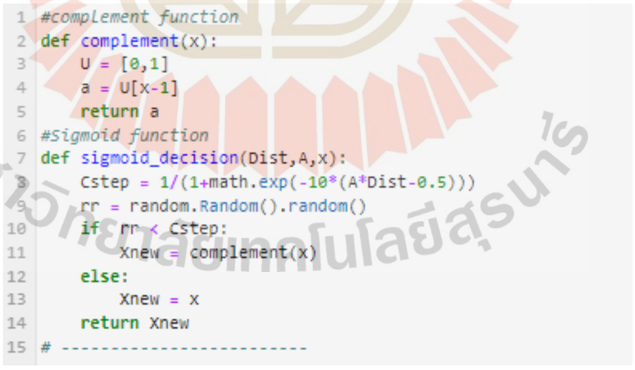
```python
# -------fitness functions---------
#binary function
def fitness_undersampling_WOA_knn(position,df_majority,df_minority,kfold,n_para):
    n_minor = len(df_minority)
    n_major = len(df_majority)

    idx_feature = []
    for j in range(len(position)):
        if j >= 0 and j < len(position)-n_para:
            if position[j]==1:
                idx_feature.append(j)
        else:
            k = round(position[j])

    if len(idx_feature)>=n_minor and k>=1:
        model = KNeighborsClassifier(n_neighbors=k)
        #reduce sampling of major
        re_major =df_majority.iloc[idx_feature]
        n_re_major = len(re_major)

        #Combine re_major into df_minority
        data = pd.concat([re_major,df_minority])

        #split X and y
        X = data[data.columns[:-1]]
        y = data[data.columns[-1]]

        #use fold cross validation
        measure,mean,std = cross_vali_value(model,kfold,X,y)

        #get measurement valus
        sensitivity,specificity,precision,accuracy,roc_auc,f1,pr_auc,matt,kappa,g_mean = mean[0],
        mean[1],mean[2],mean[3],mean[4],mean[5],mean[6],mean[7],mean[8],mean[9]

        fitness_value = (1-f1)**2+(1-roc_auc)**2+(1-sensitivity)**2+100*(n_re_major-n_minor)**2

    else:
        fitness_value = 200
    return fitness_value
```

Figure F.4 Code for creating function for fitness function.



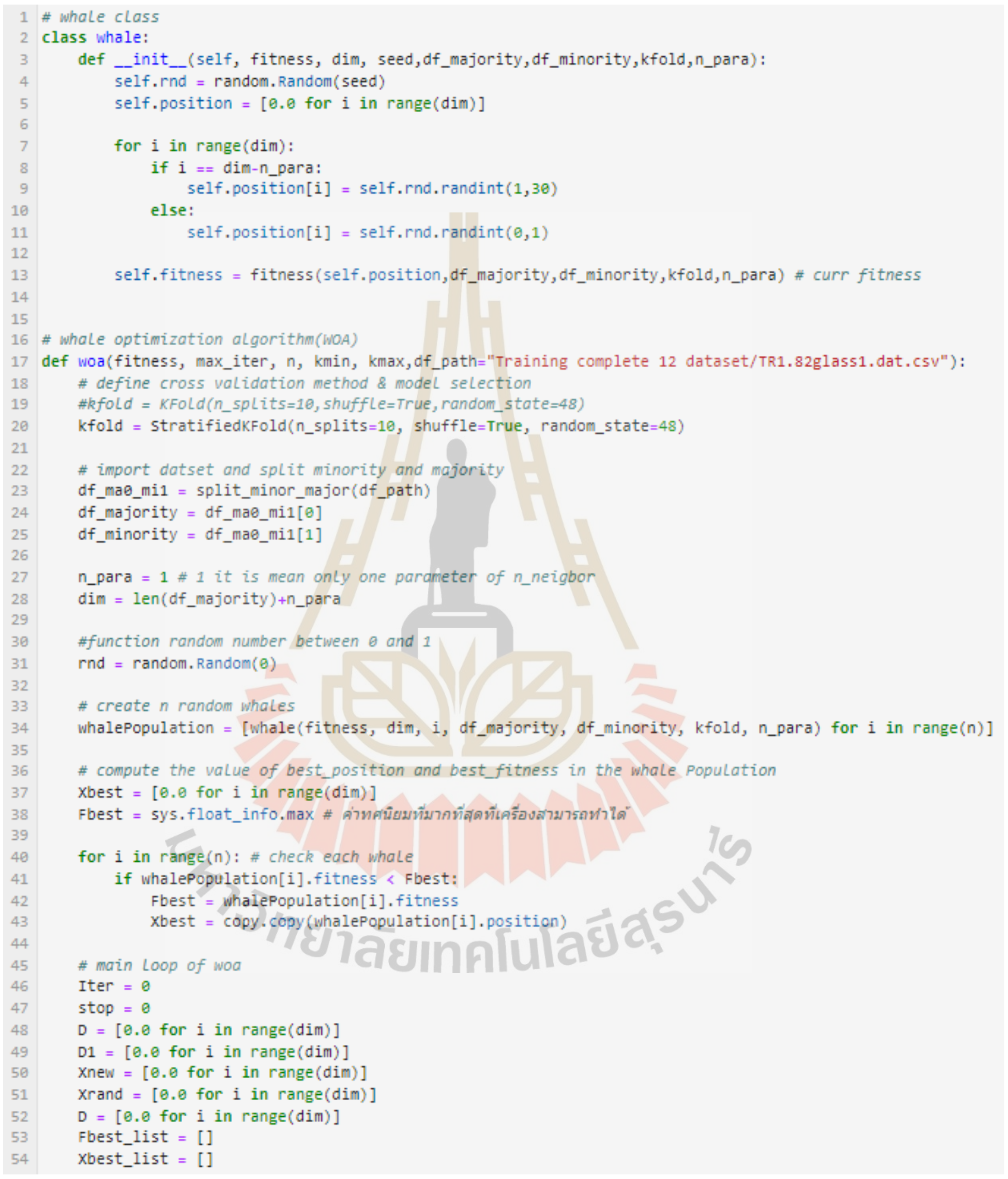
```python
def plot_fitness(table,X,Gbest_fit,fitness_bestt,elapsed_time,l_name='major'
                 ,path='Churn Dataset/Testing/',name ="data"):
    #save numpy array
    savetxt(path+"X"+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv'
            ,X,delimiter=',')
    savetxt(path+"F"+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv'
            ,Gbest_fit,delimiter=',')
    #save table
    table.to_csv(path+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.csv')
    xticks = range(1, len(Gbest_fit)+1)
    plt.figure(figsize=(10, 5))
    ax = plt.axes()
    ax.set_facecolor("white")
    plt.plot(xticks, Gbest_fit);
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title('dataset_name:'+str(name[:-8])+' best fitness = '+str(fitness_bestt)[0:8]+'
              total -time(sec) = '+str(elapsed_time)[0:6])
    #plt.pause(1e-10)
    plt.savefig(path+str(name[:-8])+'fitness'+str(fitness_bestt)[0:6]+l_name+'.png')
    plt.show()
```

Figure F.5 Code for creating function for graph fitness.



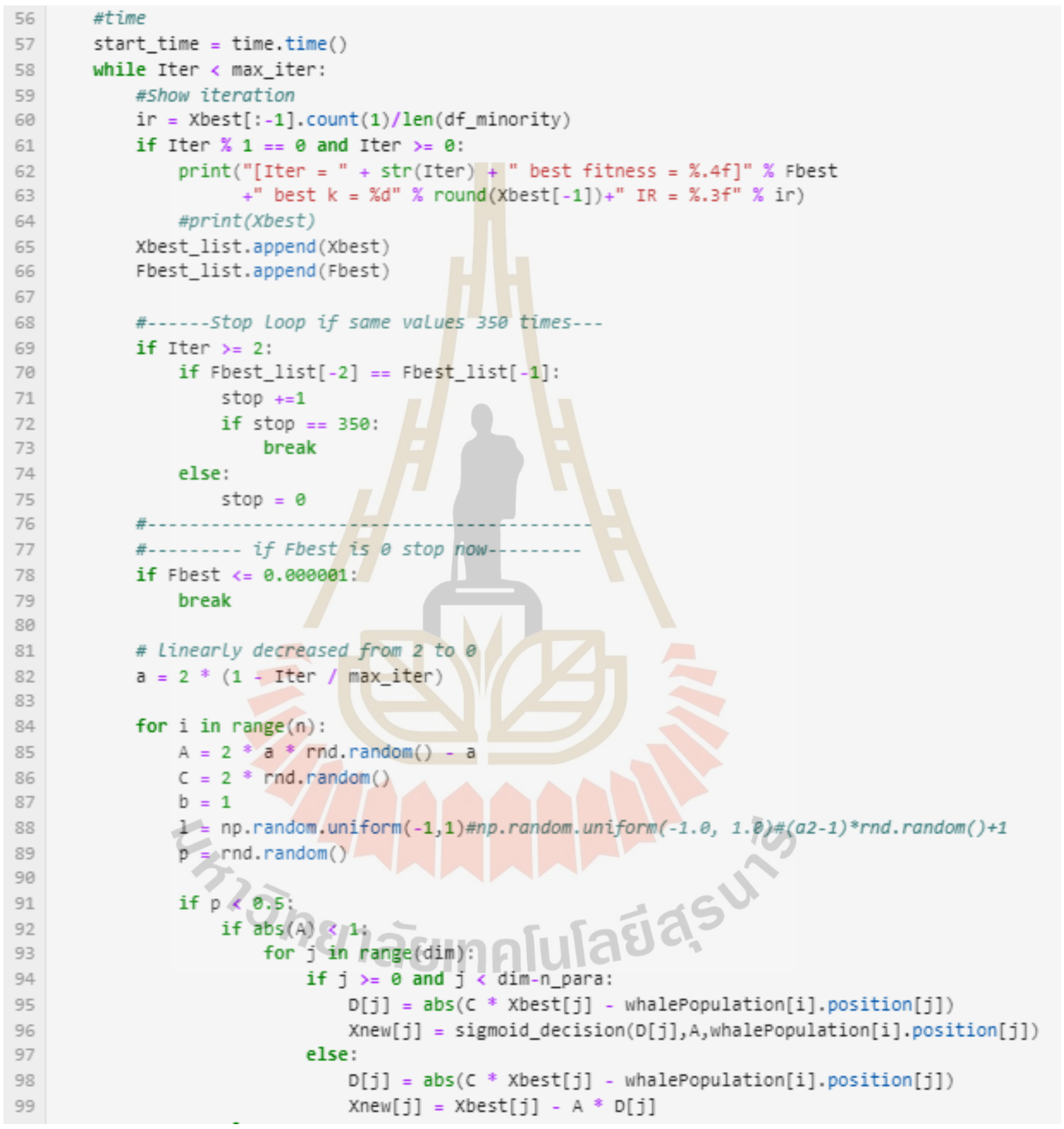
```python
#complement function
def complement(x):
    U = [0,1]
    a = U[x-1]
    return a
#Sigmoid function
def sigmoid_decision(Dist,A,x):
    Cstep = 1/(1+math.exp(-10*(A*Dist-0.5)))
    rr = random.Random().random()
    if rr < Cstep:
        Xnew = complement(x)
    else:
        Xnew = x
    return Xnew
# ------------------------
```

Figure F.6 Code for complement function.



