HYBRID CLUSTER-BASED SAMPLING TECHNIQUE FOR CLASS IMBALANCE PROBLEMS

BY

BERNARD KUDITCHAR

(10638513)

THIS THESIS IS SUBMITTED TO THE UNIVERSITY OF GHANA, LEGON IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF MPHIL COMPUTER ENGINEERING DEGREE

DEPARTMENT OF COMPUTER ENGINEERING

SCHOOL OF ENGINEERING SCIENCES

UNIVERSITY OF GHANA, LEGON

JULY 2019

DECLARATION

I, Bernard Kuditchar, hereby declare that this thesis document except where indicated by referencing, is my own work carried out under supervision in the Department of Computer Engineering, School of Engineering Sciences, University of Ghana, Legon. I further declare that this thesis, either in whole or in part, has not been presented for another degree in this University or elsewhere.

……..…….. ……..

Bernard Kuditchar Date

(Student)

……. ……

Dr Robert A. Sowah Date

(Principal Supervisor)

DEDICATION

This work is dedicated to GOD ALMIGHTY and the memory of my father, Ernest Kuditchar.

ACKNOWLEDGEMENTS

The successful completion of this work has been made possible by a great deal of support and assistance. I wish to extend my utmost appreciation to my supervisor, Dr Robert Sowah, for his invaluable contributions, guidance, and encouragements towards the successful completion of this research. I also wish to thank all members of Department of Computer Engineering, especially Dr Godfrey Mills, Dr Wiafe O. Banahene and the entire graduate committee of the department for their invaluable counsel and input to this work.

Finally, my gratitude to Moses Agebure, whose MPhil thesis work inspired this research.

ABSTRACT

Class imbalance problem is prevalent in many real-world domains and as such has become an area of increasing interest for many researchers. In binary classification problems, imbalance learning refers to learning from a dataset with a high degree of skewness to the negative class. This phenomenon causes traditional classification algorithms to perform woefully when predicting positive classes with new examples. Data resampling is among the most commonly used techniques used to deal with this problem. It involves the manipulation of the training data before applying standard classification techniques. This study presents a new hybrid sampling technique that has the capability of improving the overall performance of a wide range of traditional machine learning algorithms. The proposed method uses an undersampling technique based on CUST to under-sample majority instances and an oversampling technique derived from SNOCC to oversample minority instances. The method is implemented in Python version 3.5 on windows. The performance was evaluated using classification algorithms from ***scikit-learn*** machine learning library, namely: KNN, SVM, Decision Tree, Random Forest, Neural Network, AdaBoost, Naïve Bayes, and Quadratic Discriminant Analysis. Eleven datasets with various degrees of imbalance were used. The performance of each classifier when the proposed technique is used is compared with the performance when no sampling is performed. In addition to that, the performance of eight (8) other sampling techniques is compared with that of the proposed method. These techniques include ROS, RUS, SMOTE, ADASYN, CUST, SBC, CLUS, and OSS. The experimental results showed that HCBST performed better with most of the classifiers in terms of AUC, G-Mean, and MCC. The overall average performance also showed that HCBT performed better in most of the datasets, having the highest average scores of 0.73, 0.67 and 0.35 in AUC, G-Mean and MCC respectively across all the classifiers used for this study. Extensive testing of machine learning

algorithms and their performance metrics yielded promising results. A Graphical User Interface (GUI) to enable interactivity for machine learning with class imbalanced data task operations was incorporated to allow flexibility in the choice of algorithms for certain datasets for higher accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADASYN	Adaptive Synthetic Sampling
AUC	Area Under the Curve
CUST	Cluster Undersampling Technique
DT	Decision Trees
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
G-Mean	Geometric Mean
HCBST	Hybrid Cluster-Based Sampling Technique
KNN	K Nearest Neighbours
MCC	Matthews Correlation Coefficient
NB	Naïve Bayes
NN	Neural Networks
OSS	One-Sided Selection
QDA	Quadratic Discriminant Analysis
RF	Random Forest
ROS	Random Oversampling
RUS	Random Undersampling
SBC	Under-Sampling Based on Clustering
SMOTE	Synthetic Minority Oversampling Technique
SNOCC	Sigma Nearest Based on Convex Combination
SVM	Support Vector Machines
TN	True Negatives

TP	True Positives
TPR	True Positive Rate
CART	Classification And Regression Trees
MLP	Multilayer Perceptron

CHAPTER 1

INTRODUCTION

1.0 Introduction

Machine Learning (ML) is a branch of Artificial intelligence that allows a system to learn from data without explicitly programming it. Three main approaches are used to tackle Machine Learning, namely supervised, unsupervised, and semi-supervised methods [1]. Unsupervised data denotes a systematic process to extract intrinsic characteristics of the provided data [2],[3],[4]. The semi-supervised approach involves learning from data typically having a large proportion of unlabelled and a small proportion of labelled data [5]. The supervised approach is targeted at predicting categories from prior labelled data [6]. Supervised learning tasks can be grouped into regression (continuous output) and classification (categorical outputs) tasks.

Classification in supervised learning tasks involves learning from labelled categorical data to predict the category or class of unseen data [7]. One of the main challenges of supervised classification is the problem of class imbalance. This problem refers to the phenomenon where a dataset has more representation of one class compared to the other. In this problem domain, data is divided into two main groups or classes, the majority class (negative class) and the minority class (positive class). Instances in the majority class significantly outnumber the instances in the minority class.

Machine learning algorithms have many real-world applications such as fault detection[8], email spam detection [9], cancer predictions [10],[11], credit card fraud detection[12],[13],[14],[15], intrusion detection [16], among others. However, datasets from these real-world applications are usually imbalanced, hence most machine learning algorithms produce undesirable results subject

to these datasets. [17], [18]. Consequently, many classification algorithms have high accuracy for predicting the majority class while having a relatively low accuracy for predicting the minority class[19].

However, the minority class whose instances are far limited in number compared with the majority class is more attractive to the majority class [19]. Thus, it is, therefore, of research interest to have a balanced prediction accuracy for both the minority and the majority class instances.

Over the past decades, researchers have proposed methods to tackle this problem at both the algorithmic level and at the data level[20]. The data level methods such as ROS [21], RUS [22], SMOTE [23], SNOCC [21] involve manipulating the datasets to reduce the degree of imbalance whilst the algorithm level methods such us Cost-Sensitive learning and Recognition based learning [20] involves the tweaking of the classification algorithms[17].

Motivated by the success of the data level methods, this research proposal presents a new Hybrid Cluster-Based Sampling Technique (**HCBST)** that has the capability of improving the performance of classification algorithms when learning from imbalanced datasets.

1.1 Problem Statement

Machine learning classification algorithms often assume the datasets to have a uniform distribution between different classes [24]. However, many datasets of real-world problems such as cancer prediction [25] and facial recognition [26] have imbalance class distribution. Researchers have proposed several techniques to solve this problem at both the data level and the algorithmic level [20]. At the data level, recent sampling techniques such as SNOCC and CUST have been shown from experiment to improve the performance of the classification algorithms when compared to

other standard sampling techniques [21], [17]. CUST uses k-means clustering to under-sample majority instances removing duplicate/repeated instances. However, it does take into consideration, the presence of minority class instances, hence overlapping classes might still be persisted after the sampling process. SNOCC uses an oversampling technique based on SMOTE to oversample minority class instances such that the new synthetic samples are generated in the distribution of the minority samples using a convex combination, unlike SMOTE where the new samples are generated on the line segmented between the seed samples[21]. Although SNOCC, generates samples that are more representative of the minority class, it does not consider the presence of majority class instances. Hence the problem of overlap in classes is not alleviated.

The combination class imbalance and class overlap lead to further deterioration of the performance of standard machine learning classifiers [27].

This research, therefore, proposes a new Hybrid Cluster-Based Undersampling Technique (HCBST) that could improve the performance of classification algorithms using both oversampling and undersampling while accounting for class overlap.

1.2 Objectives of the Study

This study aims to design, implement, and examine a Hybrid Sampling Technique that can improve the overall performance of supervised learning algorithms.

1.2.1 Specific Objectives

This research sought to achieve the following specific objectives:

- To analyze instances of imbalanced datasets and formulate a novel mechanism for oversampling minority instances and undersampling majority class instances.
- To implement the proposed mechanism/algorithm in Python.
- To evaluate the efficiency of the proposed algorithm by comparing its performance to existing sampling techniques using eight (8) different classification algorithms.
- To design and implement a web portal for easy application of the proposed algorithm to datasets.

1.3 Justification of the Study

Class imbalance problem is a common problem in many real-world application domains such as intrusion detection, credit card fraud detection, cancer detection, among others. Consider cancer datasets such as the Mammography dataset[25] used in cancer detection application domain, which contains 10,923 "Healthy" samples (majority instances) and 260 "Cancerous" samples (minority instances). Approximately 97.7% of the dataset comprises of "Healthy" samples, and only 2.3% are "Cancerous" samples. In other words, for every "Cancerous" sample, there are approximately 42 "Healthy" samples. It is expected to be more costly in classifying cancerous samples as healthy compared to classifying healthy samples as cancerous [28]. However, for this kind of dataset, standard classifiers will most likely have imbalance classification accuracy such that the majority class instances will have arguably extremely high classification accuracy whereas minority class instances will have relatively low classification performance. Data sampling approaches are among the commonly used techniques in tackling class imbalance problem. Data quality issues are either not completely alleviated or ignored altogether. Inspired by the performance of CUST and SNOCC for classification tasks, this research project is aimed at designing an efficient algorithm

that improves the performance of classification algorithm considering other data quality issues that were not alleviated by these techniques.

Thus, this research contributes to the scientific community by proposing and implementing a hybrid sampling method which improves the overall performance of traditional machine learning algorithms.

1.4 Thesis Outline

The thesis is organized as follows: Chapter 2 discusses relevant scholarly articles concerning the scope of the work. Chapter 3 provides some key methodological considerations needed to correctly implement, experiment, test and validate the proposed technique against related benchmark algorithms. Chapter 4 demonstrates how the theoretical framework of the proposed technique was implemented and tested. Chapter 5 discusses the results obtained subject to some statistical analysis. Finally, Chapter 6 provides a summary to the thesis and highlights its contribution to knowledge, observations and recommendations.

CHAPTER 2

LITERATURE REVIEW

2.0 Introduction

This chapter presents a review of related work on class imbalance and data sampling techniques used to tackle this problem, including the typical performance metrics used as well as some classification algorithms. The Chapter is organized into four (4) sections, general overview of Class Imbalance, Data Sampling, Classification Algorithms and Methodological Issues. The Data Sampling section reviews some sampling techniques whilst the Classification algorithms section reviews some well-known classification algorithms. Finally, the Methodological Issues section highlights the common pitfall in judging the performance of classification algorithms and the appropriate remedies.

2.1 Class Imbalance

Many real-world datasets have imbalance class distribution that consequently overwhelms traditional machine learning algorithms as a result of the imbalanced nature of these datasets [17],[18]. Consequently, many classification algorithms have high accuracy for predicting the majority class while having a relatively low accuracy for predicting the minority class [19].

However, the minority class whose instances are far limited in number compared to the majority class is more interesting to the majority class [19]. Thus, it is therefore of research interest to have a balanced prediction accuracy for both the minority and the majority class instances

Several methods from the literature have been proposed to tackle this problem at both the algorithmic level and at the data level [20]. The algorithmic level includes methods such as cost-

sensitive learning and recognition based learning [20],[29],[30]. At the data level, the methods include re-sampling and multi-classifier committee.

2.2 Data Sampling

The data sampling approach is among the preferred methods in approaching this problem mainly because it involves manipulating the data itself instead of tweaking the classification algorithm, which requires a considerable amount of technical expertise [17].

2.2.1 Random Resampling Techniques

The sampling method can be categorized into two main approaches, oversampling and undersampling. Oversampling increases the number of minority class samples or instances, whilst under-sampling decreases the number of majority class samples. Thus, reducing the degree of imbalance in the training data.

Among the various sampling techniques, random sampling is one of the easiest to implement due to its simplicity. Random oversampling (ROS) involves randomly replicating minority class instances [31] while random undersampling (RUS) involves randomly removing majority class instances [32],[33],[34]. Though both sampling methods are relatively fast and easy to implement, they have some drawbacks. RUS may lead to loss of information since some majority class instances are removed [22]. This problem may worsen if the dataset is small. In ROS, no new information is added by randomly replicating minority class instances, hence may lead to overfitting [35].

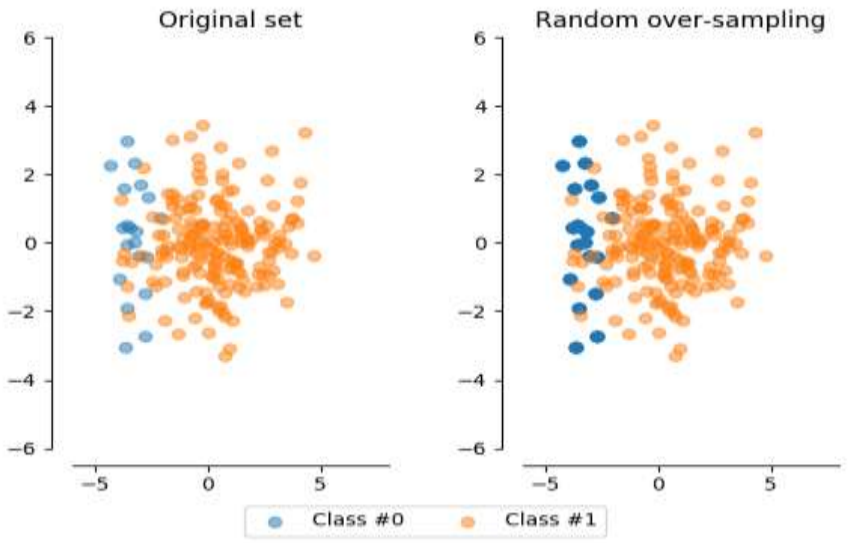


Figure 2.1 Random Oversampling

2.2.2 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is an oversampling technique that attempts to alleviate the problems of overfitting that is introduced by Random Oversampling [23]. It generates synthetic samples using the k-nearest neighbour rule such that the new synthetic samples fall on the line segment between the seed samples. Studies have shown that SMOTE provides better performance than Random Oversampling[23], [30]. Figure 2.2 shows how SMOTE generates minority samples.

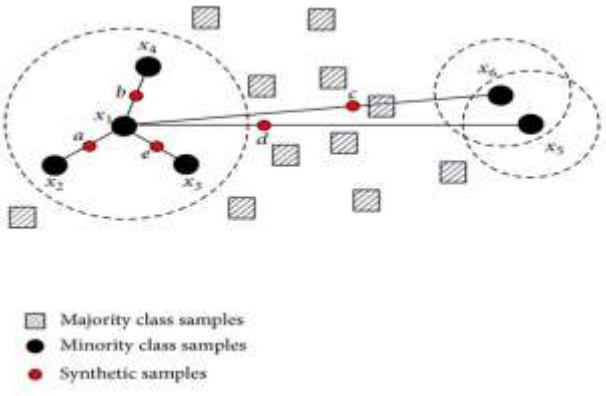


Figure 2.2. Synthetic Minority Oversampling Technique (credit: Chawla et al.)[36]

However, SMOTE can lead to class overlapping since it completely ignores the presence of Majority samples and is affected by factors such as sparsity and high dimensionality of the dataset [37],[38],[39],[40]. An extension of SMOTE, called the SMOTEBoost, has also been proposed in [41] where SMOTE is coupled with adaptive boosting algorithms to update the weights of minority class data.

2.2.3 Adaptive Synthetic Sampling (ADASYN)

ADASYN is an oversampling technique derived from SMOTE that uses a density distribution of the minority samples according to their level of difficulty to determine the number of minority instances to synthesize[42]. ADASYN assumes minority samples with more majority class k-nearest neighbours are more challenging to learn. The number of samples g_i to generate for a given minority instance x_i is given by;

$$g_i = \frac{\Delta_i}{\sum_{i=1}^{m_s} \Delta_i} \beta(m_l - m_s);\ 0<\beta \leq 1 \qquad (2.1)$$

Where β is the parameter that specifies the ratio over class balance after the sampling process, m_l is the number of majority samples m_s is the number of minority samples and Δ_i is the number samples in the k-nearest neighbours of x_i that are majority class instances.

As a result, more synthetic samples are generated for minority samples that are harder to learn. Although ADASYN improves classification performance in some cases by adaptively shifting the

decision bounding towards examples that are harder to learn, it may increase the number of class overlaps in the dataset.

2.2.4 under-Sampling Based on Clustering (SBC)

SBC is an undersampling technique that uses clustering to select majority class samples according to the ratio of the number of majority class to the number of the minority class in each cluster [20]. It is based on the idea that clusters that have more majority class instances behave like majority samples and clusters that have more minority samples behave like minority samples. Thus, more majority classes are selected from clusters with a greater ratio of majority to minority samples, and fewer majority samples are selected from clusters with a smaller ratio of majority to minority samples. However, the baseline algorithm of SBC behaves much more like random undersampling by randomly selecting majority samples after determining the number of majority samples to select from each cluster. Thus, it may lead to the prominent problem of loss of information in random undersampling.

2.2.5 One-Sided Selection (OSS)

OSS is an undersampling technique that is aimed at removing noisy/borderline samples from the majority samples[43]. It first trains all minority samples and a random selection of a majority sample using the 1-NN rule and then used to classify the remaining majority samples. All misclassified samples are added to the 1-NN trained dataset, and instances that participate in Tomek links are removed. However, OSS does not consider the presence of outliers in the dataset.

2.2.6 Sigma Nearest Oversampling based on Convex Combination (SNOCC)

SNOCC uses a technique to generate synthetic samples within the region bounded by the line segments between the seed samples. It takes an input of seed samples from the minority class, gets

the k-nearest neighbours distances, calculates their mean m_i and then computes *sigma* as the average of m_i plus the standard deviation.

$$sigma = \frac{1}{S}\left(\sum_{i=1}^{S} m_i\right) + \sigma \tag{2.2}$$

, where S is the number of seed samples and σ is the standard deviation.

The algorithm then randomly selects a seed sample s_1 , and two nearest neighbours s_2 and s_2 called sigma nearest neighbours such that their distances are less than *sigma* [21]. It then generates a three-dimensional vector $\alpha(\alpha_1, \alpha_2, \alpha_3)$ such that $\alpha_1 + \alpha_2 + \alpha_3 = 1$ [21]. Now the new synthetic sample is finally generated using the equation $s = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3$ [21]. The process is repeated until the required number of minority samples are obtained. Figure 2.3 shows how samples are generated in the distribution space of the minority samples.

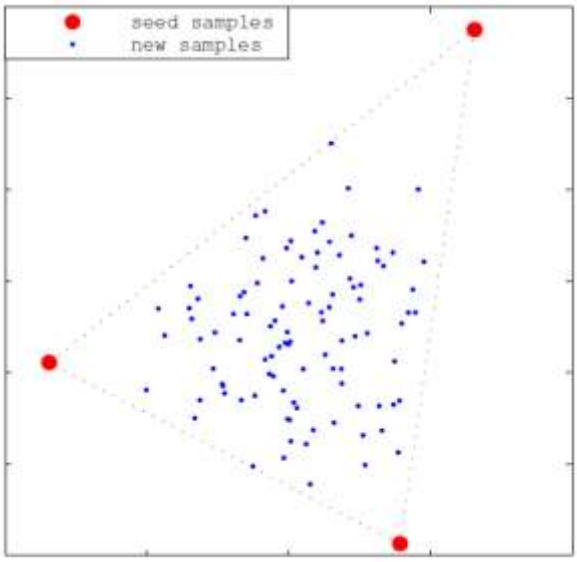


Figure 2.3 Generating Samples using SNOCC (credit: Zheng et. al[21]*)*

This technique has been shown experimentally by [21] to generate synthetic samples that are more representative of the original minority samples than SMOTE. However, this does not consider the presence of majority samples within the distribution space of the seed samples used to generate the synthetic samples.

2.2.7 Cluster Undersampling Technique (CUST)

Cluster Undersampling Technique (CUST) is an undersampling technique that is much similar to Random Undersampling but performs additional tasks of removing noisy, inconsistent, and redundant samples from the dataset[17]. CUST first removes inconsistent samples using a technique derived from Tomek links. The figure below shows how Tomek links are used in undersampling majority instances.

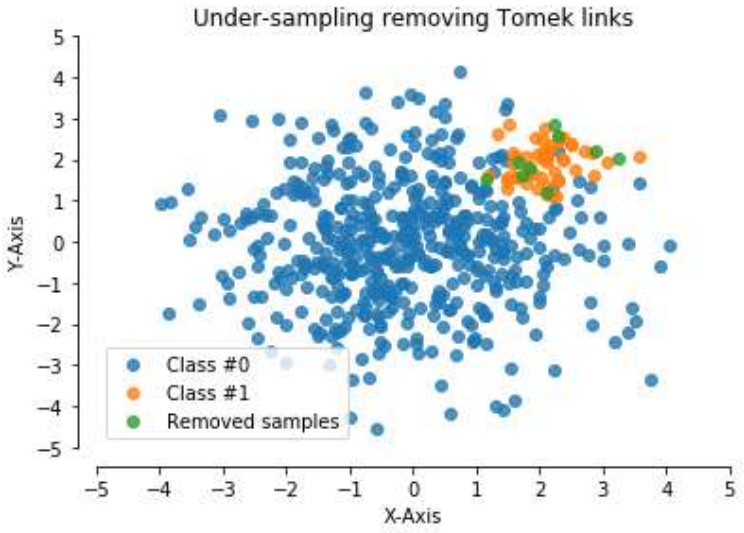


Figure 2.4. Undersampling using Tomek Links.

Then clusters the majority samples into k-clusters. For each cluster, it selects majority samples according to a specified ratio discarding duplicates during the process. CUST assumes that clustering majority class instances will lead to outliers forming small clusters or single clusters away from the main clusters[17].

CUST has been shown experimentally to significantly improve the performance of some traditional machine learning algorithms[17]. However, like SNOCC, CUST does not consider the local proximity of the opposite class instances. This may result in selecting majority instances that overlap with minority instances.

2.3 Classification Algorithms

2.3.1 K Nearest Neighbour (KNN)

KNN is a non-parametric estimator used in pattern recognition for classification and regression[44]. KNN predicts the class of an example by finding the majority vote of K closest neighbours in the training set [45]. The algorithm of KNN is summarized below;

Let X be a set of n pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ of the training set S
where each x_i lies in the feature space of S.
and y_i represents the category to which x_i belongs to
For a new example x
Compute the distances
$d(x, x_1), d(x, x_i), ..., d(x, x_n)$ in the metric space of D
Find k elements in S that form minimum distances in D such that
$Q = \{q_1, q_2, ..., q_k\}$
where $q_i = min(D \setminus Q);\ 1 < i \leq k,\ q_i = s(x_i, y_i)$
and $q_1 = min(D);\ q_1 = s(x_1, y_1)$
Determine the number of occurance of each category in Q

$$O(y_i) = \sum_{j=1}^{k} [y_j = y_i]$$

The category of x is determined by
$y = y_i$ if $O(y_i) = \max(O, O(y_i))$

It's arguably one of the simplest classification algorithms due to its intuitive non-parametric procedure. However, the value of k can also affect the output of the classifier; hence, must be carefully chosen [46].

2.3.2 Support Vector Machines (SVM)

SVM classifier builds a training model that assigns new unseen examples to categories by first representing the training examples in a feature space and mapping them such that examples in different categories are separated by a margin or gap as wide as possible[47].

The optimal hyperplane is determined by solving the constrained optimization problem

$$min(\frac{1}{2}w^T w) \tag{2.3}$$

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b = \sum_{i=1}^{n} x_i w_i + b = 0 \tag{2.4}$$

Subject to $y_n(w^T x_n + b) \geq 1$ for $n = 1,2,3,...,N, w \in \mathbb{R}^d, b \in \mathbb{R}$ where **w** is a vector of weights and b is the bias.

2.3.3 Decision Tree (CART)

Classification and Regression Tree (CART) is a decision tree algorithm that is capable of processing both continuous and categorical attributes[48]. Unlike C4.5 and C5.0 decision tree algorithms that use the concept of information gain as performance measure using the training data, CART mainly uses the concept of Gini Impurity as performance measure using an independent test data though later versions added the entropy rule [48]. The Gini impurity for a binary target t is given by;

$$G(t) = 1 - p(t)^2 - (1 - p(t))^2 \tag{2.5}$$

Where $p(t)$ is the relative frequency of class 1 in the node.

2.3.4 Random Forest

Random Forest is an ensemble method that combines multiple decision tree classifiers to fit sub-samples of the training data and uses the mean prediction or mode of the classes of each of the trees as output[49][50]. Decision trees tend to overfit or have a high variance when grown very deep. Random Forest is aimed at reducing this variance by fitting different deep decision trees on different parts of the training data and then finding the mean(for regression) or mode (classification)[50].

2.3.5 Multilayer Perceptron

The multilayer perceptron is an artificial neural network that consists of at least three layers namely, the input layer, the hidden layer and the output layer connected in a feed-forward manner such that each neuron in a layer has a direct connection to the neurons in the subsequent layer[51].

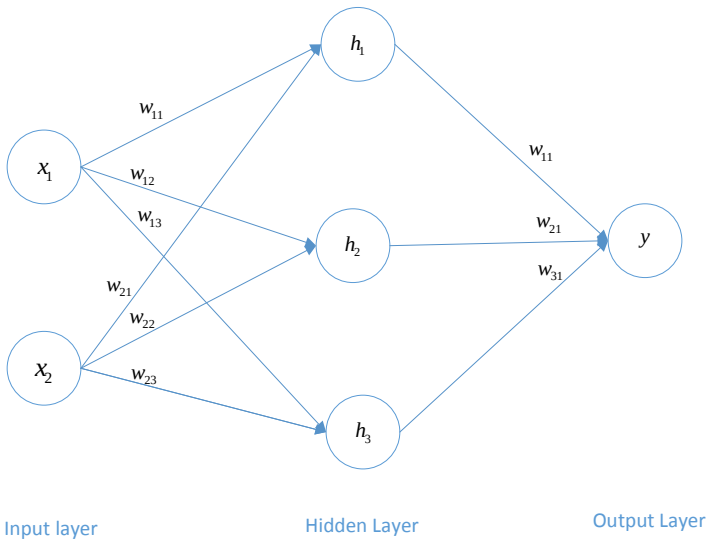


Figure 2.5 Multilayer Perceptron

Except for the input layer, each neuron is activated by a non-linear activation function[51]. Each connection to each neuron is represented by a weight w_{ij} . Learning occurs by updating the weights

after each forward pass [52]. Consider the error at the output node given by $e_i(n) = t_i(n) - y_i(n)$ where $t_i(n)$ is the target value and $y_i(n)$ is the output value for the given input (x_1, x_2). The weights are adjusted to minimize the overall error given by;

$$\varepsilon(n) = \frac{1}{2}\sum_i e_i^2(n) \tag{2.6}$$

2.3.6 Adaboost

Adaptive Boosting (Adaboost) is an ensemble method that combines several weak learners to improve overall performance[34], [53]. High dimensional data does affect not only the execution time of classification algorithms but also the accuracy[54]. Adaboost attempts to address this issue by selecting only the relevant features known to improve the predictive power, thus potentially improving the execution time. The boost classifier is represented by;

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \tag{2.7}$$

where $f_t(x)$ is a weak learner

2.4 Performance Metrics

In a binary classification problem, the four main categories, a predicted sample by a classifier will belong to[55]:

- True Positives (TP)
- False Positives (FP)
- True Negatives (TN)
- False Negatives (FN)

These four categories are usually used to form a confusion matrix[56] as shown in Table 2.1

Table 2.1 Confusion Matrix

	Actual Condition Positive	Actual Condition Negative
Predicted Condition Positive	TP	FP
Predicted Condition Negative	FN	TN

True Positives refers to the number of positive instances that are classified correctly as positive whilst True Negatives refers to the number of negative instances correctly classified as negative. False Positives represents the number of negative instances incorrectly classified as positive whilst False Negatives denotes the number of positive instances incorrectly classified as negatives.

Among the commonly used performance metrics are the accuracy and error rate. The accuracy of classifier is determined by the number of instances that are correctly classified, and the error rate is the number of instances that are incorrectly classified.

The overall accuracy A, of the classifier, is given by:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

The error rate E_r, is given by:

$$E_r = \frac{FP + FN}{TP + TN + FP + FN} \tag{2.9}$$

However, accuracy can be misleading for datasets that are highly imbalanced as this metric favours majority class [57]. For instance, in a dataset where the number of majority instances vastly outnumber the number of minority instances by a ratio of 99:1, the classifier is likely to classify all the majority instances correctly and misclassify the only one available minority instance. Hence there will 99 true negatives, 0 false negatives, 1 false-positive and 0 true positive resulting in an accuracy of 99% which is very misleading since the accurate prediction of the positive classes are usually more desirable.

$$A = \frac{0+99}{0+99+1+0} = \frac{99}{100} = 0.99$$

To avoid this paradox, it is recommended that performance measures based on class metrics are used[56]. These metrics are listed below:

True Positive Rate (**TPR**) also called **sensitivity** or **recall** is the proportion of the number of positive instances correctly classified as positive to the total number of positive instances. It is given by the equation below.

$$TPR = \frac{TP}{TP + FN} \tag{2.10}$$

True Negative Rate (**TNR**) also called **specificity** is the proportion of the number of negative instances correctly classified as negative to the total number of negative instances. It is given by the equation below.

$$TNR = \frac{TN}{TN + FP} \tag{2.11}$$

False Positive Rate (**FPR**) also called **Fall-out** is the proportion of the number of negative instances incorrectly classified as positive to the total number of negative instances. It is given by the equation below.

$$FPR = \frac{FP}{TN + FP} = 1 - TNR \tag{2.12}$$

False Negative Rate (**FNR**) also called **miss rate** is the proportion of the number of positive instances incorrectly classified as negative to the total number of positive instances. It is given by the equation below.

$$FNR = \frac{FN}{TP + FN} = 1 - TPR \tag{2.13}$$

The Receiver Operating Characteristic (ROC) curve is a graphical plot of the Sensitivity against the Fallout at different classification thresholds. ROC curves are commonly used to interpret the effects of several decision thresholds on the performance of a model[58]. A common metric used to represent a single scalar value of the ROC curve is Area under the ROC Curve (AUC)[59] as shown in equation 2.14. AUC values range from 0 to 1.0. A value of 1.0 indicates perfect classification while a value 0.5 indicates a performance of similar to an unbiased random selection and a value of 0 corresponds to the worst performance (all positive instances incorrectly classified).

$$AUC = \frac{1 + TPR - FPR}{2} \tag{2.14}$$

G-Mean is a metric that measures the average between the classification performance of both Sensitivity and Specificity[57].

$$G-Mean=\sqrt{TPR\times TNR} \tag{2.15}$$

Mathews Correlation Coefficient (MCC) measures the correlation coefficient between the observations and the predictions from a classifier[57]. It takes into account all aspects of the confusion matrix and is generally regarded as one of the metrics that can represent the confusion matrix with a single value[60][61].

$$MCC=\frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{2.16}$$

2.5 Summary

This study uniquely contributes to the scientific community by implementing a hybrid sampling technique which improves the overall performance of the classification algorithms discussed in this chapter. This sampling technique is based on CUST and SNOCC that takes into consideration the presence of overlap regions in the imbalanced dataset. Furthermore, the performances of all sampling techniques discussed in this chapter are compared to that of the proposed technique by evaluating the performance of the classification algorithms after the sampling methods have been applied on a variety of datasets.

CHAPTER 3

METHODOLOGY

3.0 Introduction

In this chapter, the research strategy, the research method, the research approach, the methods of data collection, the selection of the sample, the research process, the type of data analysis, the ethical considerations and the research limitations of the project are outlined (Soares, 2003). This Chapter is organized into two (2) main sections, Experimental Setup and Design of HCBST. The Experimental section highlights the data sources and approach for the validation of the performance of the various sampling techniques used in this study. The next section highlights the development of the theoretical framework of the proposed technique.

3.1 Experimental Setup

3.1.1 Datasets

Eleven datasets were used in the experimental setup of which six were obtained from National Aeronautics and Space Administration (NASA) Metric Data Program (MDP) [62] and five from University of California Irvine Repository [63]. Table 3.1 summarizes the datasets used from both repositories.

The NASA MDP datasets consist of data from various software projects undertaken by NASA, which was obtained from a backup of the original data by Tantithamthavorn [64]. In 2011 Gray et al. [65] thoroughly documented a data cleansing process involving all the 13 NASA Software defect datasets which resulted in each of the datasets having 6 to 90 percent of their original data. The backup datasets used in this research constitute a cleaned version of the datasets by Shepperd et al. [66]. The main objective of Shepperd et al. [66] was to investigate the depth to which

published research analysis based on the NASA Software Defect Datasets are comparatively insightful[66]. However, Petrić et al. [67] identified additional rules for removing problematic data which were not identified by [66]. Applying these rules led to the observation that JM1 and MC2 datasets were the most problematic among the 13 NASA Software defects datasets [67].

The UCI Repository is a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. For this study, the multi-class datasets sourced from the UCI Repository were modified into binary classification problems. Table 3.1 shows a summary of the datasets their class distributions.

NASA Software Defects and UCI datasets were used as data samples for this study because they are publicly available and are extensively used by other researchers in undertaking studies related to this work [67].