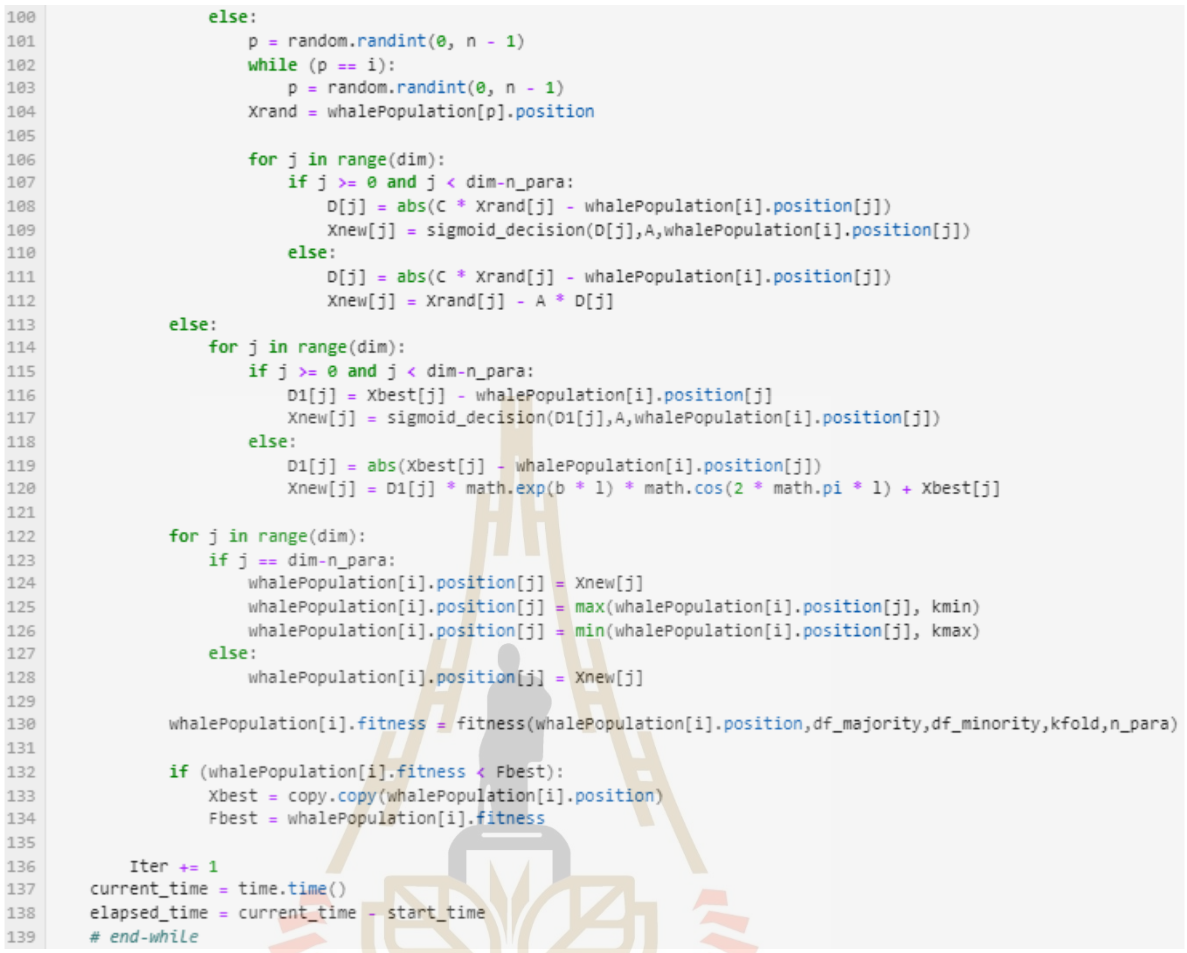
```python
# whale class
class whale:
    def __init__(self, fitness, dim, seed,df_majority,df_minority,kfold,n_para):
        self.rnd = random.Random(seed)
        self.position = [0.0 for i in range(dim)]

        for i in range(dim):
            if i == dim-n_para:
                self.position[i] = self.rnd.randint(1,30)
            else:
                self.position[i] = self.rnd.randint(0,1)

        self.fitness = fitness(self.position,df_majority,df_minority,kfold,n_para) # curr fitness


# whale optimization algorithm(WOA)
def woa(fitness, max_iter, n, kmin, kmax,df_path="Training complete 12 dataset/TR1.82glass1.dat.csv"):
    # define cross validation method & model selection
    #kfold = KFold(n_splits=10,shuffle=True,random_state=48)
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=48)

    # import datset and split minority and majority
    df_ma0_mi1 = split_minor_major(df_path)
    df_majority = df_ma0_mi1[0]
    df_minority = df_ma0_mi1[1]

    n_para = 1 # 1 it is mean only one parameter of n_neigbor
    dim = len(df_majority)+n_para

    #function random number between 0 and 1
    rnd = random.Random(0)

    # create n random whales
    whalePopulation = [whale(fitness, dim, i, df_majority, df_minority, kfold, n_para) for i in range(n)]

    # compute the value of best_position and best_fitness in the whale Population
    Xbest = [0.0 for i in range(dim)]
    Fbest = sys.float_info.max # ค่าทศนิยมที่มากที่สุดที่เครื่องสามารถทำได้

    for i in range(n): # check each whale
        if whalePopulation[i].fitness < Fbest:
            Fbest = whalePopulation[i].fitness
            Xbest = copy.copy(whalePopulation[i].position)

    # main loop of woa
    Iter = 0
    stop = 0
    D = [0.0 for i in range(dim)]
    D1 = [0.0 for i in range(dim)]
    Xnew = [0.0 for i in range(dim)]
    Xrand = [0.0 for i in range(dim)]
    D = [0.0 for i in range(dim)]
    Fbest_list = []
    Xbest_list = []
```

Figure F.7 Code for creating function for WBWOA KNN.



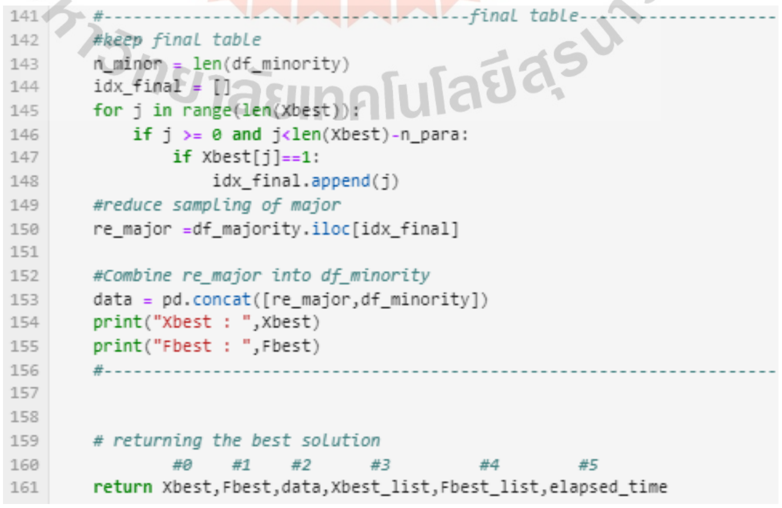
```python
    #time
    start_time = time.time()
    while Iter < max_iter:
        #Show iteration
        ir = Xbest[:-1].count(1)/len(df_minority)
        if Iter % 1 == 0 and Iter >= 0:
            print("[Iter = " + str(Iter) + " best fitness = %.4f]" % Fbest
                  +" best k = %d" % round(Xbest[-1])+" IR = %.3f" % ir)
            #print(Xbest)
        Xbest_list.append(Xbest)
        Fbest_list.append(Fbest)

        #------Stop Loop if same values 350 times---
        if Iter >= 2:
            if Fbest_list[-2] == Fbest_list[-1]:
                stop +=1
                if stop == 350:
                    break
            else:
                stop = 0
        #------------------------------------------
        #--------- if Fbest is 0 stop now--------
        if Fbest <= 0.000001:
            break

        # linearly decreased from 2 to 0
        a = 2 * (1 - Iter / max_iter)

        for i in range(n):
            A = 2 * a * rnd.random() - a
            C = 2 * rnd.random()
            b = 1
            l = np.random.uniform(-1,1)#np.random.uniform(-1.0, 1.0)#(a2-1)*rnd.random()+1
            p = rnd.random()

            if p < 0.5:
                if abs(A) < 1:
                    for j in range(dim):
                        if j >= 0 and j < dim-n_para:
                            D[j] = abs(C * Xbest[j] - whalePopulation[i].position[j])
                            Xnew[j] = sigmoid_decision(D[j],A,whalePopulation[i].position[j])
                        else:
                            D[j] = abs(C * Xbest[j] - whalePopulation[i].position[j])
                            Xnew[j] = Xbest[j] - A * D[j]
```

Figure F.8 Code for creating function for WBWOA KNN (Continued1).