Table 3.1 Summary of Datasets

Dataset		**Num. of Attributes**	**Total Instances**	**Positive instances**	**Negative instances**	**Imbalance ratio**
NASA Datasets	CM1	38	344	42	302	7.19
	KC3	40	200	36	164	4.56
	MC2	40	127	44	83	1.89
	MW1	38	264	27	237	8.78
	PC1	38	759	61	698	11.44
	PC2	37	1585	16	1569	98.06
UCI Datasets	Abalone9v18	9	731	42	689	16.40
	Abalone19	9	4177	32	4145	129.53

	Ecoli4	8	336	20	316	15.80
	Glass2	10	214	17	197	11.59
	Yeast2v8	9	264	20	244	12.20

Table 3.2 Description of NASA MDP Datasets

Dataset	Language	Project
CM1	C	Is a NASA spacecraft instrument for data collection and processing
KC3	Java	is an application that collects, processes and delivers satellite metadata
MC2	C++	Is a video guidance system
MW1	C	Is an application from a zero-gravity combustion experiment
PC1	C	is a flight software for an earth-orbiting satellite
PC2	C	Is from a dynamic simulator for attitude control systems

3.1.2 Sampling Methods Used

To assess the effectiveness and efficiency of the proposed technique, eight (8) other data sampling techniques are considered in this study, the proposed sampling technique. The methods include Random Undersampling, Random Oversampling, Synthetic Minority Oversampling Technique, Adaptive Synthetic Sampling, under-Sampling based on Clustering, Cluster Undersampling Technique, One-Sided Selection, and CLUSter-based hybrid sampling approach. Cluster Undersampling Technique, under-Sampling based on Clustering, and CLUSter-based hybrid

sampling approach are implemented in Python for his research. Scikit-learn [68] implementation of the remaining standard sampling techniques is used.

3.1.2.1 Random Sampling

Random Undersampling and Random Oversampling are the main random sampling techniques considered for this study. The imbalance ratio sampling parameters used for random undersampling are; 0.05, 0.10, 0.25, 0.50, 0.75, and 0.90. If a parameter of 0.75 is specified, then 25% of majority class instances are removed, and 75% is maintained. These sampling parameters were used in the previous study[17]. The undersampling function used in ***scikit-learn*** requires the experimenter to specify the final ratio of the number of minority samples to the number of majority samples after the undersampling process and not the percentage of majority samples to retain or discard. To achieve the desired parameter specification, a mapping function is used to map the desired parameter specification onto the parameter space of scikit-learn's random undersampling function.

$$f(x) = \frac{1}{x} \times \frac{|Min|}{|Maj|} \tag{3.1}$$

$$g(f(x)) = \min\left(1, f(x)\right) \tag{3.2}$$

Where $|Min|$ and $|Maj|$ are the number of instances in the minority and majority instances respectively and $x \in X\{0.05,\ 0.10,\ 0.25,\ 0.50,\ 0.75, 0.9\}$.

The imbalance ratio sampling parameters used for random oversampling are; 0.5, 1, 2, 3, 5, 7.5, and 10. If a parameter of 5 is specified, the number of minority class instances in the new training set is increased by 5 folds. Also, these sampling parameters were used in the previous study[17]. The oversampling function used in scikit-learn requires the experimenter to specify the final ratio of the number of minority samples to the number of majority samples after the oversampling process and not the percentage of majority samples to retain or discard. To achieve the desired parameter specification, a mapping function is used to map the desired parameter specification onto the parameter space of ***scikit-learn's*** random oversampling function.

$$f(x) = (1+x) \times \frac{|Min|}{|Maj|} \tag{3.3}$$

$$g(f(x)) = \min\left(1, f(x)\right)$$

Where $|Min|$ and $|Maj|$ are the number of instances in the minority and majority instances respectively and $x \in X\{0.5,\ 1,\ 2,\ 3,\ 5,\ 7.5,10\}$.

The performance of the classifiers is evaluated using the range of parameters specified, and the highest classification performance is used as the performance of the classification algorithm under consideration.

3.1.2.2 SMOTE

The sampling parameters used for random oversampling are also used for SMOTE, and a default number of neighbours of 5 with a random seed of 0 is also used in all experiments. The performance of the classifiers using different performance metrics is evaluated using the range of parameters specified, and the highest classification performance is used as the performance of the classification algorithm for the metric under consideration.

3.1.2.3 OSS

OSS requires only the number of neighbours and a random seed to be specified. The default parameter of 1NN and the random seed of 0 is used for this study.

3.1.2.4 ADASYN

ADASYN requires similar parameters as SMOTE; hence, the same parameters used for SMOTE are also utilized for ADASYN.

3.1.2.5 Under-Sampling Based on Clustering (SBC)

SBC requires the number of clusters and the ratio of the number majority instances to minority instances to be specified. The parameters used for the number of clusters are 3, 5, 7, 10, 15, 20, 15, 30, 35 and 50 and the parameters used for the ratio of the number of majority instances to minority instances are 1, 1.5, 1.75, 2, 3, 5, 10, 15, 20, 25, 50. If a parameter of 1.5 is selected, it implies that the final training dataset will have 50% more majority samples than that minority samples. The performance of the classifiers using different performance metrics is evaluated using the range of parameters specified, and the highest classification performance is used as the performance of the classification algorithm for the metric under consideration.

3.1.2.6 CLUS

CLUS requires only the number of neighbours to be specified. The default parameter of 5 nearest neighbours is used for this study.

3.1.2.7 Cluster Undersampling Technique (CUST)

CUST requires the number of clusters k and the imbalance ratio r to be specified. The parameters for the number of clusters used for CUST in this study were 3, 5, 7, 10, 15, 20, 15, 30, 35 and 50 and values of r used for the experiment were 1, 1.25, 1.5, 1.75, 2.0, 2.5, 3, 10, 25 and 50. The

performance of the classifiers using different performance metrics is evaluated using the range of parameters specified, and the highest classification performance is used as the performance of the classification algorithm for the metric under consideration

3.1.3 Classification Algorithms

Eight classification algorithms, namely KNN, SVM, Decision Tree, Random Forest, Neural Network, AdaBoost, Naïve Bayes, and Quadratic Discriminant Analysis, were used for this study. All the classifiers used in this study are implementations from scikit-learn python library. Unless otherwise stated, the default parameters for all classifiers were used. It is worth noting that the Neural network classifier referred to in this context is the Multilayer Perceptron (MLP) classifier.

3.1.4 Tools Used

In this study, the tools used include a Python IDE running on 64-bit Windows 10 and scikit-learn, a machine learning module written in python with fully integrated a wide range of machine learning algorithms [68]. Python is a relatively simple interpretive language that allows writing more readable and typically much shorter code than equivalent C or C++ programs[69]. The classification algorithms used from scikit learn include K-Nearest Neighbours, Support Vector Machines, Random Forest, Multilayer Perceptron, Adaboost, Naïve Bayes, and Quadratic Discriminant Analysis. The default parameters for these classifiers were used in this study.

3.1.5 System Specifications

The system specification of the platform on which the proposed technique was developed is outlined below.

Operating System	Windows 10
System Architecture	64-bit

RAM	32 Gigabytes
CPU	Intel Core i7 7700 HQ
Number of Cores	4
Number of Threads	8
Cache	6 Megabytes
Base Frequency	2.8 Gigahertz
Turbo Frequency	3.8 Gigahertz
Storage	256 Gigabytes M.2 SATA III SSD + 2 Terabytes SATA HDD

3.1.6 Performance Metrics

The main performance measures used for evaluating the performance of the classifiers include Area Under the receiver-operating-characteristic Curve (AUC), Geometric Mean (G-Mean) and Matthews Correlation Coefficient (MCC). The functions for calculation these performance metrics are built into the classifiers provided by ***scikit-learn***.

3.1.7 Experimental Method

Eight sampling techniques are used to sample the training data before training the classification models. To get accurate out of sample performance evaluation, each dataset is first split into training and testing using stratification and a random seed. Stratification allows the data to be divided in such a way that the resulting datasets each have equal representation of the original dataset. The random seed provides for the easy reproducibility of the results.

Additionally, stratified tenfold cross-validation is performed on the training dataset. For each validation done on the training dataset, the validation performance is estimated using the held-out data and results recorded as validation performance for further analysis and the resulting model is used to evaluate the test dataset and the results recorded as testing performance. After the tenfold cross-validation is completed, the results are averaged and recorded for both validation and testing performances.

Furthermore, the whole process is repeated ten times, each time changing the random seed value to reduce the biases that might have been introduced the during the stratification process in the splitting of the training and testing data or the cross-validation process. For each dataset, the validation and testing performances are averaged to give the overall performance of the classifier under consideration. The overall testing performance is used as the performance of the model.

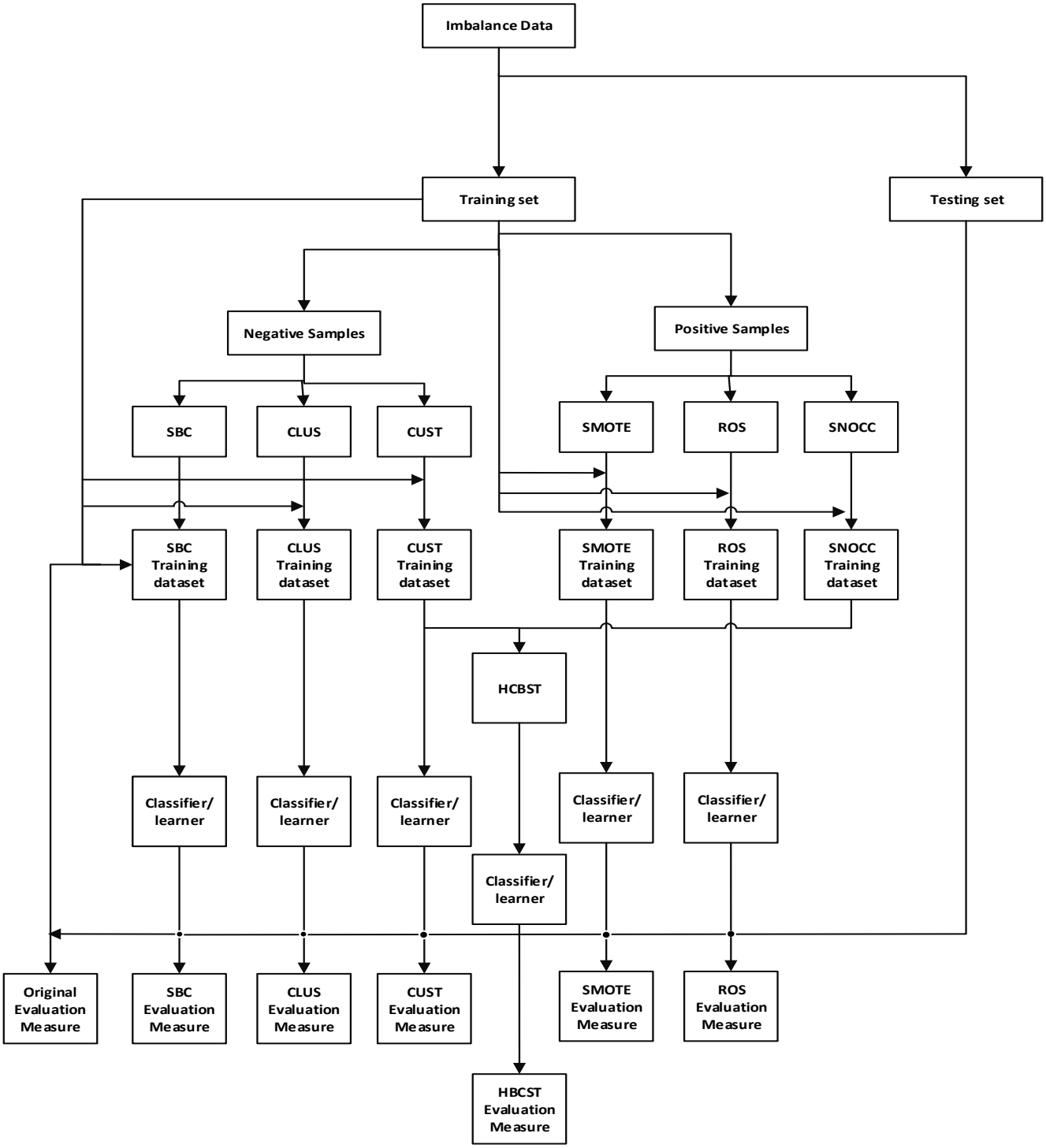


Figure 3.1 Experimental Framework

3.2 Design of HCBST

Hybrid Cluster-Based Sampling Technique aims to improve the overall performance of well-known machine learning algorithms while improving the computational time compared to other sampling techniques employing k-means algorithm. HCBST algorithm occurs in two stages; the first stage is the oversampling stage where synthetic minority class instances generated using SMOTE[23]. The second stage is the undersampling stage, which is based on the notion of using clustering to identify outliers in data [70][71].

The design of HCBST allows for the flexibility of using different sampling parameters for both the undersampling and oversampling process. Thus, the sampling parameters are estimated before the sampling process. Figure 3.2 shows a summary of the design of HCBST.

The oversampling process uses a technique derived from SNOCC[21] to oversample minority class instances. Contrary to SMOTE, where the synthetic samples lie on the line segment between the seed samples, SNOCC uses a technique to generate synthetic samples within the region bounded by the line segments between the seed samples. SNOCC takes an input of seed samples from the minority class, gets the k-nearest neighbours distances, calculates their mean m_i and then computes *sigma* as the average of m_i plus the standard deviation from the equation (2.2).

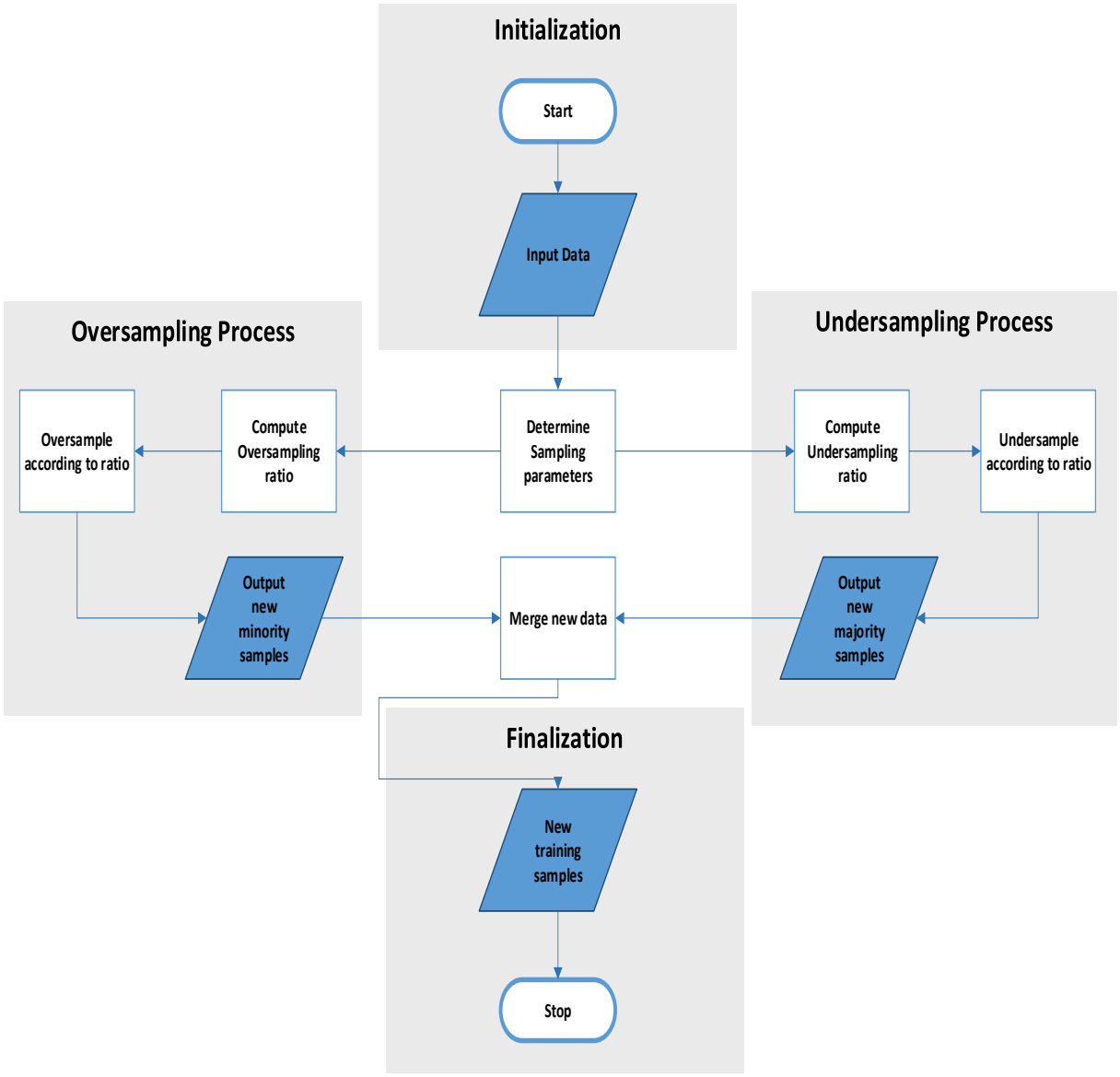


Figure 3.2 Summary of the design of HCBST

The algorithm then randomly selects a seed sample s_1 , and two nearest neighbours s_2 and s_2 called sigma nearest neighbours such that their distances are less than *sigma* [21]. It then generates a three-dimensional vector $\alpha(\alpha_1, \alpha_2, \alpha_3)$ such that

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \qquad (3.4)$$

Now the new synthetic sample is finally generated using the equation

$$s = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3 \tag{3.5}$$

The process is repeated until the required number of minority samples are obtained. This technique has been shown experimentally by [21] to generate synthetic samples that are more representative of the original minority samples than SMOTE. However, this does not consider the presence of majority samples within the distribution space of the seed samples used to generate the synthetic samples. To address this issue, HCBST uses a selection criterion that ignores synthetic samples with a higher probability of overlapping with majority samples.

The method in HCBST that performs the oversampling process requires the experimenter to specify the parameters r_o, b_m and o_s where r_o is the ratio of the number of samples after the oversampling to the number of original minority samples, b_m specifies whether to match the number of sampled minority instances to the number of sampled majority instances and o_s specifies whether to return only synthetic samples or both original samples and synthetic samples. The value r_o is used to determine the number of minority samples to generate N_o which is given by the equation below:

$$N_o = N_{mi}(r_o - 1) \tag{3.6}$$

$\forall\ r_o \in$ □ and $r_o > 1$

Where N_{mi} is the number of original minority samples

If r_o is set to 2, then the sampled minority instances will be approximately twice the number of the original minority samples and if r_o is set to 1 then no oversampling will take place.

After determining the number of minority samples to generate, HCBST then generates a synthetic sample according to SNOCC, for each of the sigma neighbours s_2 and s_2, computes their distance to the closest majority sample, then find the average between the computed distances as μ_s. Finally, it computes the distance of the new synthetic sample to the closest majority sample d_n. If $d_n < \mu_s$, the new synthetic sample is discarded and the process repeated using a new seed sample s_1. The process is repeated until the number of accepted synthetic samples = N_o.

The second part of the sampling process uses a technique from CUST to under-sample majority class instances. CUST first removes inconsistent samples using a technique derived from Tomek links, then clusters the majority samples into k-clusters. For each cluster, it selects majority samples according to a specified ratio discarding duplicates during the process. However, like SNOCC, CUST does not consider the local proximity of the opposite class instances. This may result in selecting majority instances that overlap with minority instances. To address this issue, like the oversampling process, HCBST uses a technique to exclude majority instances that have a high probability of overlap with minority class instances. The summary of the oversampling process is shown in Figure 3.3.

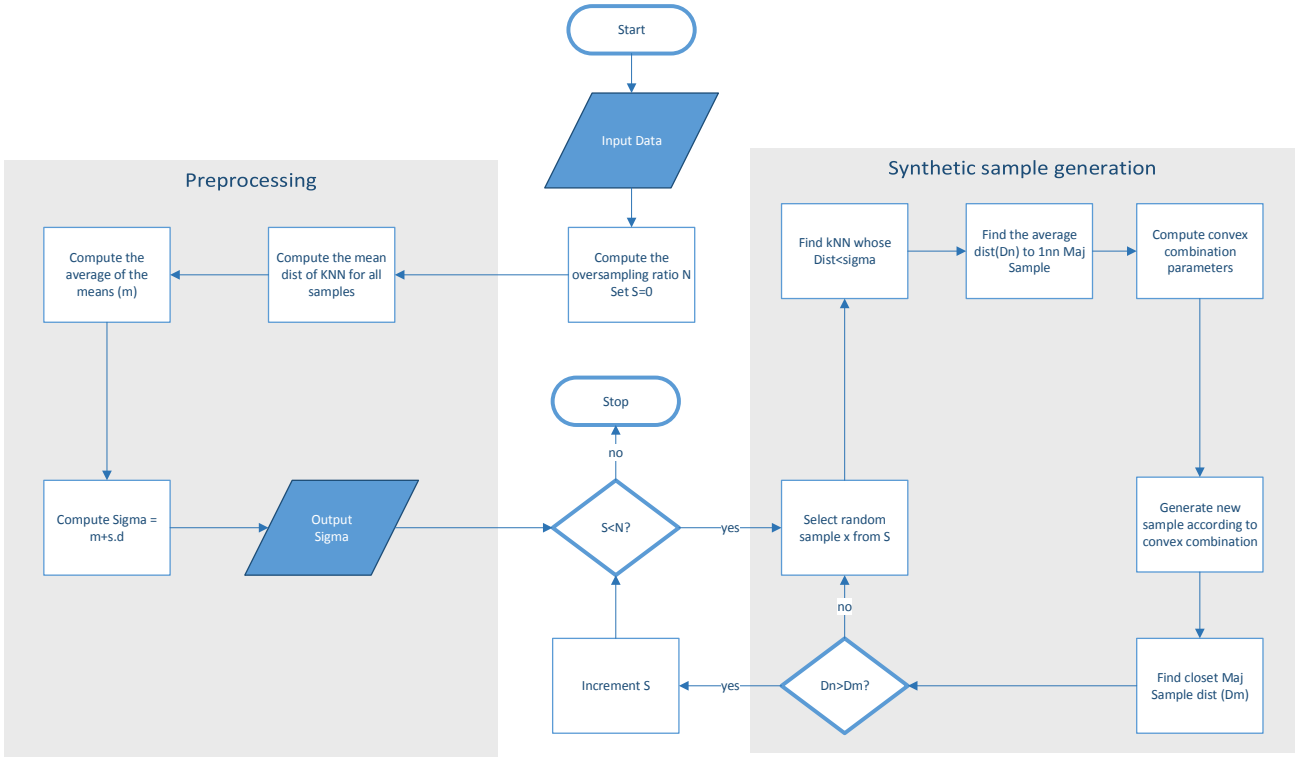


Figure 3.3 Summary of Oversampling Process

The method in HCBST that performs the undersampling process requires the experimenter to specify the parameters r_u , p_m , k_m and k_n where r_u is the ratio of the number of majority samples after the undersampling to the number of original minority samples, p_m specifies the fraction of majority samples to discard, k_m specifies the number of clusters to use from the majority samples and k_n determines the number of minority neighbours to search.

In the first part of the undersampling process, the algorithm determines the mode of undersampling, either by selecting or discarding samples from the majority samples depending on the parameters set by the experimenter. Setting the parameter p_m tells the algorithm to under-

sample by discarding majority instances. Otherwise, the parameter r_u is used to select samples from the majority instances. If p_m is set to 0.1, then 10% of majority samples are discarded, the parameter r_u is ignored.

After determining the mode of undersampling, the algorithm proceeds with clustering using the k-means algorithm to cluster the majority samples into k_m clusters. For each cluster, if p_m is not set, then the algorithm proceeds using CUST. The required number of samples to be selected from each cluster is determined by equation (3.7).

$$Maj^i = r_u \text{ x } \frac{MI}{MA} \text{ x MC}^i \text{ ;} 1 \leq \text{i} \leq \text{k}_m, \text{MA} \neq 0 \qquad (3.7)$$

Where Maj^i is the number of majority instances to select from the cluster i, MI is the total number of minority instances, MA is the total number of majority instances and MC^i is the number of majority class samples in the i^{th} cluster.

A random instance is then selected from the cluster, discarding instance if it is a replica of an already selected instance. However, unlike CUST, HCBST takes into consideration the local proximity of minority instances. When an instance S_{ci} is selected from the cluster, k_n nearest neighbours are searched, if $k_n - 1$ neighbours are minority instances, then S_{ci} is discarded. Otherwise, it is added to selected samples. The process is repeated until the required number of instances from each the cluster is obtained. The selected samples from each cluster are combined with the minority samples to form the new training set.

On the other hand, if p_m is set, then the number of majority samples to discard from each cluster is given by;

$$Maj^{i} = p_m \text{ x MA } ; 1 \leq \text{i} \leq \text{k}_m \tag{3.8}$$

A random instance S_{ci} is selected from the cluster, k_n nearest neighbours are searched, if $k_n - 1$ neighbours are minority instances, then S_{ci} is discarded. The process is repeated until the required number of instances to remove from each cluster is obtained. The remaining instances in each cluster are combined with the minority samples to form the new training set. It is, however, worth noting that distance caching is used during the k-means clustering process to improve the computation time during the undersampling process.

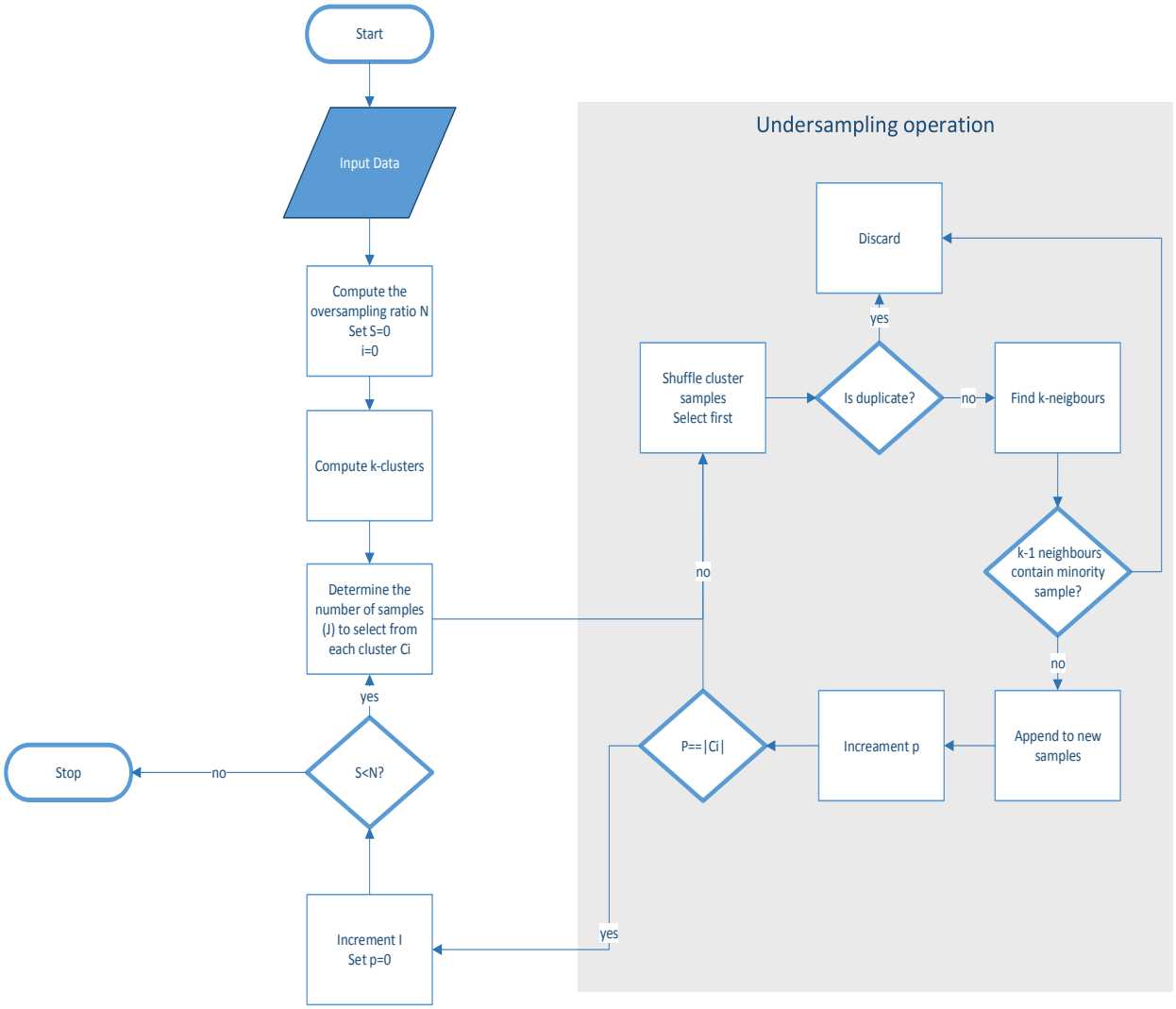


Figure 3.4 Summary of Undersampling Process

The algorithm of HCBST is summarized in the table below.

Input	_Min: minority samples _Maj: majority samples N_{mi} : number of minority samples N_{ma} : number of majority samples

	r_o : oversampling ratio
Output	S_{min}, S_{max} : Sampled minority and Majority instances respectively **Perform Oversampling**: Determine N_o the number of minority samples to generate: $N_o = N_{mi}(r_o - 1)$ Compute the mean m_i of all k-neighbours to all the instances in _Min Compute $sigma = \frac{1}{N_o}\left(\sum_{i=1}^{N_o} m_i\right) + \sigma$ new_samples = Array() For $i \leftarrow N_o$ Randomly select a sample s_1 in _Min Select two sigma nearest neighbours s_2 and s_2 Such that distance(s_1,s_2) $\leq$ sigma. Choose a 3-dimensional vector $\alpha(x, y, z)$ such that $x + y + z = 1$

Let

$c = rand(0,1)$

$x = rand(0, c)$

$y = rand(0, 1-c)$

$z = 1-(z+y)$

Generate the new sample using $s_i = xs_1 + ys_2 + zs_3$

Compute the distance of s_2 and s_2 to their closest majority sample,

as d_1 and d_2

Compute the mean of the distances d_1 and d_2 as μ_s

Compute the distance d_n from the new synthetic sample to the closest

majority sample.

If $d_n < \mu_s$

skip,

Else

push s_i to new_samples

End For

S_{min} = append(new_samples,_Min)

Perform Undersampling

Cluster majority samples into k-clusters

For each cluster c_i

Compute $Maj^i = r_u \text{ x } \frac{MI}{MA} \text{ x } MC^i \; ; 1 \leq i \leq k_m, MA \neq 0$

While $Maj^i > 0$

Randomize instances in c_i

$s_i = c_i.pop()$

If s_i is not in S_{max}

k_s = find k-neighbours of s_i

If $|k_s|$-1 neighbours are minority

Discard

Else

S_{max}.push(s_i)

End while

End For

CHAPTER 4

IMPLEMENTATION AND TESTING OF THE HYBRID CLUSTER BASED SAMPLING ALGORITHM (HCBST)

4.0 Introduction

This chapter gives an in-depth overview of the design of the Hybrid Cluster-Based Sampling Algorithm. The Chapter is organized into two (2) main sections, Implementation and Testing of HCBST. The first section highlights on implementation of the proposed technique using the theoretical framework developed in Chapter 3. Finally, the last section highlights the processes used to validate the theoretical framework.

4.1 Implementation of HCBST

The complete source code for the implementation of HCBST together with the required import modules is provided in the appendices. HCBST was implemented in python using PyCharm IDE 2018 running in a 64bit Windows 10 environment.

As indicated in chapter 4, the algorithm of HCBST performs two main processes to produce the final output. The first is the oversampling process, and the second is the undersampling process. During oversampling process, the minority class sample dataset is traversed using a for loop, determining the nearest neighbours together with their corresponding distances of each minority instance, temporary array variables hold the neighbours, the distances to the neighbours and the corresponding mean. At the end of the iteration, the value of sigma is calculated as described in chapter 3. The code snippet that performs this operation is shown Code Listing 1 of the appendix.

After computing the value sigma, synthetic samples can now be generated using the sigma nearest neighbours technique. To perform this operation, an iteration is performed using a ‘for’ loop up to the number of required minority samples that must be generated. Within this for loop, a ‘while’

loop is used to search for nearest neighbours of a randomly selected sample whose distances are less than sigma. A parameter for limiting the search count is used to prevent the while loop from running forever if no suitable nearest neighbours are found. Upon finding suitable nearest neighbours, a synthetic sample is generated based on the convex procedure described in SNOCC[21]. The code snippet that performs this task is shown in Code Listing 2 of the appendix. After generating the synthetic sample, the nearest distance to a majority sample is computed. If the distance to the nearest majority instance is less than the average distances of the sigma neighbours to their nearest majority samples, the new synthetic sample is discarded. Otherwise, it is added to the new minority samples set. The original minority instances are appended to the newly generated samples. The overall oversampling processes are shown in Code Listing 3 of the appendix.

Upon successful completion of the oversampling phase, the algorithm proceeds with the under-sampling phase.

As described in chapter 3, the undersampling process begins by first determining the mode of undersampling. After determining the mode of undersampling, the algorithm proceeds with clustering using the k-means algorithm to cluster the majority samples into k_m clusters.

The code snippet that performs this operation is shown in Code Listing 3 of the appendix.

A for loop is used to iterate through all clusters; a parameter is used to determine whether to remove or select samples from the original majority based on the local proximity of minority samples For each cluster, if p_m is not set, then the algorithm proceeds using CUST. The required number of samples to be selected from each cluster is determined by the equation (3.7).

A random instance is then selected from the cluster, discarding instance if it is a replica of an already selected instance. When an instance S_{ci} is selected from the cluster, k_n nearest neighbours are searched, if $k_n - 1$ neighbours are minority instances, then S_{ci} is discarded. Otherwise, it is added to selected samples. The process is repeated using a while loop until the required number of instances to select from each cluster is obtained

The code snippet that implements this process is shown in Code Listing 4 in the appendix.