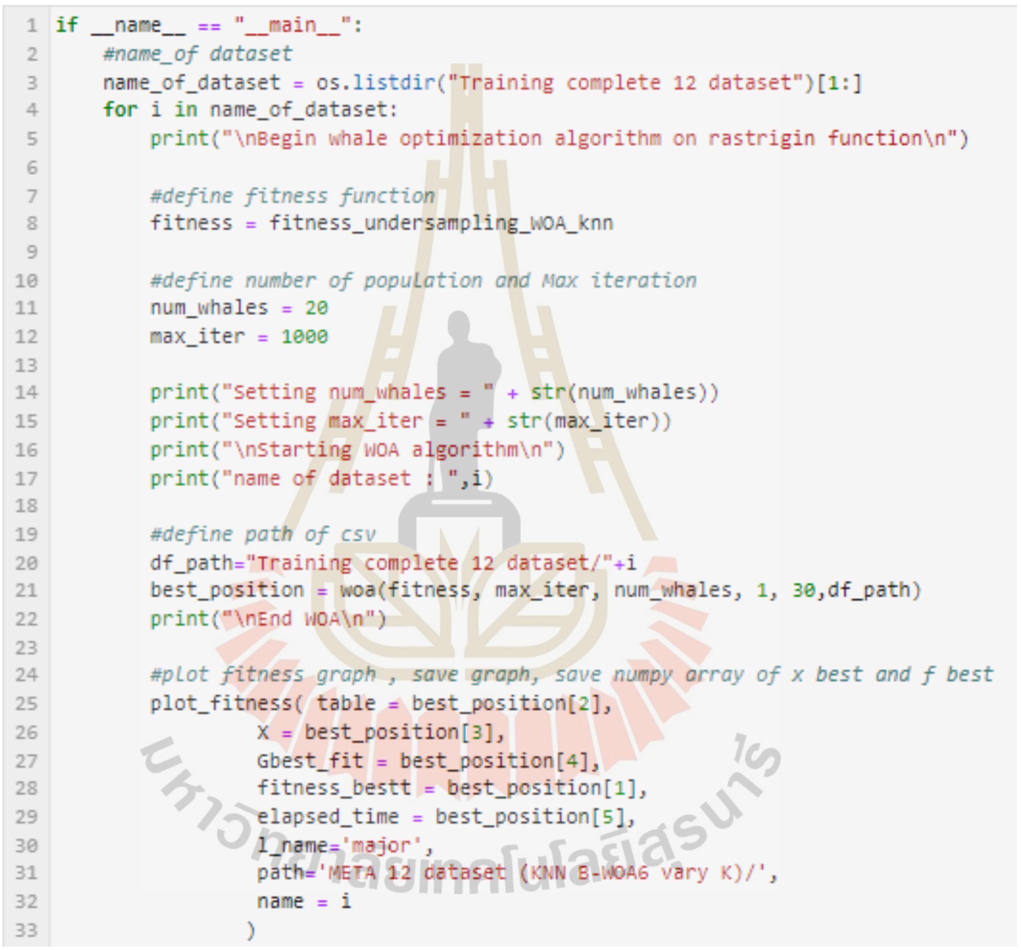
```
                else:
                    p = random.randint(0, n - 1)
                    while (p == i):
                        p = random.randint(0, n - 1)
                    Xrand = whalePopulation[p].position

                    for j in range(dim):
                        if j >= 0 and j < dim-n_para:
                            D[j] = abs(C * Xrand[j] - whalePopulation[i].position[j])
                            Xnew[j] = sigmoid_decision(D[j],A,whalePopulation[i].position[j])
                        else:
                            D[j] = abs(C * Xrand[j] - whalePopulation[i].position[j])
                            Xnew[j] = Xrand[j] - A * D[j]
            else:
                for j in range(dim):
                    if j >= 0 and j < dim-n_para:
                        D1[j] = Xbest[j] - whalePopulation[i].position[j]
                        Xnew[j] = sigmoid_decision(D1[j],A,whalePopulation[i].position[j])
                    else:
                        D1[j] = abs(Xbest[j] - whalePopulation[i].position[j])
                        Xnew[j] = D1[j] * math.exp(b * l) * math.cos(2 * math.pi * l) + Xbest[j]

            for j in range(dim):
                if j == dim-n_para:
                    whalePopulation[i].position[j] = Xnew[j]
                    whalePopulation[i].position[j] = max(whalePopulation[i].position[j], kmin)
                    whalePopulation[i].position[j] = min(whalePopulation[i].position[j], kmax)
                else:
                    whalePopulation[i].position[j] = Xnew[j]

            whalePopulation[i].fitness = fitness(whalePopulation[i].position,df_majority,df_minority,kfold,n_para)

            if (whalePopulation[i].fitness < Fbest):
                Xbest = copy.copy(whalePopulation[i].position)
                Fbest = whalePopulation[i].fitness

        Iter += 1
    current_time = time.time()
    elapsed_time = current_time - start_time
    # end-while
```

Figure F.9 Code for creating function for WBWOA KNN (Continued2).



```
    #-------------------------------------final table-----------------------
    #keep final table
    n_minor = len(df_minority)
    idx_final = []
    for j in range(len(Xbest)):
        if j >= 0 and j<len(Xbest)-n_para:
            if Xbest[j]==1:
                idx_final.append(j)
    #reduce sampling of major
    re_major =df_majority.iloc[idx_final]

    #Combine re_major into df_minority
    data = pd.concat([re_major,df_minority])
    print("Xbest : ",Xbest)
    print("Fbest : ",Fbest)
    #-----------------------------------------------------------------------


    # returning the best solution
            #0    #1    #2      #3         #4        #5
    return Xbest,Fbest,data,Xbest_list,Fbest_list,elapsed_time
```

Figure F.10 Code for creating function for WBWOA KNN (Continued3).

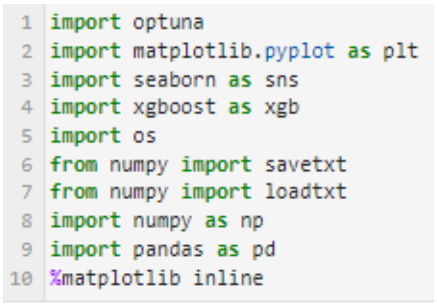


```python
if __name__ == "__main__":
    #name_of dataset
    name_of_dataset = os.listdir("Training complete 12 dataset")[1:]
    for i in name_of_dataset:
        print("\nBegin whale optimization algorithm on rastrigin function\n")

        #define fitness function
        fitness = fitness_undersampling_WOA_knn

        #define number of population and Max iteration
        num_whales = 20
        max_iter = 1000

        print("Setting num_whales = " + str(num_whales))
        print("Setting max_iter = " + str(max_iter))
        print("\nStarting WOA algorithm\n")
        print("name of dataset : ",i)

        #define path of csv
        df_path="Training complete 12 dataset/"+i
        best_position = woa(fitness, max_iter, num_whales, 1, 30,df_path)
        print("\nEnd WOA\n")

        #plot fitness graph , save graph, save numpy array of x best and f best
        plot_fitness( table = best_position[2],
                 X = best_position[3],
                 Gbest_fit = best_position[4],
                 fitness_bestt = best_position[1],
                 elapsed_time = best_position[5],
                 l_name='major',
                 path='META 12 dataset (KNN B-WOA6 vary K)/',
                 name = i
                )
```

Figure F.11 Code for running the process.