However, if p_m is set, then the number of majority samples to discard from each cluster is given by $Maj^i = p_m \text{ x MA} ; 1 \leq \text{i} \leq \text{k}_m$. A random instance S_{ci} is selected from the cluster, k_n nearest neighbours are searched, if $k_n - 1$ neighbours are minority instances, then S_{ci} is discarded. The process is repeated using a while loop until the required number of instances to remove from each cluster is obtained. The code snippet that performs this operation is shown in Code Listing 5 of the appendix:

The remaining instances in each cluster are combined with the minority samples to form the new training set.

4.2 Testing of HCBST

To examine the efficiency of the proposed technique, several test experiments were performed across a wide range of input parameters. As indicated in chapter 3, six NASA MDP software defect datasets and five datasets from the UCI repository were used to determine how well HCBST can

improve a variety of well-known classification algorithms. These algorithms include K-Nearest Neighbors, Support Vector Machines, Decision Tree, Random Forest, Neural Network, Adaboost, Gaussian Naive Bayes and Quadratic Discriminant Analysis. The range of parameters as outlined in chapter 3 is used to test HCBST are as follows:

r_u ={1, 1.25, 1.5, 1.75, 2.5, 2.75, 3, 5, 7, 10, 15, 20, 30}

r_o ={1, 1.25, 1.5, 1.75, 2.5, 2.75, 3, 5, 7, 10, 15, 20, 30}

p_m ={0.1, 0.15, 0.175, 0.2, 0.25, 0.35,0.4, 0.5, 0.6, 0.75}

k_m ={3, 5, 7, 10, 15, 20, 30, 50}

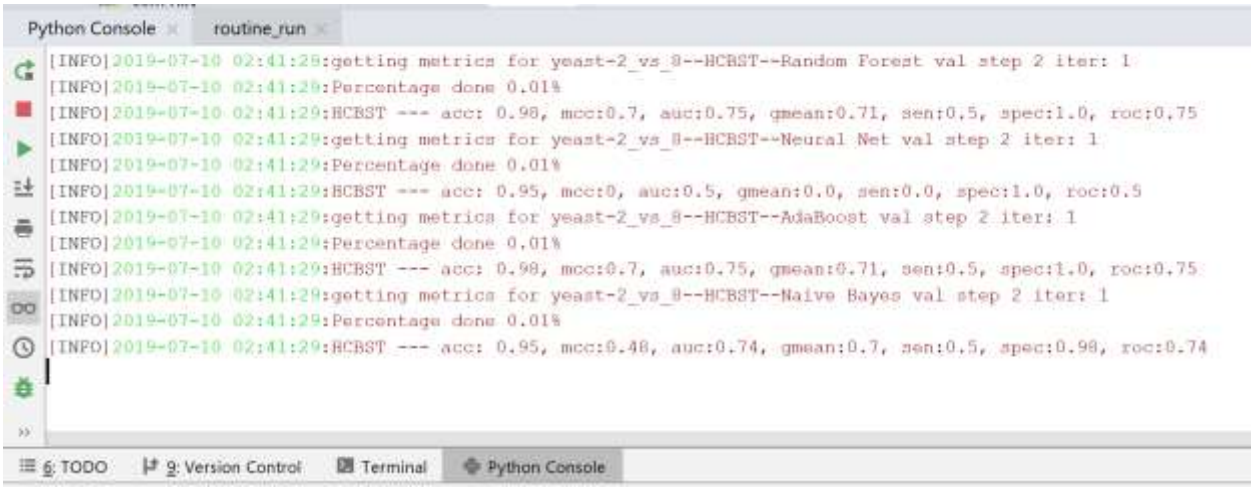


Figure 4.1 Snapshot of the Algorithm Running in PyCharm IDE

CHAPTER 5

RESULTS AND DISCUSSION

5.0 Introduction

This chapter discusses the experimental results obtained from this study and compares how well the proposed technique Hybrid Cluster-Based Sampling Algorithm (**HCBST),** performed against other sample techniques. The Chapter is organized into two (2) main sections, Results and Discussion, and Practical Application of HCBST. The Results and discussion section outlines and interpret the results first by comparing the performance of the proposed technique to performance of the classifiers when no sampling technique is applied then compares the performance of the proposed technique to other sampling techniques. Furthermore, the overall average performances of all the sampling techniques across each classifier are compared. Lastly, the results are subjected to ANOVA analysis to further enforce the validity of the theoretical framework. The second section demonstrates how the proposed technique can be used for real-world applications.

5.1 Results

To assess the extent to which HCBST improves traditional machine learning algorithm, the performance is first compared that of raw unsampled data using the various classifiers and then compared to the other sampling techniques specified in chapter 3. For each of the classifiers, the results in tables Table B.1 through to Table B.8 show the AUC performance, Table B.9 through to Table B.16 show the geometric mean (G-Mean) and Table B.17 through to Table B.24 show the Matthews Correlation Coefficient (MCC) on all the datasets. These tables can be found in Appendix B of the document. For each table, the first row shows the sampling techniques used in this study, and the first column displays the datasets. The sampling technique that produced that the highest performance for that metric is bold-faced. The average performance over all the datasets is shown in the last row.

5.2 HCBST vs. NONE Using AUC

The performance of the classification algorithms when HCBST is used to sample the data before classification is compared to the performance when no sampling is performed. This is to show that HCBST can be used to improve classification algorithms.

Figure 5.1 illustrates the performance of HCBST versus NONE in terms of AUC when using K Nearest Neighbours classifier. HCBST increased the performance of KNN with a minimum increase of 0.013, representing 1.93% and a maximum increase of 0.27, representing 54.23%. However, it fails in MC2 dataset with a decrease in performance by a value of 0.023, representing 3.46%.

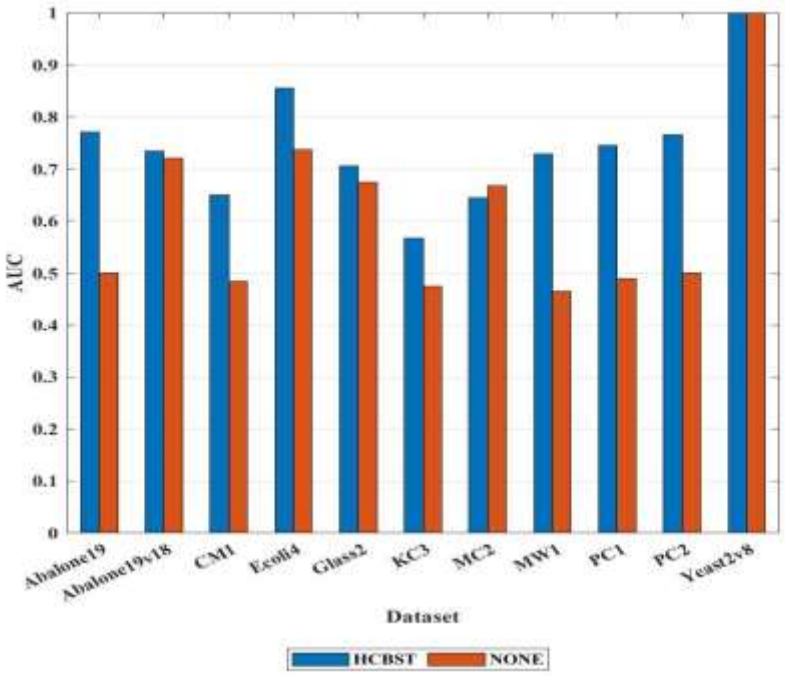


Figure 5.1 AUC of HCBST vs NONE for KNN Classifier

Figure 5.2 illustrates the performance of HCBST versus NONE in terms of AUC when using SVM classifier. HCBST increased the performance of SVM with a minimum increase of 0.05, representing 8.43% and a maximum increase of 0.42, representing 83.61%.

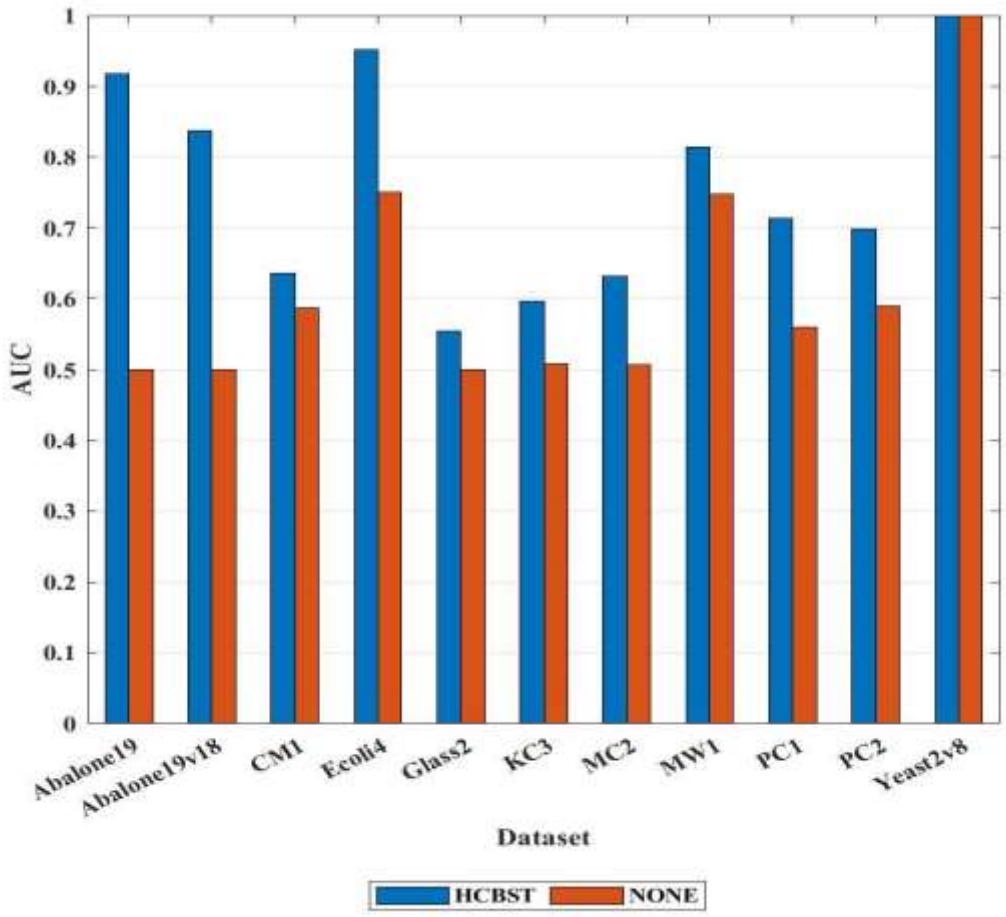


Figure 5.2 AUC of HCBST vs NONE for SVM Classifier

Figure 5.3 illustrates the performance of HCBST versus NONE in terms of AUC when using Decision Tree (DT) classifier. HCBST increased the performance of DT with a minimum increase of 0.02, representing 3.06% and a maximum increase of 0.2, representing 56.9%. However, it fails in ecoli4 MC2 datasets with a decrease in performance by a value of 0.002, representing 0.25% and 0.005 representing 0.65% respectively.

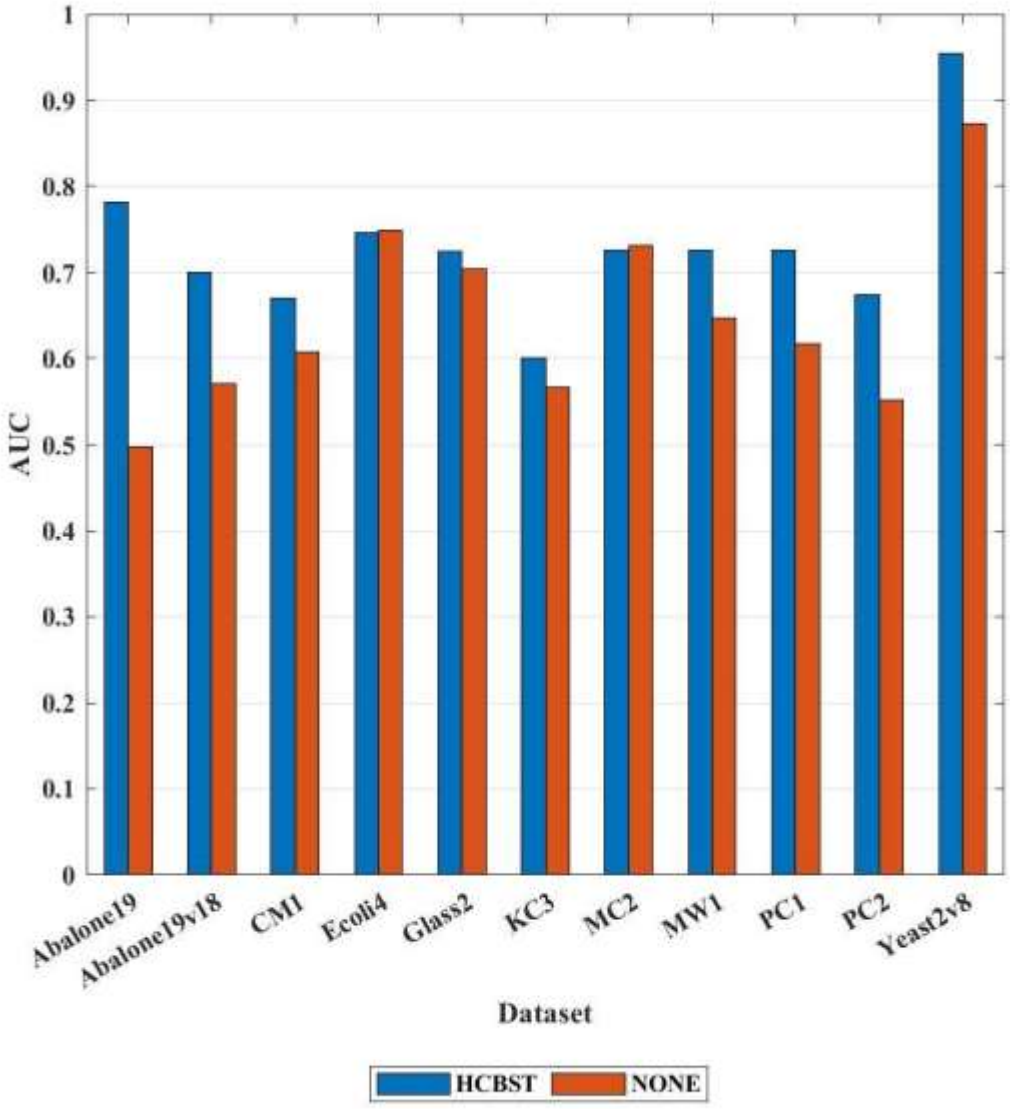


Figure 5.3 AUC of HCBST vs. NONE for Decision Tree Classifier

Figure 5.4 illustrates the performance of HCBST versus NONE in terms of AUC when using Random Forest (RF) classifier. HCBST increased the performance of RF with a minimum increase of 0.03, representing 4.45% in the MC2 dataset and a maximum increase of 0.307, representing 61.44% in the Abalone19 dataset.

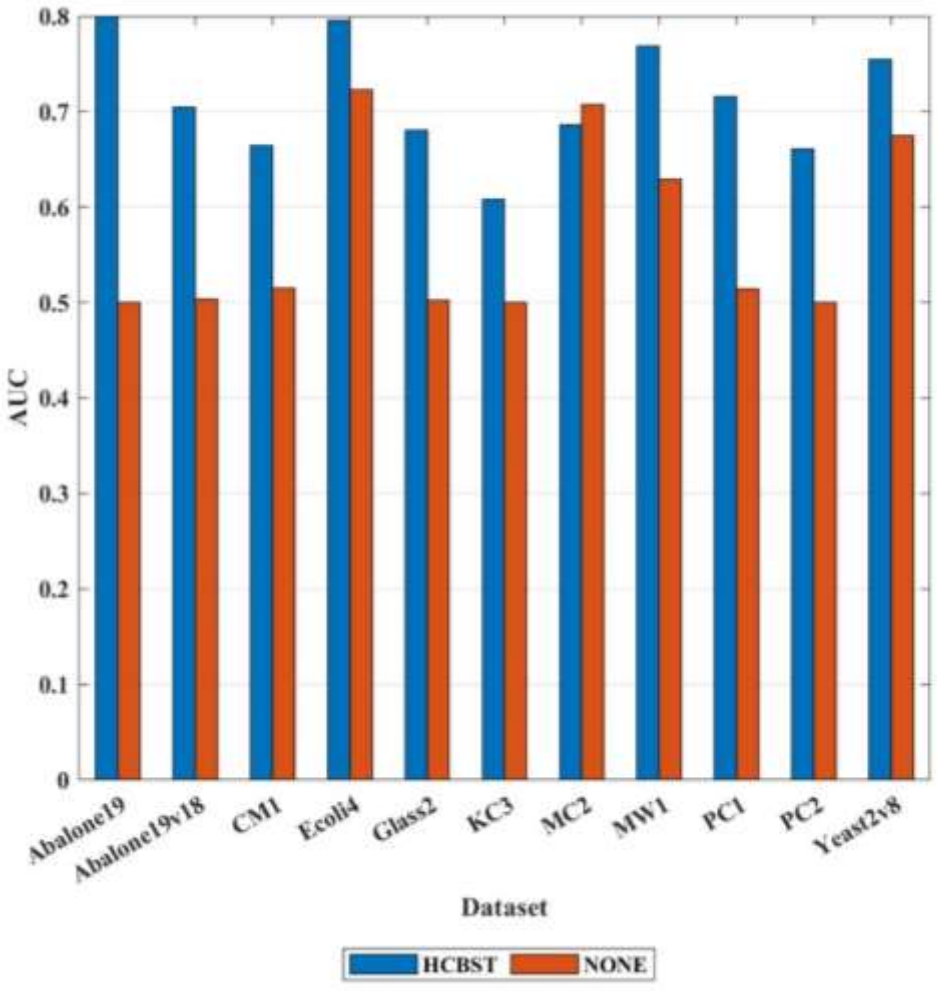


Figure 5.4 AUC of HCBST vs. NONE for Random Forest Classifier

Figure 5.5 illustrates the performance of HCBST versus NONE in terms of AUC when using a Neural Network (NN) classifier. HCBST increased the performance of NN with a minimum increase of 0.02, representing 3.86% in the PC2 dataset and a maximum increase of 0.38, representing 75.49% in the Abalone19 dataset.

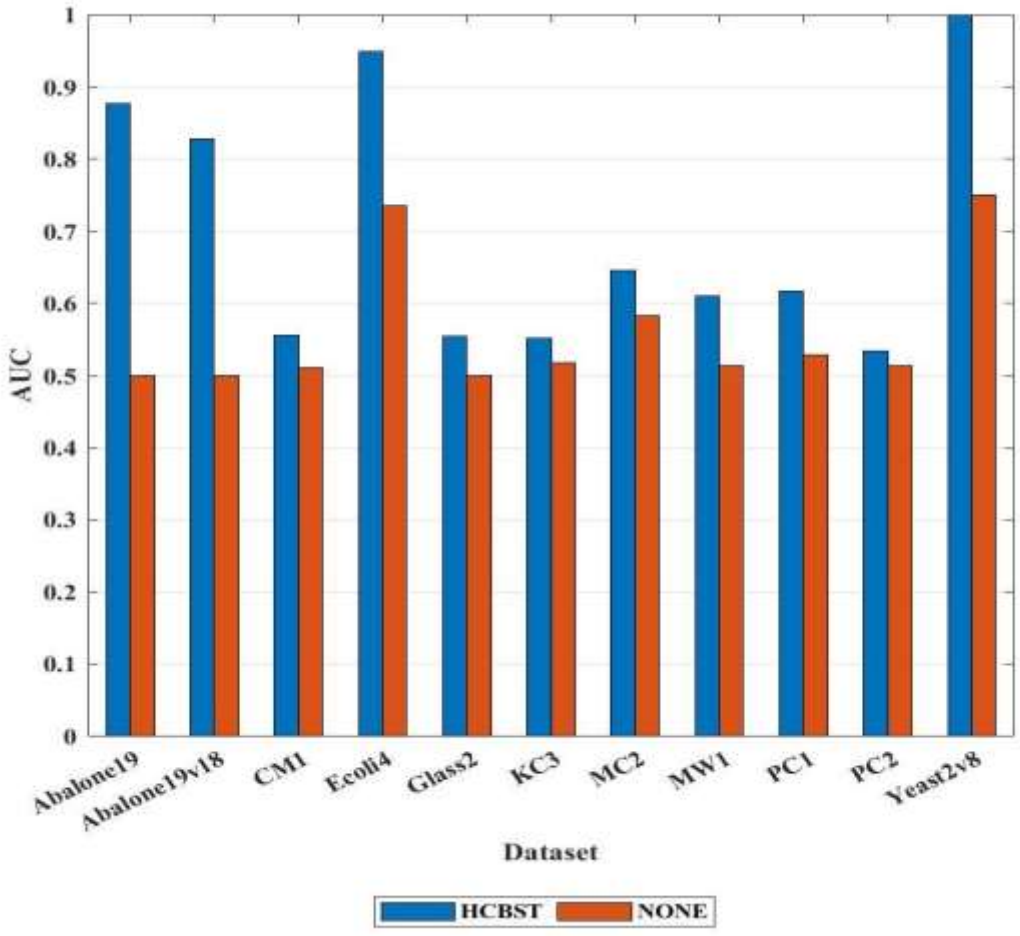


Figure 5.5 AUC of HCBST vs. NONE for Neural Network Classifier

Figure 5.6 illustrates the performance of HCBST versus NONE in terms of AUC when using AdaBoost classifier. HCBST increased the performance of the classifier with a minimum increase of 0.009, representing 1.17% in the ecoli4 dataset and a maximum increase of 0.296, representing 59.16% in the Abalone19 dataset.

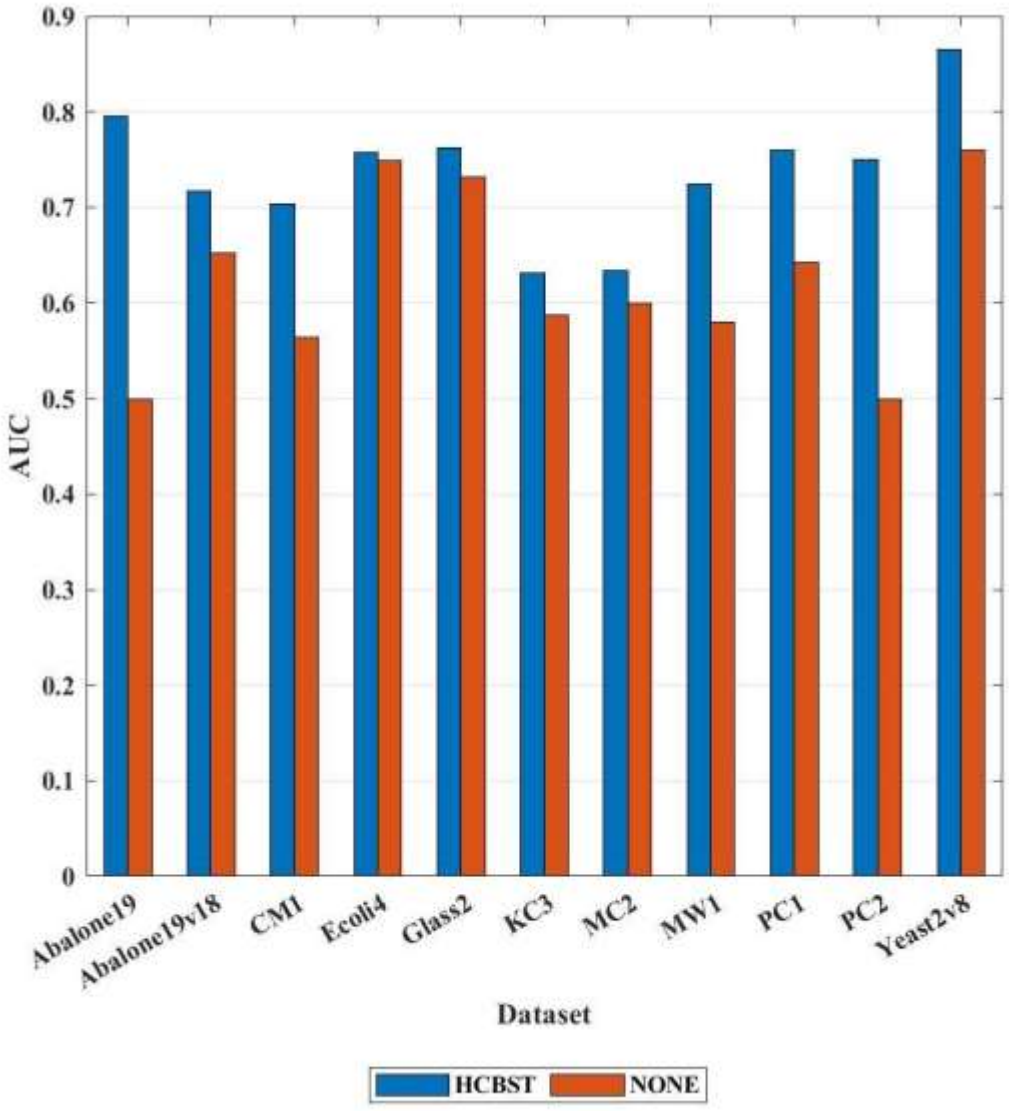


Figure 5.6 AUC of HCBST vs. NONE for AdaBoost Classifier

Figure 5.7 illustrates the performance of HCBST versus NONE in terms of AUC when using Naïve Bayes classifier. HCBST increased the performance of the classifier with a minimum increase of 0.006, representing 0.089% in the MC2 dataset and a maximum increase of 0.306, representing 45% in the Yeast2v8 dataset. However, it fails in PC1, PC2, and Abalone18v9 datasets with a decrease in performance by a value of 0.00073 representing 0.11 %, 0.0021 representing 0.65% and 0.33, representing 3.68% respectively.

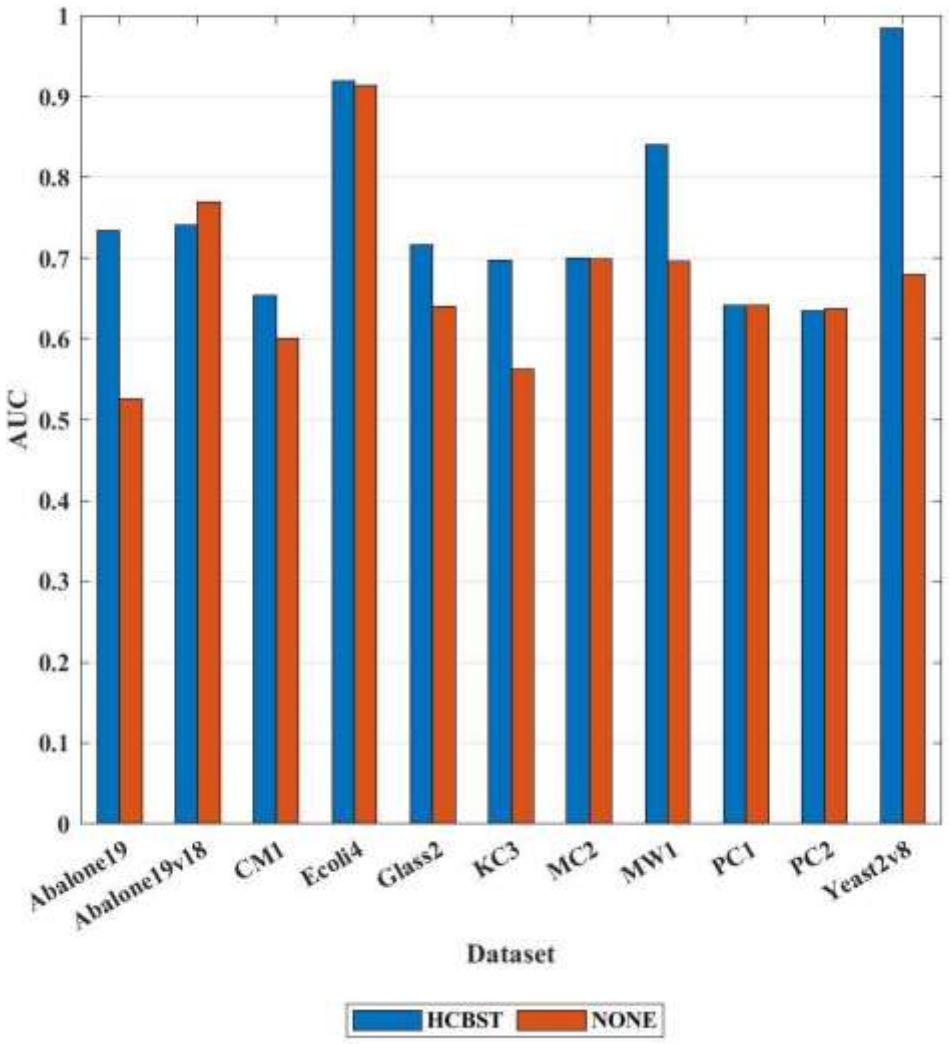


Figure 5.7 AUC of HCBST vs. NONE for Naïve Bayes Classifier

Figure 5.8 illustrates the performance of HCBST versus NONE in terms of AUC when using Quadratic Discriminant Analysis classifier. HCBST increased the performance of the classifier with a minimum increase of 0.0076, representing 1.1% in the PC1 dataset and a maximum increase of 0.238, representing 48.3% in the Abalone19 dataset.

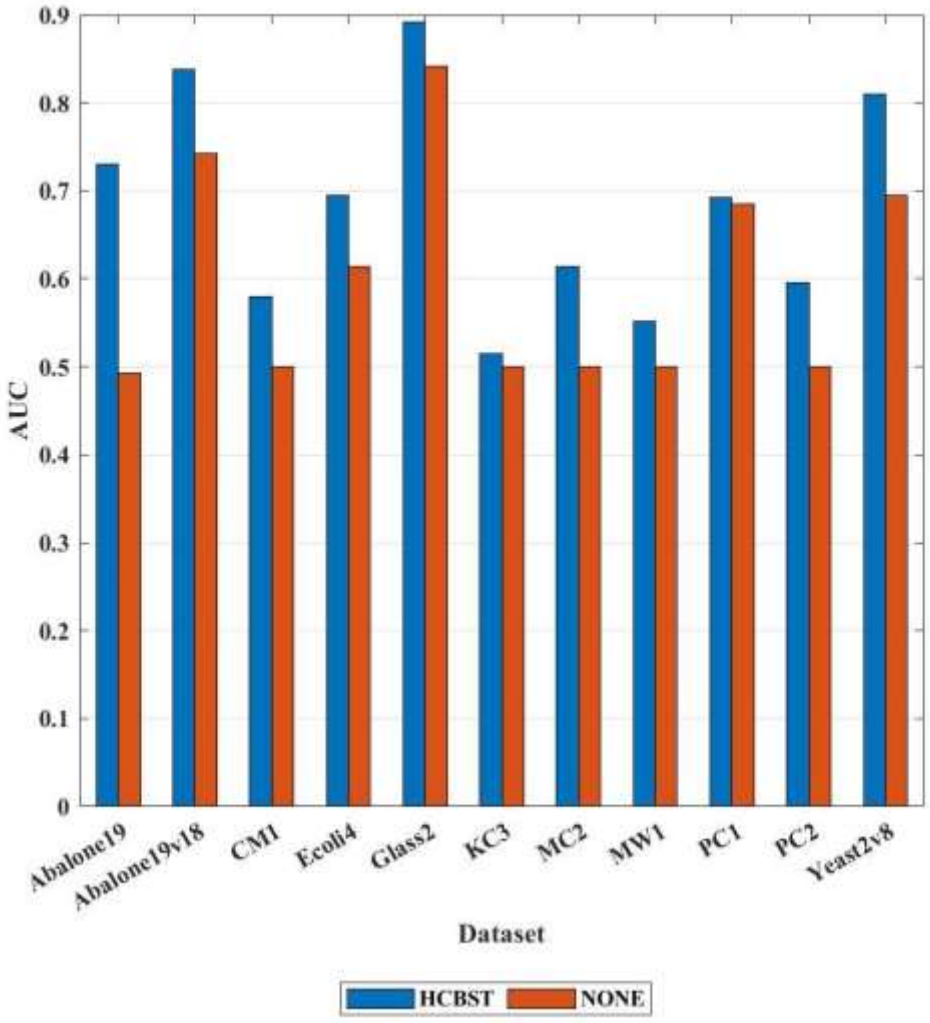


Figure 5.8 AUC of HCBST vs NONE for QDA Classifier

5.3 HCBST vs. NONE Using G-Mean

Figure 5.9 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the KNN classifier. HCBST increased the performance of the classifier with a minimum increase of 0.069, representing 10.47% in the Abalone9v18 dataset and a maximum increase of 0.627, representing 6434.25% in the CM1 dataset. An increase in performance from 0 to 0.756, 0.734, 0.725, 0.55, 0.25 and 0.39 in Abalone19, PC2, PC1, MW1 and KC3 respectively was observed. The 0 G-Mean values were as a result of the inability of the classifier to record an average TPR greater than 0. In other words, no minority class instances were correctly classified. However, it fails in MC2 With a decrease in performance by 0.027 representing 4.134%

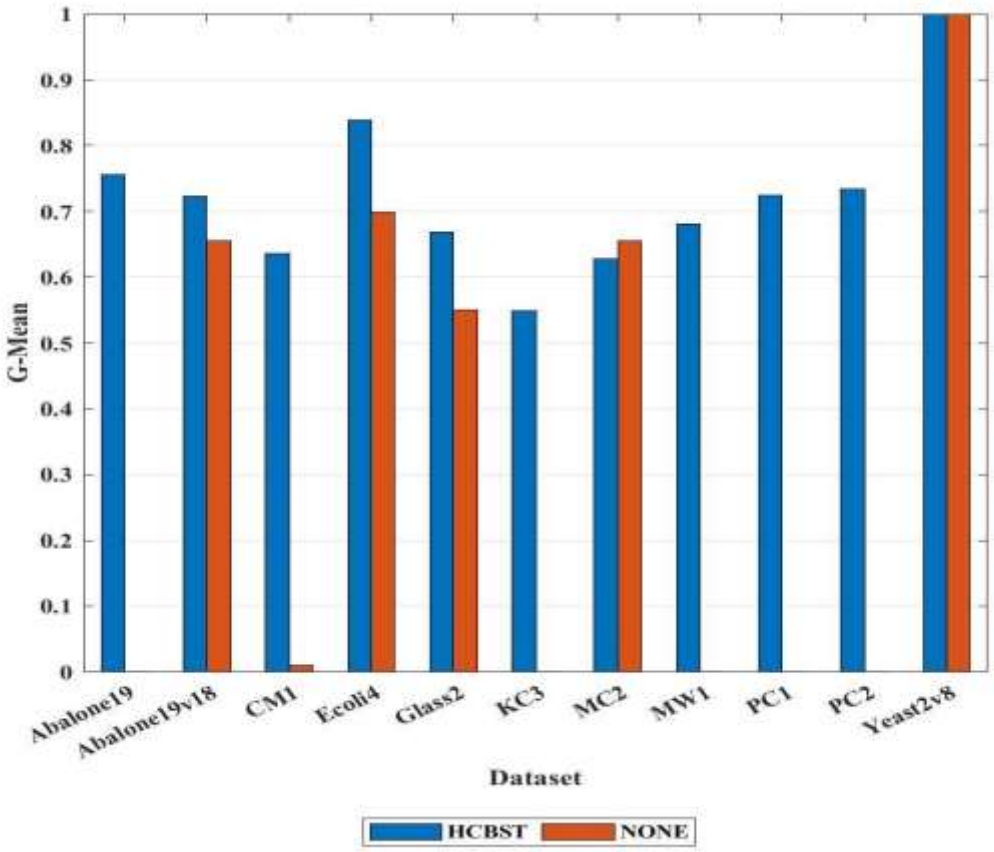


Figure 5.9 G-Mean of HCBST vs. NONE for KNN Classifier

Figure 5.10 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the Linear SVM classifier. HCBST increased the performance of the classifier with a minimum increase of 0.113, representing 16.652% in the MW1 dataset and a maximum increase of 0.306, representing 90.785% in the PC1 dataset. An increase in performance from 0 to 0.915, 0.833, 0.296 in abalone19, abalone9v18, glass2 respectively.