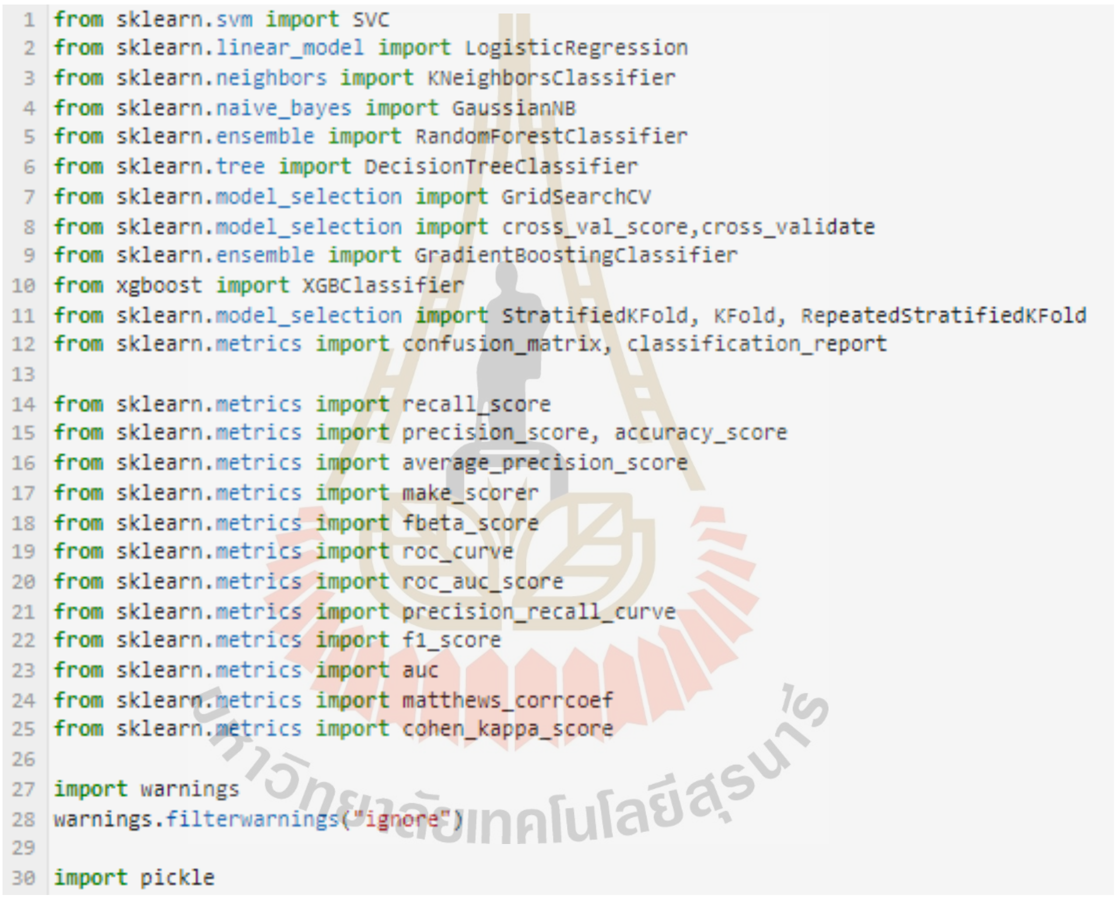
APPENDIX G

CODE FOR PARAMETER OPTIMIZATION



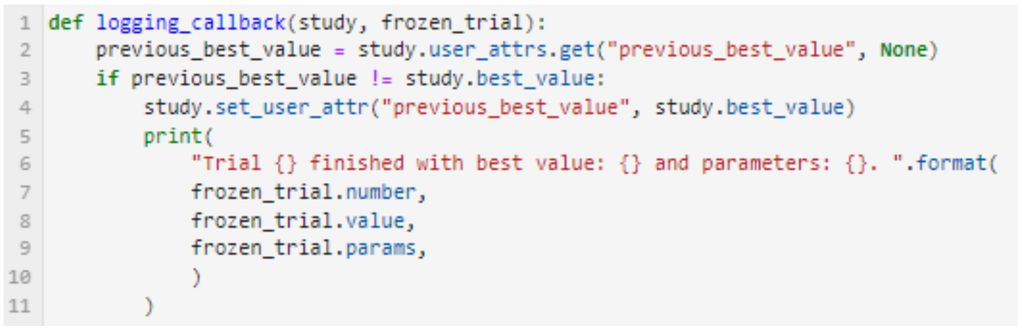
```python
import optuna
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
import os
from numpy import savetxt
from numpy import loadtxt
import numpy as np
import pandas as pd
%matplotlib inline
```

Figure G.1 Code for importing library.



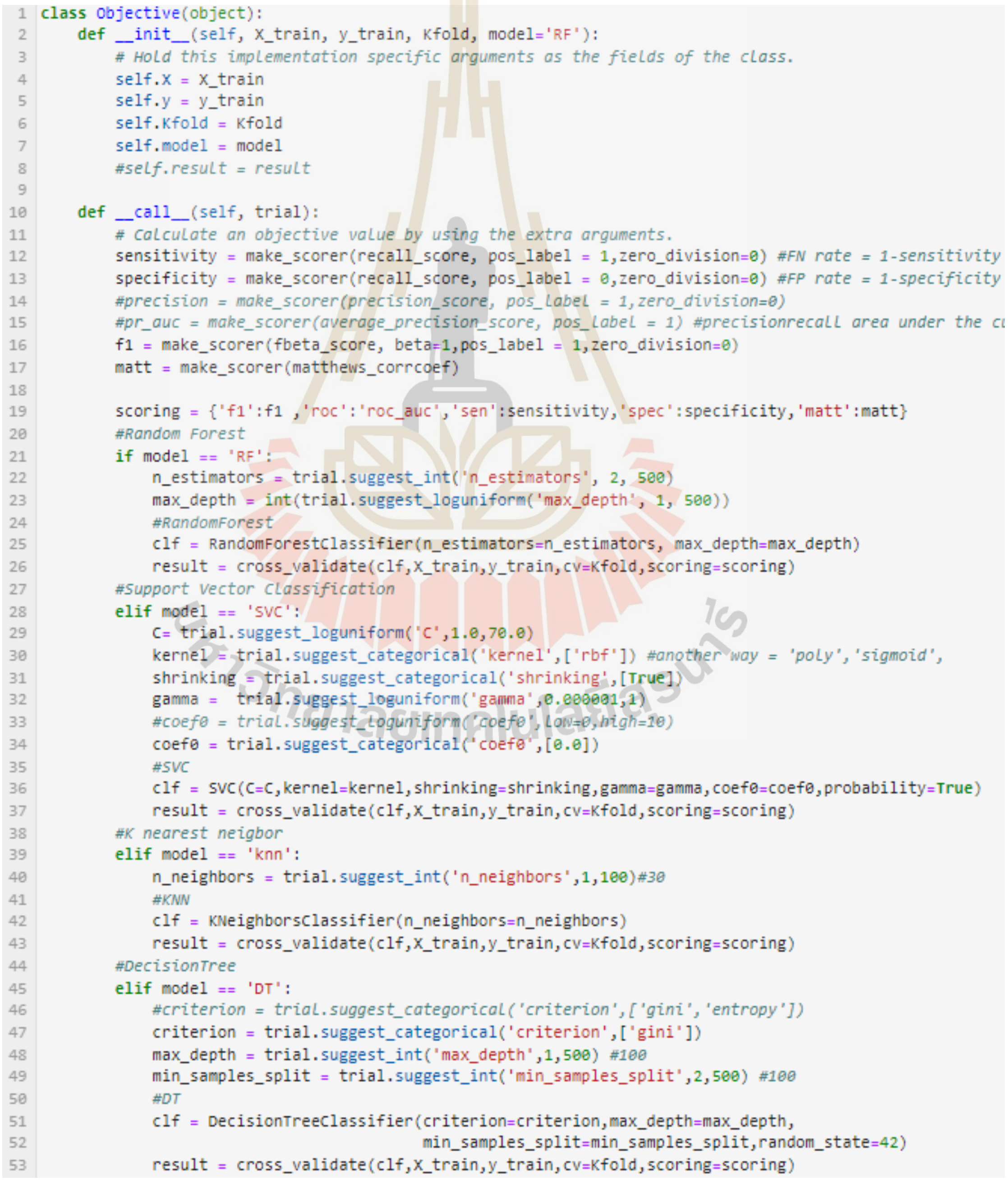
```python
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score,cross_validate
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold, KFold, RepeatedStratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.metrics import recall_score
from sklearn.metrics import precision_score, accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import make_scorer
from sklearn.metrics import fbeta_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import cohen_kappa_score

import warnings
warnings.filterwarnings("ignore")

import pickle
```

Figure G.2 Code for importing library (Continued).



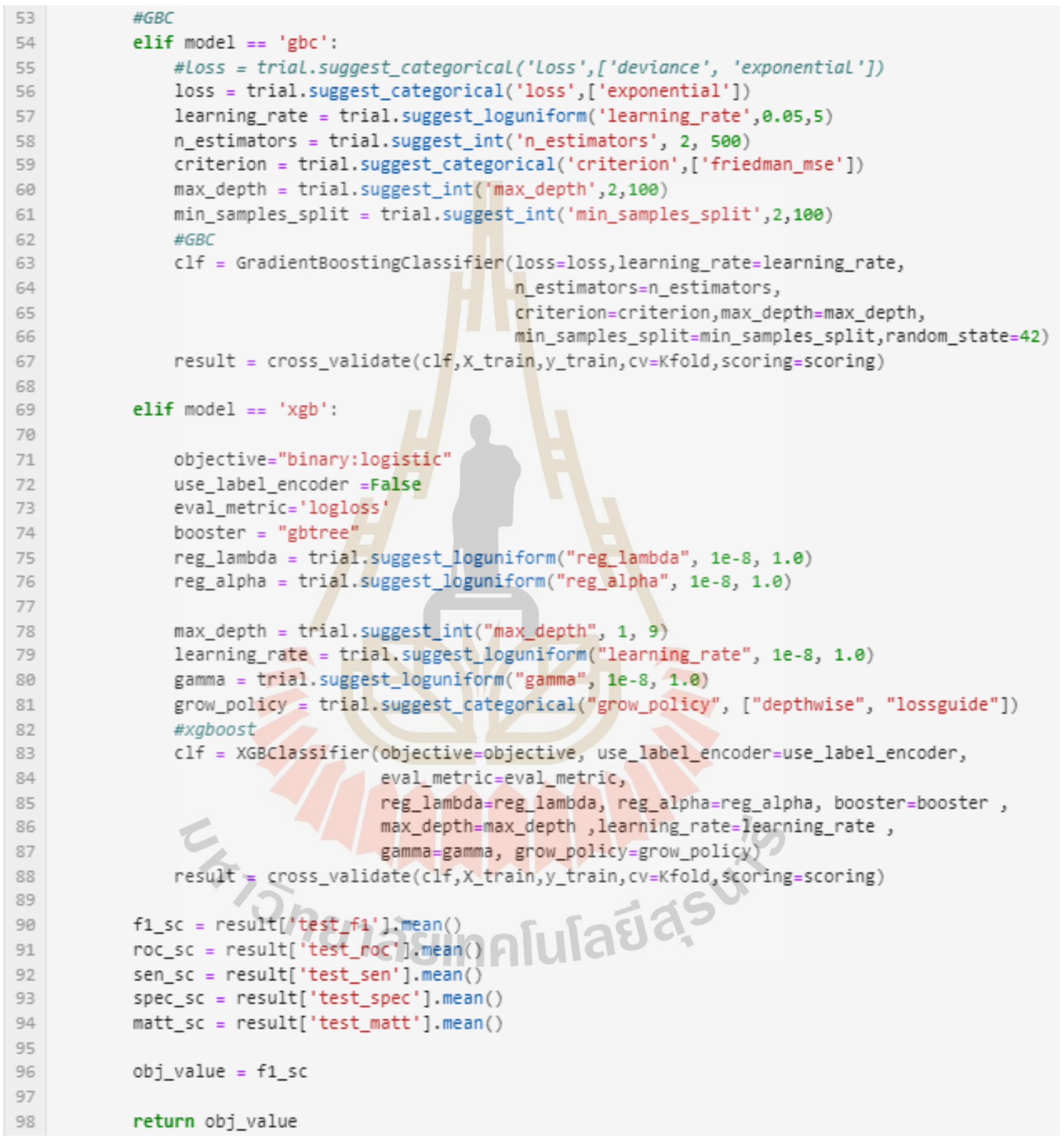
```python
def logging_callback(study, frozen_trial):
    previous_best_value = study.user_attrs.get("previous_best_value", None)
    if previous_best_value != study.best_value:
        study.set_user_attr("previous_best_value", study.best_value)
        print(
            "Trial {} finished with best value: {} and parameters: {}. ".format(
            frozen_trial.number,
            frozen_trial.value,
            frozen_trial.params,
            )
        )
```

Figure G.3 Code for ceating function for show the best parameter.



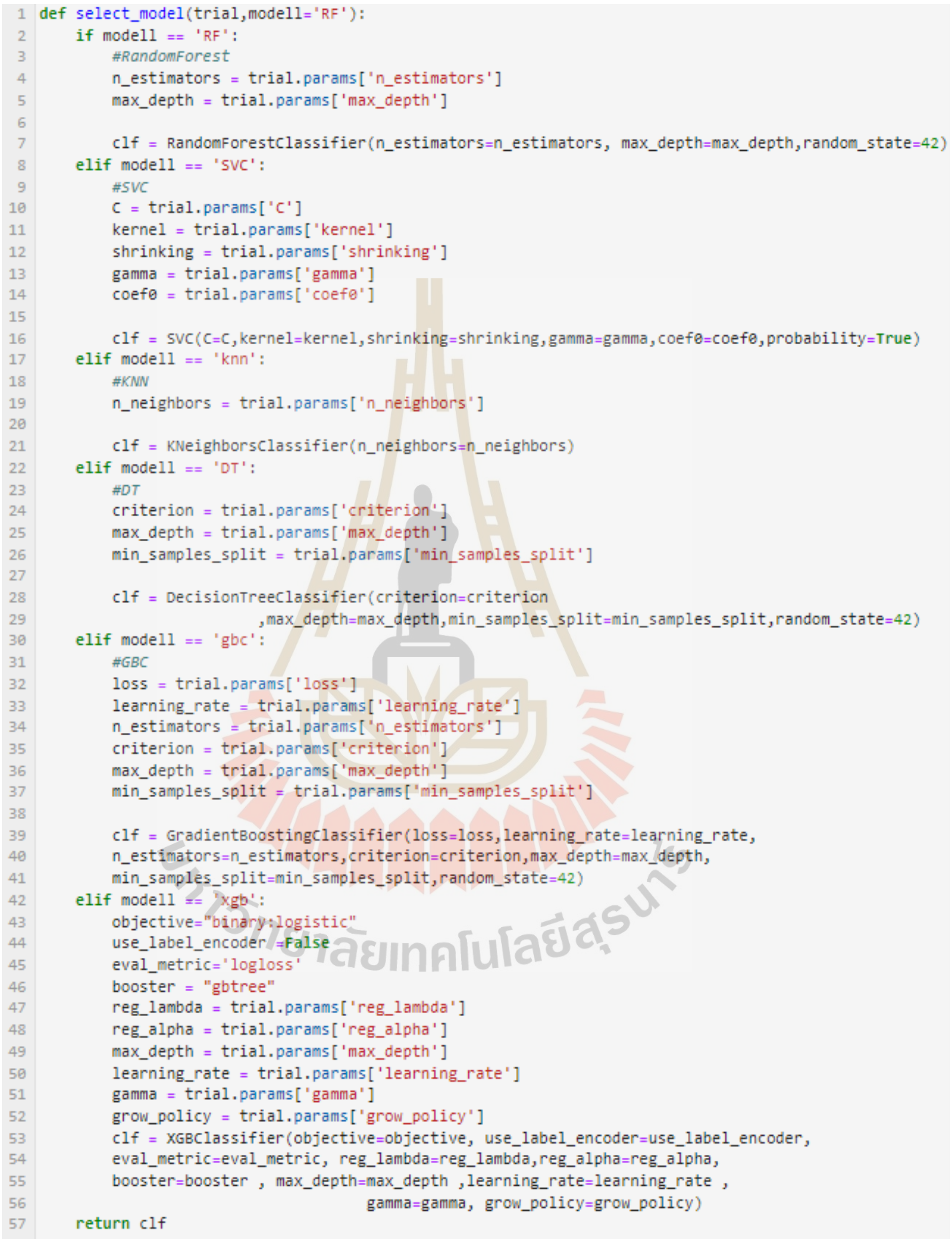
```python
class Objective(object):
    def __init__(self, X_train, y_train, Kfold, model='RF'):
        # Hold this implementation specific arguments as the fields of the class.
        self.X = X_train
        self.y = y_train
        self.Kfold = Kfold
        self.model = model
        #self.result = result

    def __call__(self, trial):
        # Calculate an objective value by using the extra arguments.
        sensitivity = make_scorer(recall_score, pos_label = 1,zero_division=0) #FN rate = 1-sensitivity
        specificity = make_scorer(recall_score, pos_label = 0,zero_division=0) #FP rate = 1-specificity
        #precision = make_scorer(precision_score, pos_label = 1,zero_division=0)
        #pr_auc = make_scorer(average_precision_score, pos_label = 1) #precisionrecall area under the c
        f1 = make_scorer(fbeta_score, beta=1,pos_label = 1,zero_division=0)
        matt = make_scorer(matthews_corrcoef)

        scoring = {'f1':f1 ,'roc':'roc_auc','sen':sensitivity,'spec':specificity,'matt':matt}
        #Random Forest
        if model == 'RF':
            n_estimators = trial.suggest_int('n_estimators', 2, 500)
            max_depth = int(trial.suggest_loguniform('max_depth', 1, 500))
            #RandomForest
            clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)
        #Support Vector Classification
        elif model == 'SVC':
            C= trial.suggest_loguniform('C',1.0,70.0)
            kernel = trial.suggest_categorical('kernel',['rbf']) #another way = 'poly','sigmoid',
            shrinking = trial.suggest_categorical('shrinking',[True])
            gamma =  trial.suggest_loguniform('gamma',0.000001,1)
            #coef0 = trial.suggest_loguniform('coef0',low=0,high=10)
            coef0 = trial.suggest_categorical('coef0',[0.0])
            #SVC
            clf = SVC(C=C,kernel=kernel,shrinking=shrinking,gamma=gamma,coef0=coef0,probability=True)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)
        #K nearest neigbor
        elif model == 'knn':
            n_neighbors = trial.suggest_int('n_neighbors',1,100)#30
            #KNN
            clf = KNeighborsClassifier(n_neighbors=n_neighbors)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)
        #DecisionTree
        elif model == 'DT':
            #criterion = trial.suggest_categorical('criterion',['gini','entropy'])
            criterion = trial.suggest_categorical('criterion',['gini'])
            max_depth = trial.suggest_int('max_depth',1,500) #100
            min_samples_split = trial.suggest_int('min_samples_split',2,500) #100
            #DT
            clf = DecisionTreeClassifier(criterion=criterion,max_depth=max_depth,
                                         min_samples_split=min_samples_split,random_state=42)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)
```

Figure G.4 Code for creating function for objective value of model.



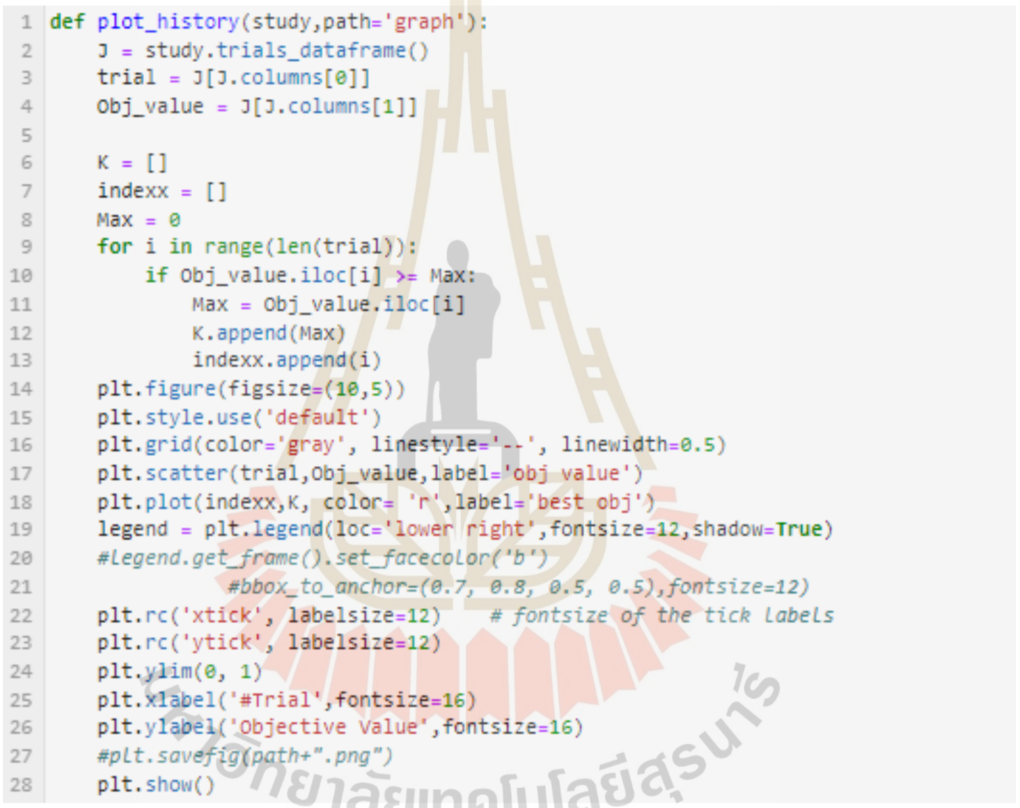
```
        #GBC
        elif model == 'gbc':
            #loss = trial.suggest_categorical('loss',['deviance', 'exponential'])
            loss = trial.suggest_categorical('loss',['exponential'])
            learning_rate = trial.suggest_loguniform('learning_rate',0.05,5)
            n_estimators = trial.suggest_int('n_estimators', 2, 500)
            criterion = trial.suggest_categorical('criterion',['friedman_mse'])
            max_depth = trial.suggest_int('max_depth',2,100)
            min_samples_split = trial.suggest_int('min_samples_split',2,100)
            #GBC
            clf = GradientBoostingClassifier(loss=loss,learning_rate=learning_rate,
                                             n_estimators=n_estimators,
                                             criterion=criterion,max_depth=max_depth,
                                             min_samples_split=min_samples_split,random_state=42)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)

        elif model == 'xgb':

            objective="binary:logistic"
            use_label_encoder =False
            eval_metric='logloss'
            booster = "gbtree"
            reg_lambda = trial.suggest_loguniform("reg_lambda", 1e-8, 1.0)
            reg_alpha = trial.suggest_loguniform("reg_alpha", 1e-8, 1.0)

            max_depth = trial.suggest_int("max_depth", 1, 9)
            learning_rate = trial.suggest_loguniform("learning_rate", 1e-8, 1.0)
            gamma = trial.suggest_loguniform("gamma", 1e-8, 1.0)
            grow_policy = trial.suggest_categorical("grow_policy", ["depthwise", "lossguide"])
            #xgboost
            clf = XGBClassifier(objective=objective, use_label_encoder=use_label_encoder,
                                eval_metric=eval_metric,
                                reg_lambda=reg_lambda, reg_alpha=reg_alpha, booster=booster ,
                                max_depth=max_depth ,learning_rate=learning_rate ,
                                gamma=gamma, grow_policy=grow_policy)
            result = cross_validate(clf,X_train,y_train,cv=Kfold,scoring=scoring)

        f1_sc = result['test_f1'].mean()
        roc_sc = result['test_roc'].mean()
        sen_sc = result['test_sen'].mean()
        spec_sc = result['test_spec'].mean()
        matt_sc = result['test_matt'].mean()

        obj_value = f1_sc

        return obj_value
```