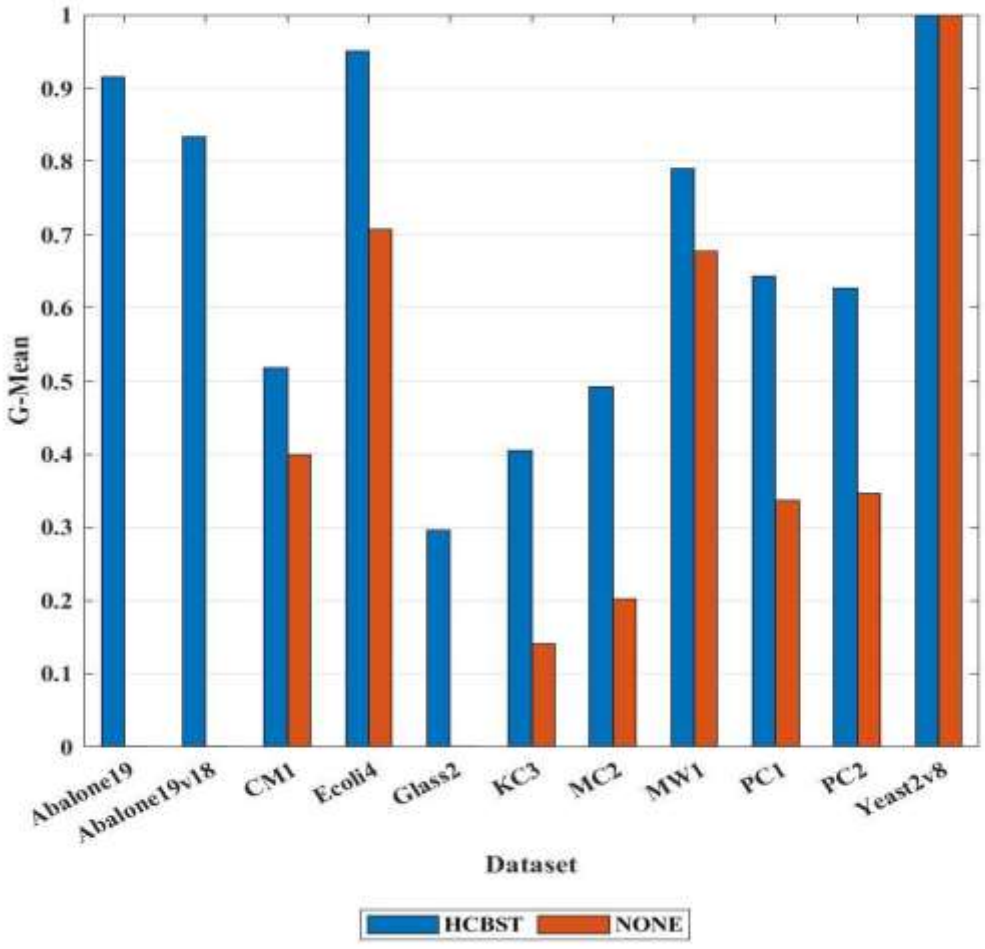


Figure 5.10 G-Mean of HCBST vs NONE for SVM Classifier

Figure 5.11 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the Decision Tree classifier. HCBST increased the performance of the classifier with a minimum increase of 0.004 representing 0.623% in the ecoli4 dataset and a maximum increase of 0.445 representing 300.762% in the PC2 dataset. An increase in performance from 0 to 0.767 in abalone19, respectively.

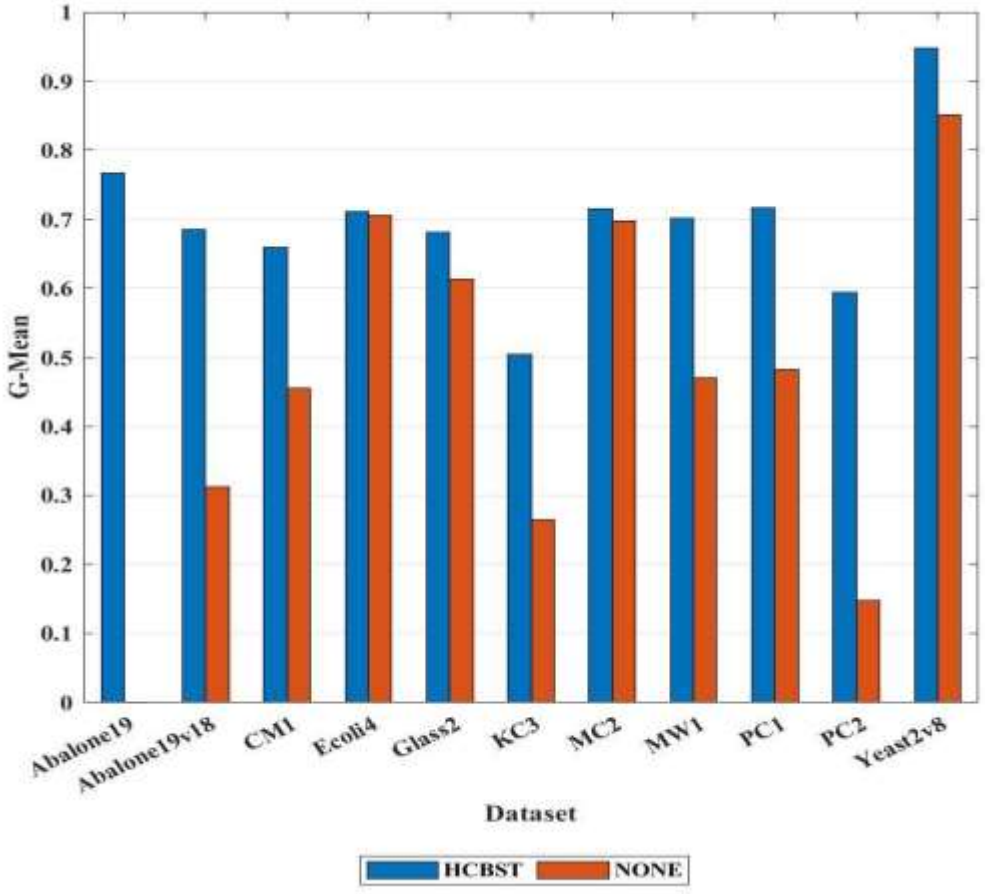


Figure 5.11 G-Mean of HCBST vs. NONE for Decision Tree Classifier

Figure 5.12 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the Random Forest classifier. HCBST increased the performance of the classifier with a minimum increase of 0.079 representing 12.339% in the MC2 dataset and a maximum increase of 0.667 representing 2226.52% in the abalone9v18 dataset. An increase in performance from 0 to 0.651, 0.798 in PC2, abalone19 respectively.

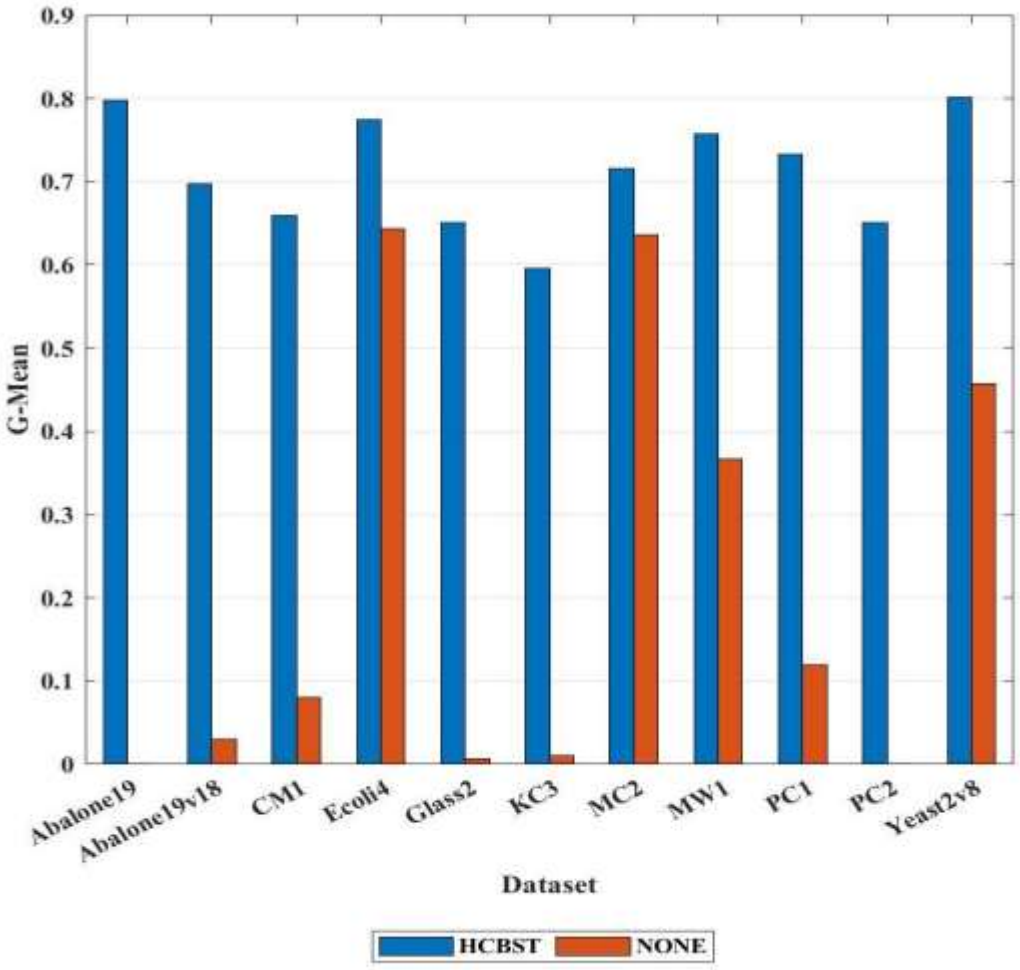


Figure 5.12 G-Mean of HCBST vs. NONE for Random Forest Classifier

Figure 5.13 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the Neural Net classifier. HCBST increased the performance of the classifier with a minimum increase of 0.106 representing 57.242% in the KC3 dataset and a maximum increase of 0.295 representing 143.847% in the PC1 dataset. An increase in performance from 0 to 0.869, 0.823, 0.326 in abalone19, abalone9v18, glass2 respectively.

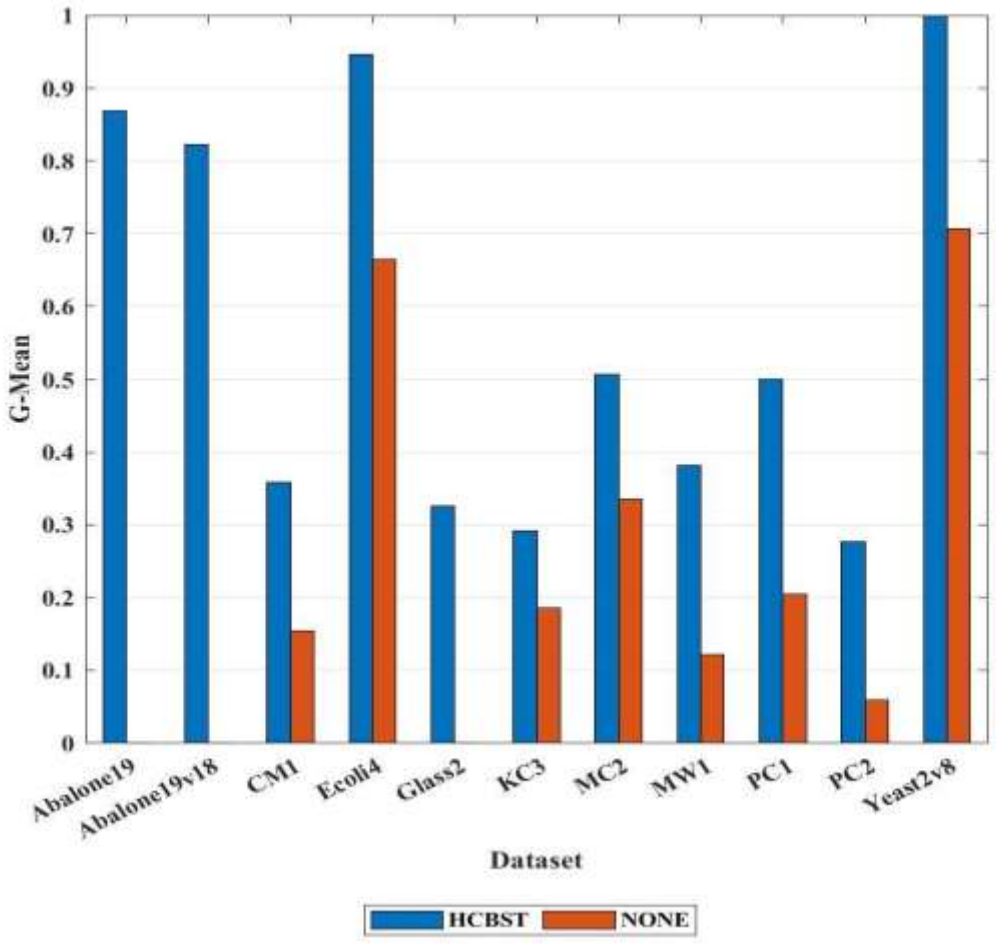


Figure 5.13 G-Mean of HCBST vs. NONE for Neural Network Classifier

Figure 5.14 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the AdaBoost classifier. HCBST increased the performance of the classifier with a minimum increase of 0.029 representing 4.102% in the ecoli4 dataset and a maximum increase of 0.704 representing 10096.427% in the PC2 dataset. An increase in performance from 0 to 0.783 in abalone19, respectively.