Figure G.5 Code for creating function for objective value of model (Continued).



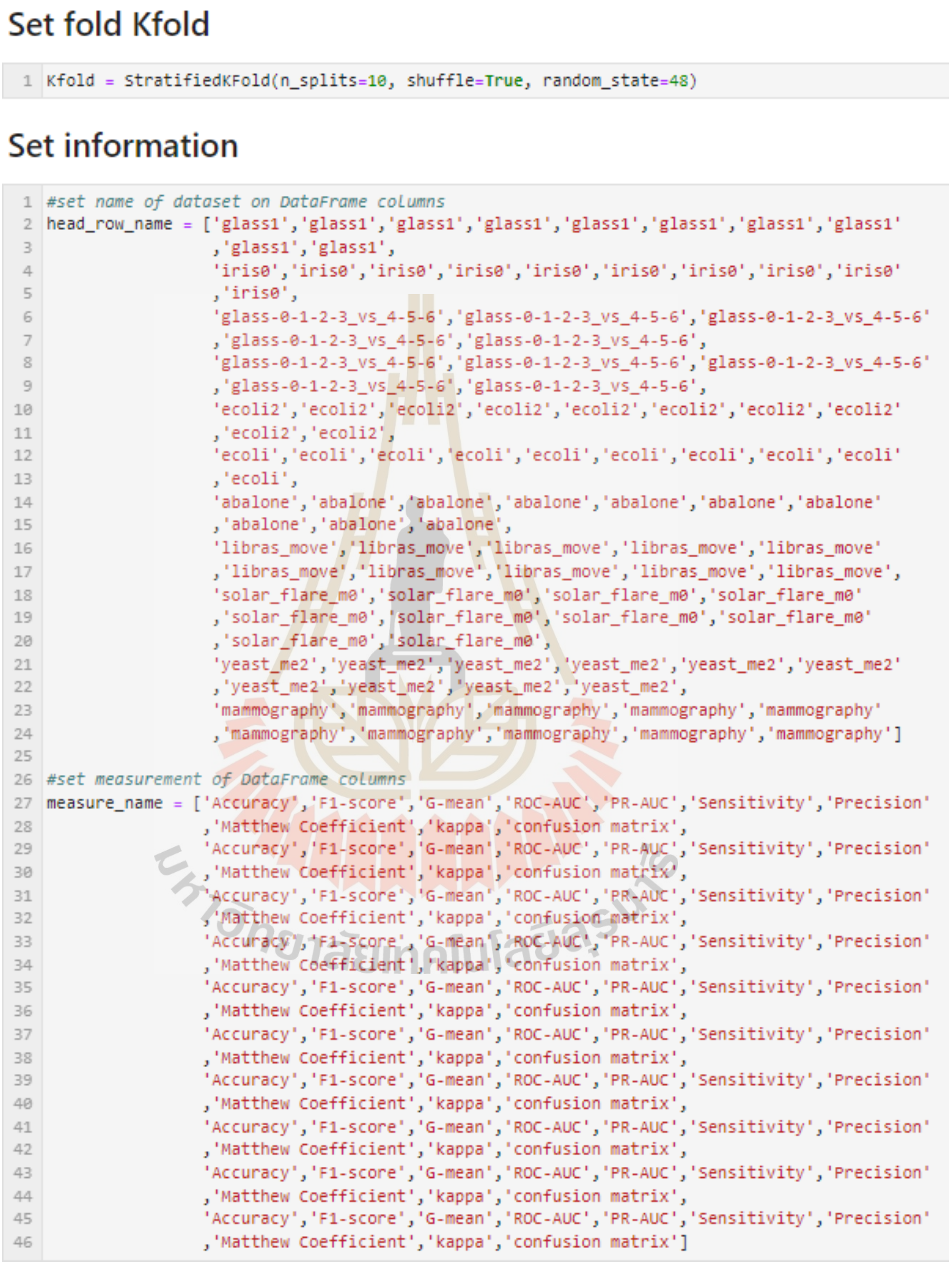
```python
def select_model(trial,modell='RF'):
    if modell == 'RF':
        #RandomForest
        n_estimators = trial.params['n_estimators']
        max_depth = trial.params['max_depth']

        clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,random_state=42)
    elif modell == 'SVC':
        #SVC
        C = trial.params['C']
        kernel = trial.params['kernel']
        shrinking = trial.params['shrinking']
        gamma = trial.params['gamma']
        coef0 = trial.params['coef0']

        clf = SVC(C=C,kernel=kernel,shrinking=shrinking,gamma=gamma,coef0=coef0,probability=True)
    elif modell == 'knn':
        #KNN
        n_neighbors = trial.params['n_neighbors']

        clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    elif modell == 'DT':
        #DT
        criterion = trial.params['criterion']
        max_depth = trial.params['max_depth']
        min_samples_split = trial.params['min_samples_split']

        clf = DecisionTreeClassifier(criterion=criterion
                        ,max_depth=max_depth,min_samples_split=min_samples_split,random_state=42)
    elif modell == 'gbc':
        #GBC
        loss = trial.params['loss']
        learning_rate = trial.params['learning_rate']
        n_estimators = trial.params['n_estimators']
        criterion = trial.params['criterion']
        max_depth = trial.params['max_depth']
        min_samples_split = trial.params['min_samples_split']

        clf = GradientBoostingClassifier(loss=loss,learning_rate=learning_rate,
        n_estimators=n_estimators,criterion=criterion,max_depth=max_depth,
        min_samples_split=min_samples_split,random_state=42)
    elif modell == 'xgb':
        objective="binary:logistic"
        use_label_encoder =False
        eval_metric='logloss'
        booster = "gbtree"
        reg_lambda = trial.params['reg_lambda']
        reg_alpha = trial.params['reg_alpha']
        max_depth = trial.params['max_depth']
        learning_rate = trial.params['learning_rate']
        gamma = trial.params['gamma']
        grow_policy = trial.params['grow_policy']
        clf = XGBClassifier(objective=objective, use_label_encoder=use_label_encoder,
        eval_metric=eval_metric, reg_lambda=reg_lambda,reg_alpha=reg_alpha,
        booster=booster , max_depth=max_depth ,learning_rate=learning_rate ,
                            gamma=gamma, grow_policy=grow_policy)
    return clf
```

Figure G.6 Code for creating function for select models.



```python
def plot_history(study,path='graph'):
    J = study.trials_dataframe()
    trial = J[J.columns[0]]
    Obj_value = J[J.columns[1]]

    K = []
    indexx = []
    Max = 0
    for i in range(len(trial)):
        if Obj_value.iloc[i] >= Max:
            Max = Obj_value.iloc[i]
            K.append(Max)
            indexx.append(i)
    plt.figure(figsize=(10,5))
    plt.style.use('default')
    plt.grid(color='gray', linestyle='--', linewidth=0.5)
    plt.scatter(trial,Obj_value,label='obj value')
    plt.plot(indexx,K, color= 'r',label='best obj')
    legend = plt.legend(loc='lower right',fontsize=12,shadow=True)
    #legend.get_frame().set_facecolor('b')
                 #bbox_to_anchor=(0.7, 0.8, 0.5, 0.5),fontsize=12)
    plt.rc('xtick', labelsize=12)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=12)
    plt.ylim(0, 1)
    plt.xlabel('#Trial',fontsize=16)
    plt.ylabel('Objective Value',fontsize=16)
    #plt.savefig(path+".png")
    plt.show()
```

Figure G.7 Code for ploting graph.



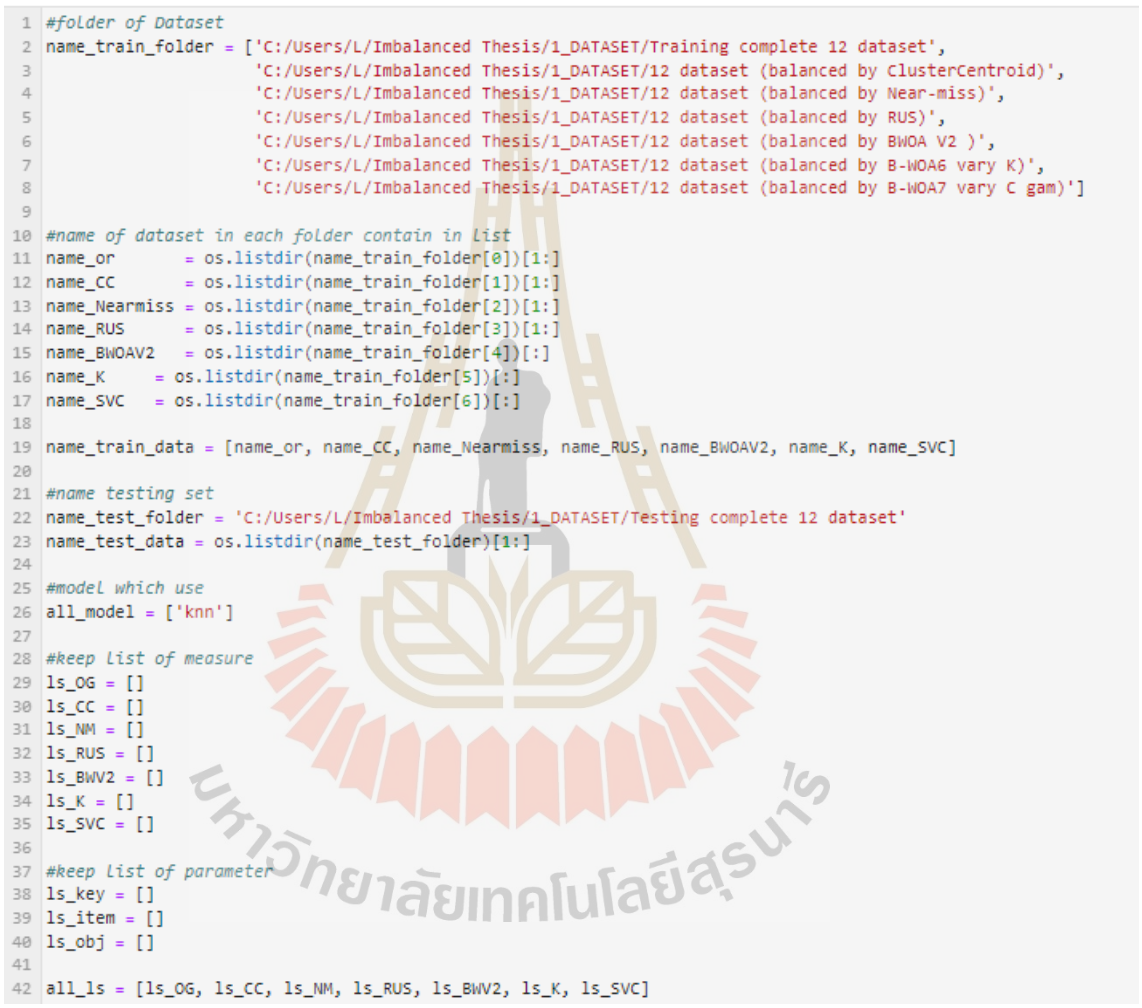
Set fold Kfold

```python
Kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=48)
```

Set information

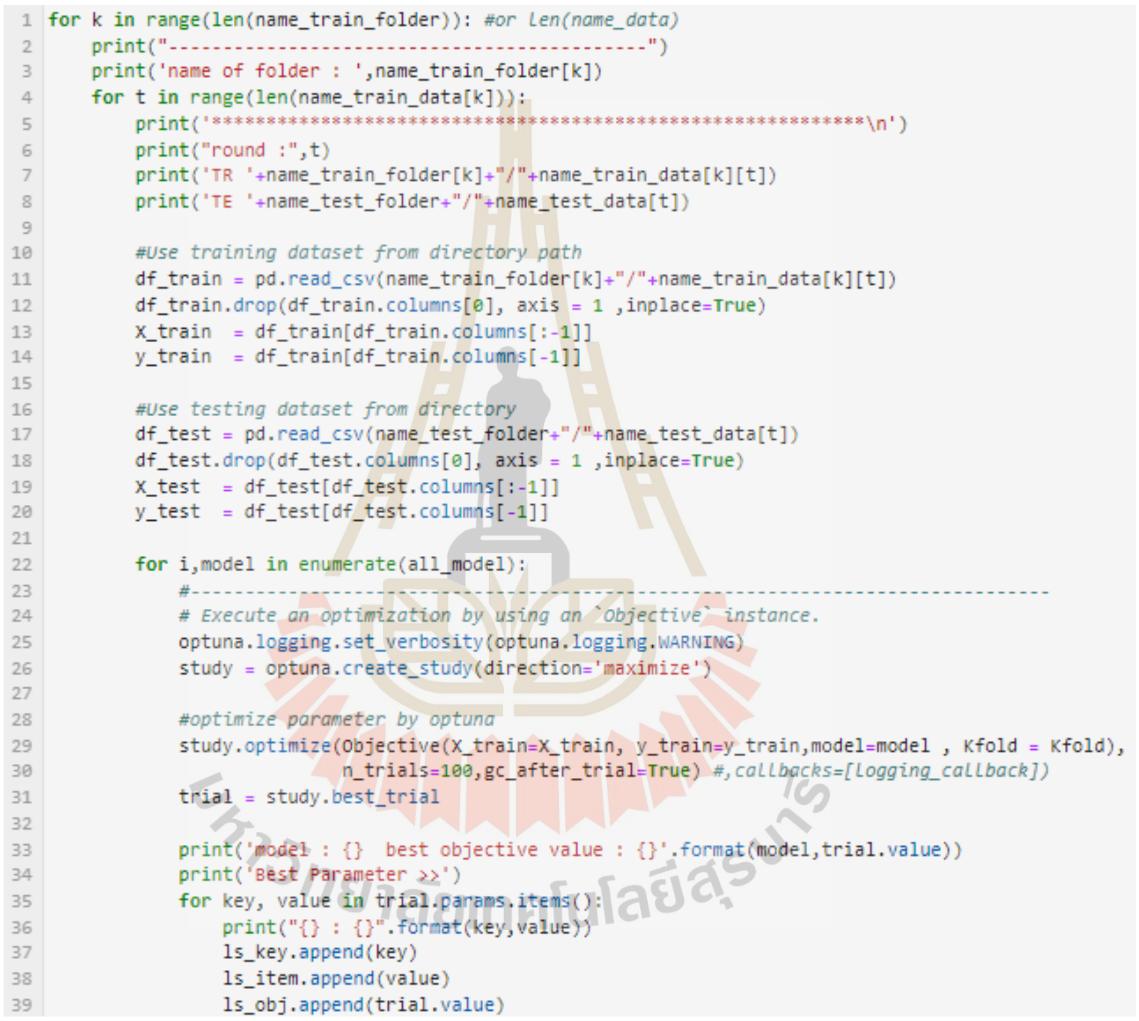
```python
#set name of dataset on DataFrame columns
head_row_name = ['glass1','glass1','glass1','glass1','glass1','glass1','glass1','glass1'
                  ,'glass1','glass1',
                  'iris0','iris0','iris0','iris0','iris0','iris0','iris0','iris0','iris0'
                  ,'iris0',
                  'glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6'
                  ,'glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6',
                  'glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6'
                  ,'glass-0-1-2-3_vs_4-5-6','glass-0-1-2-3_vs_4-5-6',
                  'ecoli2','ecoli2','ecoli2','ecoli2','ecoli2','ecoli2','ecoli2','ecoli2'
                  ,'ecoli2','ecoli2',
                  'ecoli','ecoli','ecoli','ecoli','ecoli','ecoli','ecoli','ecoli','ecoli'
                  ,'ecoli',
                  'abalone','abalone','abalone','abalone','abalone','abalone','abalone'
                  ,'abalone','abalone','abalone',
                  'libras_move','libras_move','libras_move','libras_move','libras_move'
                  ,'libras_move','libras_move','libras_move','libras_move','libras_move',
                  'solar_flare_m0','solar_flare_m0','solar_flare_m0','solar_flare_m0'
                  ,'solar_flare_m0','solar_flare_m0','solar_flare_m0','solar_flare_m0'
                  ,'solar_flare_m0','solar_flare_m0',
                  'yeast_me2','yeast_me2','yeast_me2','yeast_me2','yeast_me2','yeast_me2'
                  ,'yeast_me2','yeast_me2','yeast_me2','yeast_me2',
                  'mammography','mammography','mammography','mammography','mammography'
                  ,'mammography','mammography','mammography','mammography','mammography']

#set measurement of DataFrame columns
measure_name = ['Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix',
                 'Accuracy','F1-score','G-mean','ROC-AUC','PR-AUC','Sensitivity','Precision'
                 ,'Matthew Coefficient','kappa','confusion matrix']
```

Figure G.8 Code for setting name of table.



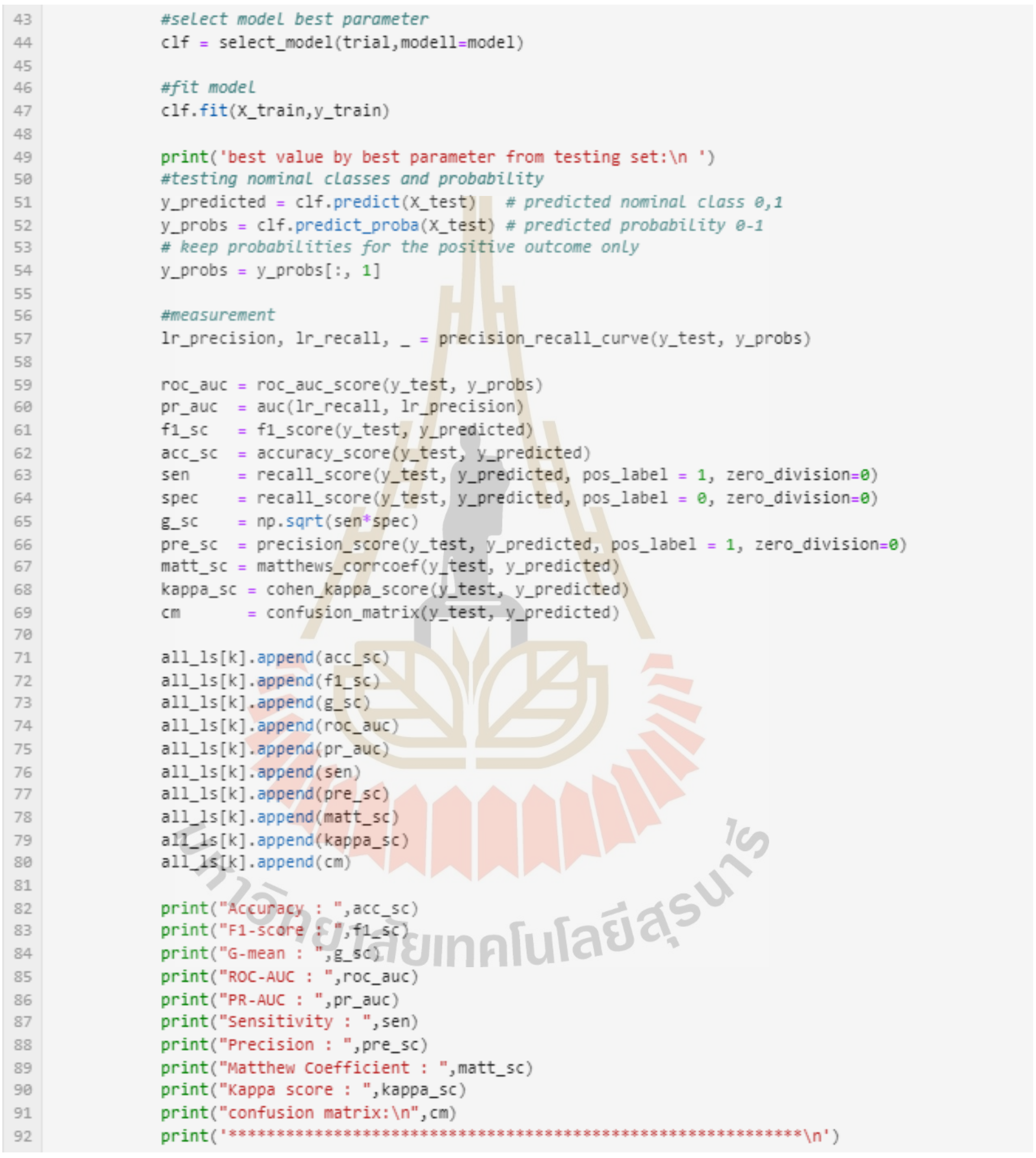
```python
#folder of Dataset
name_train_folder = ['C:/Users/L/Imbalanced Thesis/1_DATASET/Training complete 12 dataset',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by ClusterCentroid)',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by Near-miss)',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by RUS)',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by BWOA V2 )',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by B-WOA6 vary K)',
                     'C:/Users/L/Imbalanced Thesis/1_DATASET/12 dataset (balanced by B-WOA7 vary C gam)']

#name of dataset in each folder contain in List
name_or       = os.listdir(name_train_folder[0])[1:]
name_CC       = os.listdir(name_train_folder[1])[1:]
name_Nearmiss = os.listdir(name_train_folder[2])[1:]
name_RUS      = os.listdir(name_train_folder[3])[1:]
name_BWOAV2   = os.listdir(name_train_folder[4])[:]
name_K     = os.listdir(name_train_folder[5])[:]
name_SVC   = os.listdir(name_train_folder[6])[:]

name_train_data = [name_or, name_CC, name_Nearmiss, name_RUS, name_BWOAV2, name_K, name_SVC]

#name testing set
name_test_folder = 'C:/Users/L/Imbalanced Thesis/1_DATASET/Testing complete 12 dataset'
name_test_data = os.listdir(name_test_folder)[1:]

#model which use
all_model = ['knn']

#keep List of measure
ls_OG = []
ls_CC = []
ls_NM = []
ls_RUS = []
ls_BWV2 = []
ls_K = []
ls_SVC = []

#keep List of parameter
ls_key = []
ls_item = []
ls_obj = []

all_ls = [ls_OG, ls_CC, ls_NM, ls_RUS, ls_BWV2, ls_K, ls_SVC]
```

Figure G.9 Code for setting folder.



```python
for k in range(len(name_train_folder)): #or len(name_data)
    print("--------------------------------------------")
    print('name of folder : ',name_train_folder[k])
    for t in range(len(name_train_data[k])):
        print('************************************************************\n')
        print("round :",t)
        print('TR '+name_train_folder[k]+"/"+name_train_data[k][t])
        print('TE '+name_test_folder+"/"+name_test_data[t])

        #Use training dataset from directory path
        df_train = pd.read_csv(name_train_folder[k]+"/"+name_train_data[k][t])
        df_train.drop(df_train.columns[0], axis = 1 ,inplace=True)
        X_train  = df_train[df_train.columns[:-1]]
        y_train  = df_train[df_train.columns[-1]]

        #Use testing dataset from directory
        df_test = pd.read_csv(name_test_folder+"/"+name_test_data[t])
        df_test.drop(df_test.columns[0], axis = 1 ,inplace=True)
        X_test  = df_test[df_test.columns[:-1]]
        y_test  = df_test[df_test.columns[-1]]

        for i,model in enumerate(all_model):
            #--------------------------------------------------------------------------------
            # Execute an optimization by using an `Objective` instance.
            optuna.logging.set_verbosity(optuna.logging.WARNING)
            study = optuna.create_study(direction='maximize')

            #optimize parameter by optuna
            study.optimize(Objective(X_train=X_train, y_train=y_train,model=model , Kfold = Kfold),
                           n_trials=100,gc_after_trial=True) #,callbacks=[logging_callback])
            trial = study.best_trial

            print('model : {}  best objective value : {}'.format(model,trial.value))
            print('Best Parameter >>')
            for key, value in trial.params.items():
                print("{} : {}".format(key,value))
                ls_key.append(key)
                ls_item.append(value)
                ls_obj.append(trial.value)
```

Figure G.10 Code for running process work.



```
            #select model best parameter
            clf = select_model(trial,modell=model)

            #fit model
            clf.fit(X_train,y_train)

            print('best value by best parameter from testing set:\n ')
            #testing nominal classes and probability
            y_predicted = clf.predict(X_test)   # predicted nominal class 0,1
            y_probs = clf.predict_proba(X_test) # predicted probability 0-1
            # keep probabilities for the positive outcome only
            y_probs = y_probs[:, 1]

            #measurement
            lr_precision, lr_recall, _ = precision_recall_curve(y_test, y_probs)

            roc_auc = roc_auc_score(y_test, y_probs)
            pr_auc  = auc(lr_recall, lr_precision)
            f1_sc   = f1_score(y_test, y_predicted)
            acc_sc  = accuracy_score(y_test, y_predicted)
            sen     = recall_score(y_test, y_predicted, pos_label = 1, zero_division=0)
            spec    = recall_score(y_test, y_predicted, pos_label = 0, zero_division=0)
            g_sc    = np.sqrt(sen*spec)
            pre_sc  = precision_score(y_test, y_predicted, pos_label = 1, zero_division=0)
            matt_sc = matthews_corrcoef(y_test, y_predicted)
            kappa_sc = cohen_kappa_score(y_test, y_predicted)
            cm       = confusion_matrix(y_test, y_predicted)

            all_ls[k].append(acc_sc)
            all_ls[k].append(f1_sc)
            all_ls[k].append(g_sc)
            all_ls[k].append(roc_auc)
            all_ls[k].append(pr_auc)
            all_ls[k].append(sen)
            all_ls[k].append(pre_sc)
            all_ls[k].append(matt_sc)
            all_ls[k].append(kappa_sc)
            all_ls[k].append(cm)

            print("Accuracy : ",acc_sc)
            print("F1-score : ",f1_sc)
            print("G-mean : ",g_sc)
            print("ROC-AUC : ",roc_auc)
            print("PR-AUC : ",pr_auc)
            print("Sensitivity : ",sen)
            print("Precision : ",pre_sc)
            print("Matthew Coefficient : ",matt_sc)
            print("Kappa score : ",kappa_sc)
            print("confusion matrix:\n",cm)
            print('*************************************************************\n')
```

Figure G.11 Code for running process work (Continued).

CURRICULUM VITAE

NAME : Jakkrit Polrob **GENDER :** Male

EDUCATION BACKGROUND:

- Bachelor of Engineering (Aeronautical Engineering), Suranaree University of Technology, Thailand, 2020

SCHOLARSHIP:

- Kittibandit Scholarship of Suranaree University of Technology

CONFERENCE:

- Polrob, J., Rodjanadid B., Tanthanuch, J., and Schulz E. (2022) Binary Whale Optimization Algorithm for Improving Data Balance Based on Undersampling Techniques., **The 26th Annual Meeting in Mathematics 2022 (AMM 2022) and The 1st International Annual Meeting in Mathematics 2022 May 18 - 20, 2022**, Suranaree University of Technology, Nakhon Ratchasima, May 18th, 2022 (40-57)

EXPERIENCE:

- Teaching assistant in Suranaree University of Technology, Mathematics in daily life (Thai course) Term 1/2020 and Term 2/2020, Calculus I (Thai course) Term 2/2021, Essential Calculus (Thai course) Term 2/2021, Statistical Methods (Thai course) Term 3/2021, Probability and Statistics (Thai course) Term 3/2021, and Calculus III (Thai course) Term 3/2021