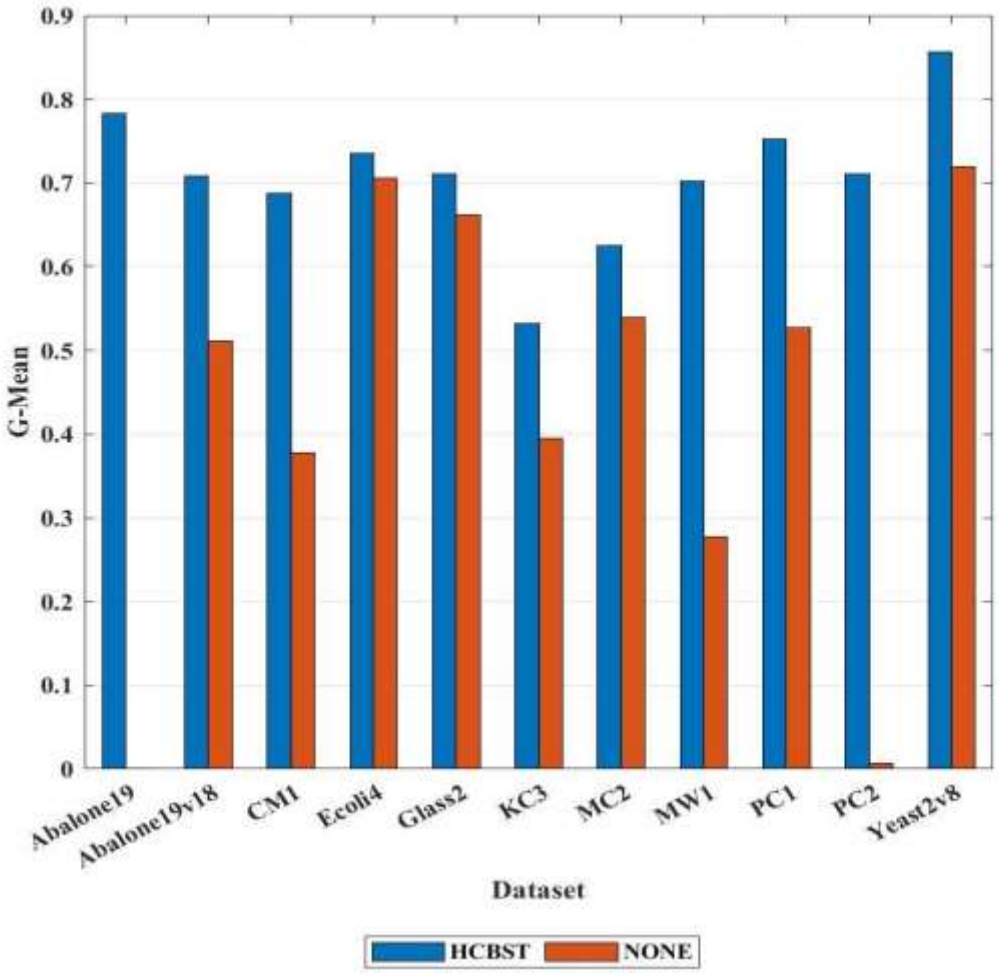


Figure 5.14 G-Mean of HCBST vs. NONE for AdaBoost Classifier

Figure 5.15 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the Naive Bayes classifier. HCBST increased the performance of the classifier with a minimum increase of 0.00019 representing 0.065% in the MC2 dataset and a maximum increase of 0.413 representing 141.365% in the abalone19 dataset. However, it fails in abalone9v18 with a decrease in performance by 0.033, representing 4.266% respectively.

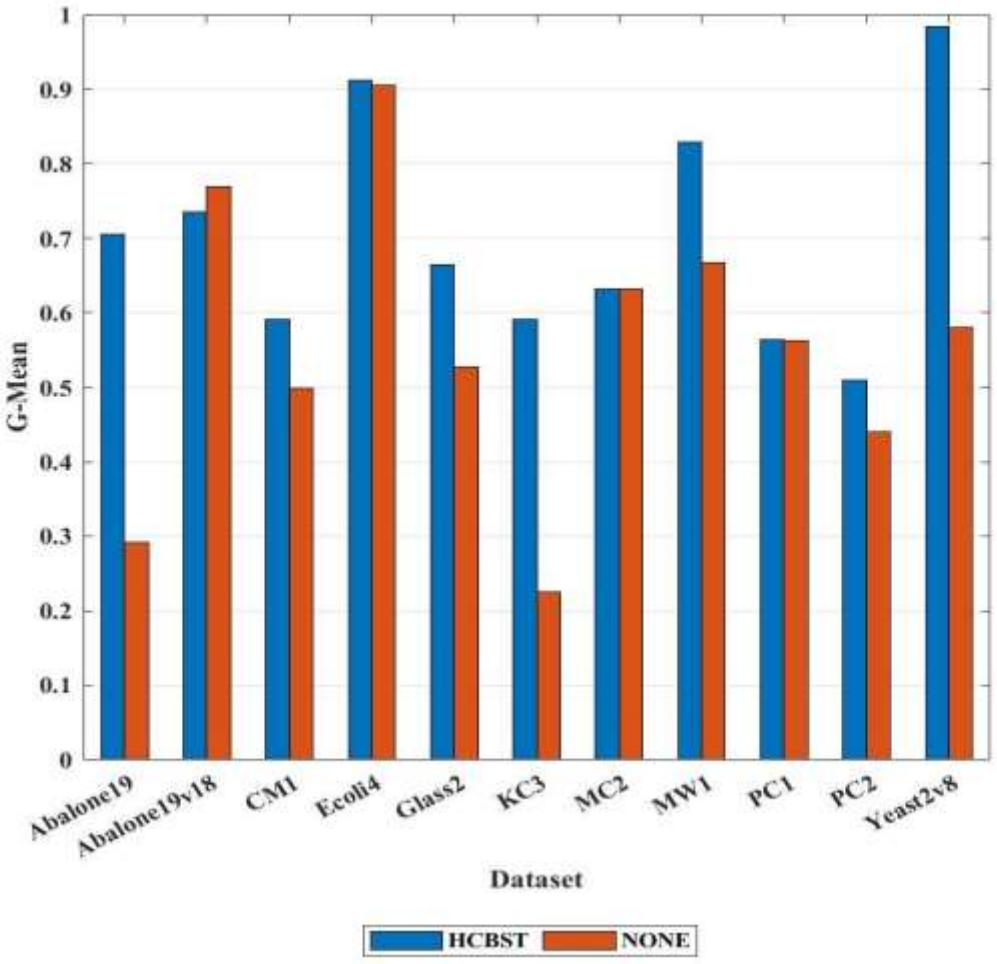


Figure 5.15 G-Mean of HCBST vs. NONE for Naïve Bayes Classifier

Figure 5.16 illustrates the performance of HCBST versus NONE in terms of G-Mean when using the QDA classifier. HCBST increased the performance of the classifier with a minimum increase of 0.049 representing 7.81% in the PC1 dataset and a maximum increase of 0.205 representing 63.52% in the ecoli4 dataset. An increase in performance from 0 to 0.391, 0.25, 0.402, 0.181, 0.373, 0.724 in CM1, KC3, MC2, MW1, PC2, abalone19 respectively.

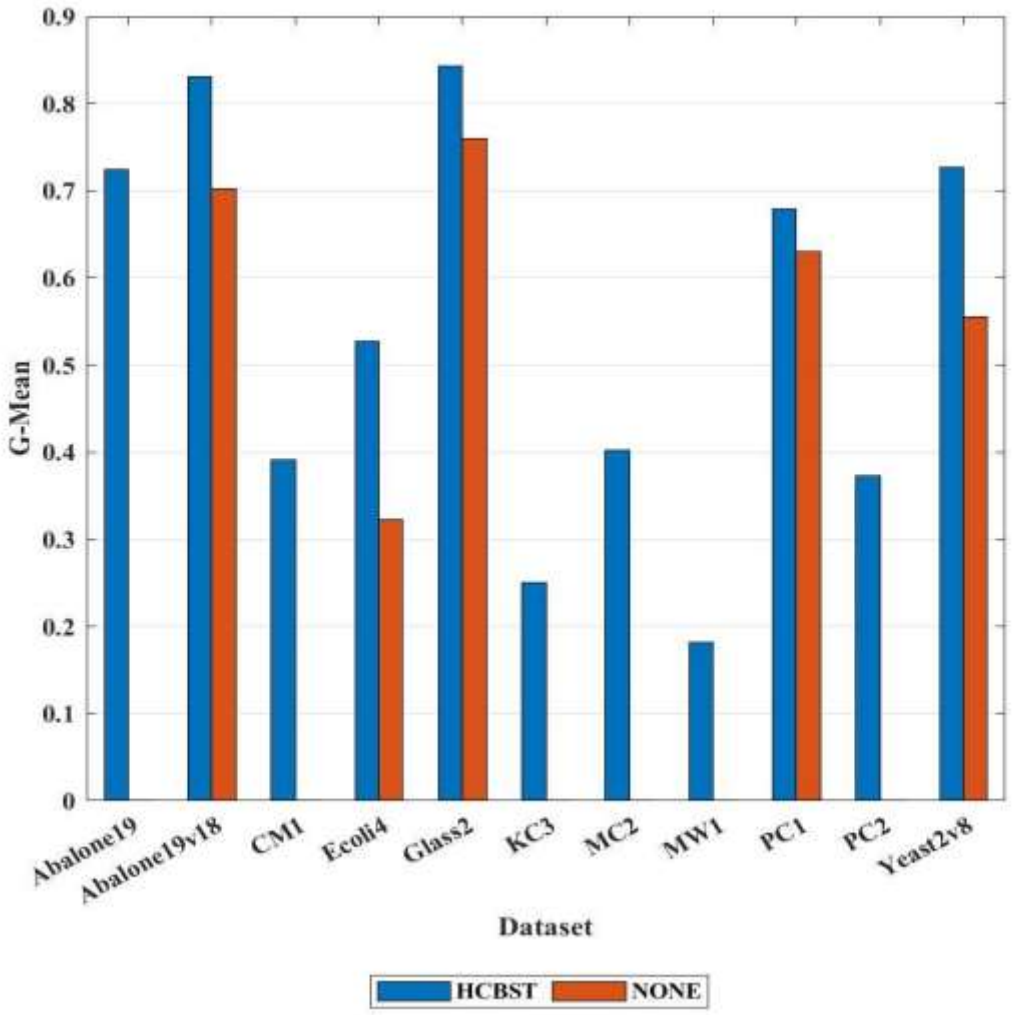


Figure 5.16 G-Mean of HCBST vs NONE QDA Classifier

5.4 HCBST vs. NONE Using MCC

Figure 5.17 illustrates the performance of HCBST versus NONE in terms of MCC when using the Nearest Neighbors classifier. HCBST increased the performance of the classifier with a minimum increase of 0.03 representing 5.84% in the ecoli4 dataset and a maximum increase of 0.359 representing 511.486% in the MW1 dataset. However, it fails in MC2, abalone9v18 With a decrease in performance by 0.043, 0.281 representing 12.399%, 43.67% respectively. An increase in performance from 0 to 0.203 in PC2, respectively.

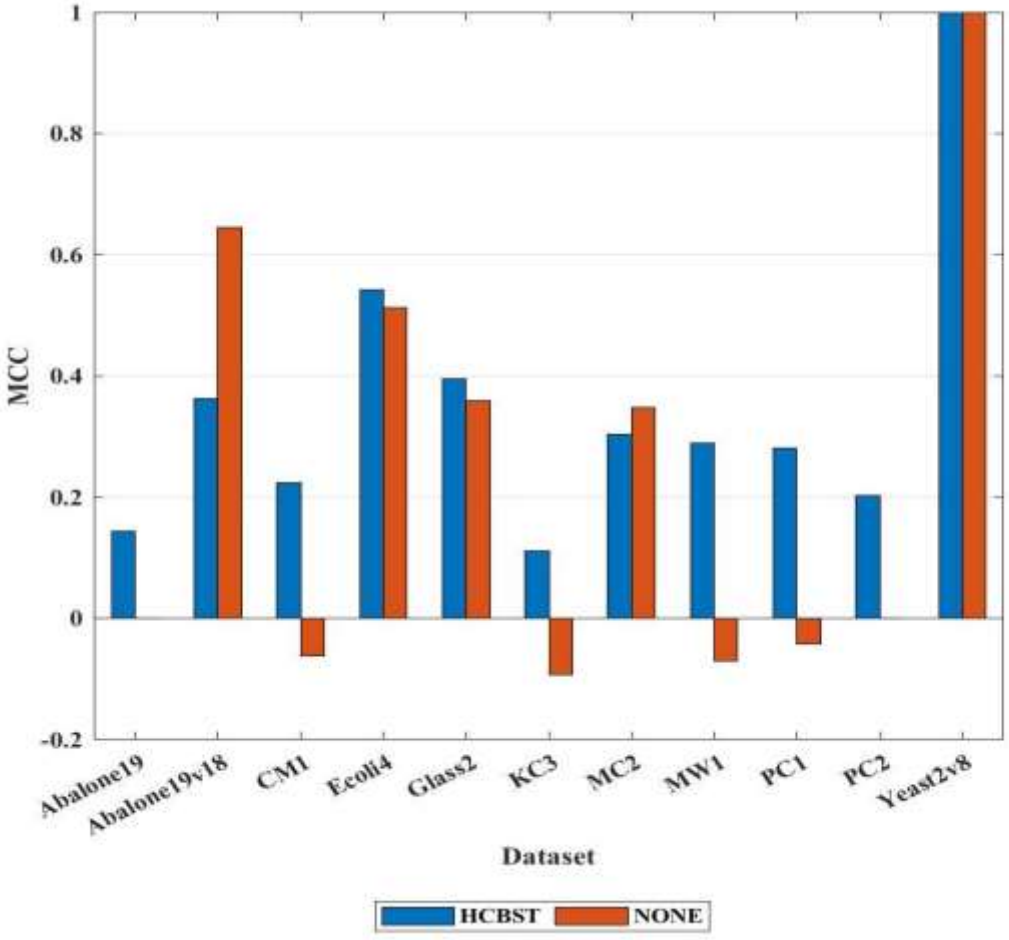


Figure 5.17 MCC of HCBST vs NONE KNN Classifier

Figure 5.17 illustrates the performance of HCBST versus NONE in terms of MCC when using the Linear SVM classifier. HCBST increased the performance of the classifier with a minimum increase of 0.062 representing 69.49% in the PC2 dataset and a maximum increase of 0.287 representing 14751.232% in the MC2 dataset. However, it fails in MW1 With a decrease in performance by 0.075 representing 12.943% respectively. An increase in performance from 0 to 0.212, 0.506, 0.077 in abalone19, abalone9v18, glass2 respectively.

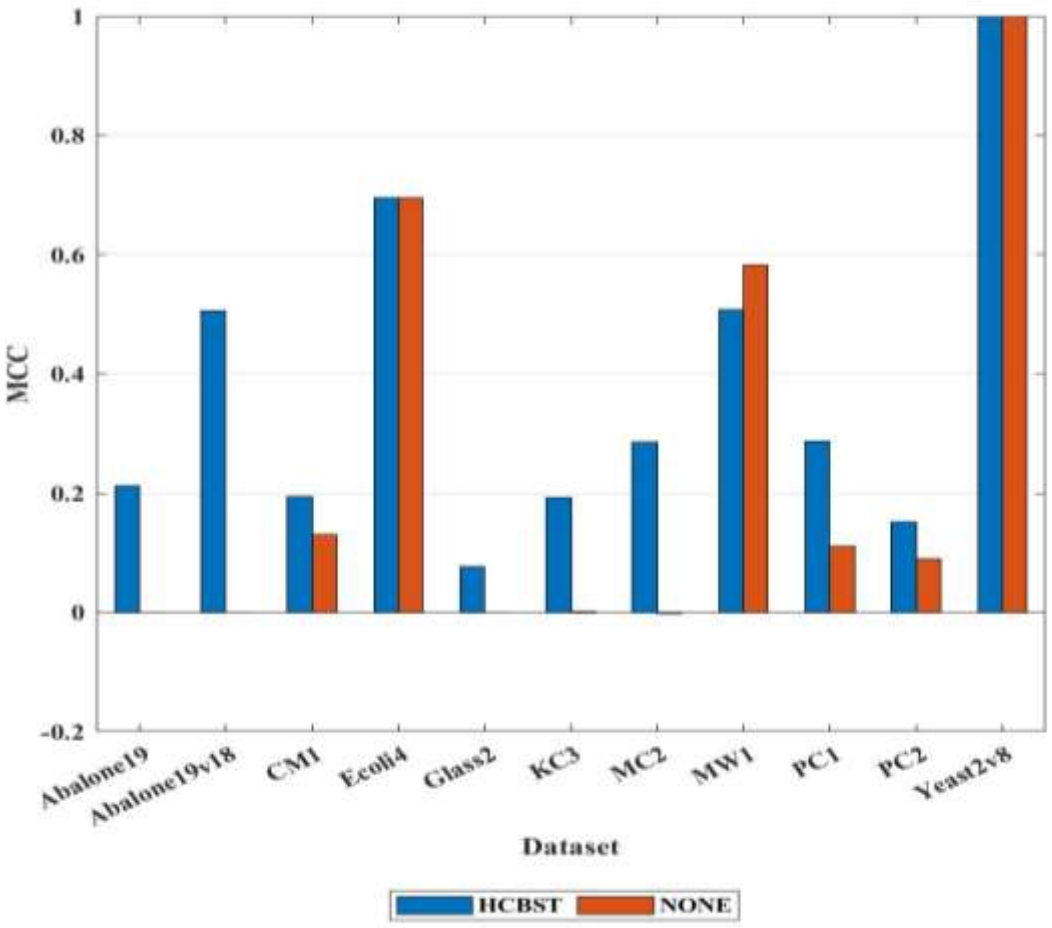


Figure 5.18 MCC of HCBST vs NONE SVM Classifier

Figure 5.19 illustrates the performance of HCBST versus NONE in terms of MCC when using the Decision Tree classifier HCBST. increased the performance of the classifier with a minimum increase of 0.012 representing 9.11% in the PC2 dataset and a maximum increase of 0.125 representing 2445.612% in the abalone19 dataset. However it fails in MC2, PC1, ecoli4, glass2 With a decrease in performance by 0.037, 0.015, 0.024, 0.001 representing 6.785%, 4.134%, 3.5%, 0.138% respectively.

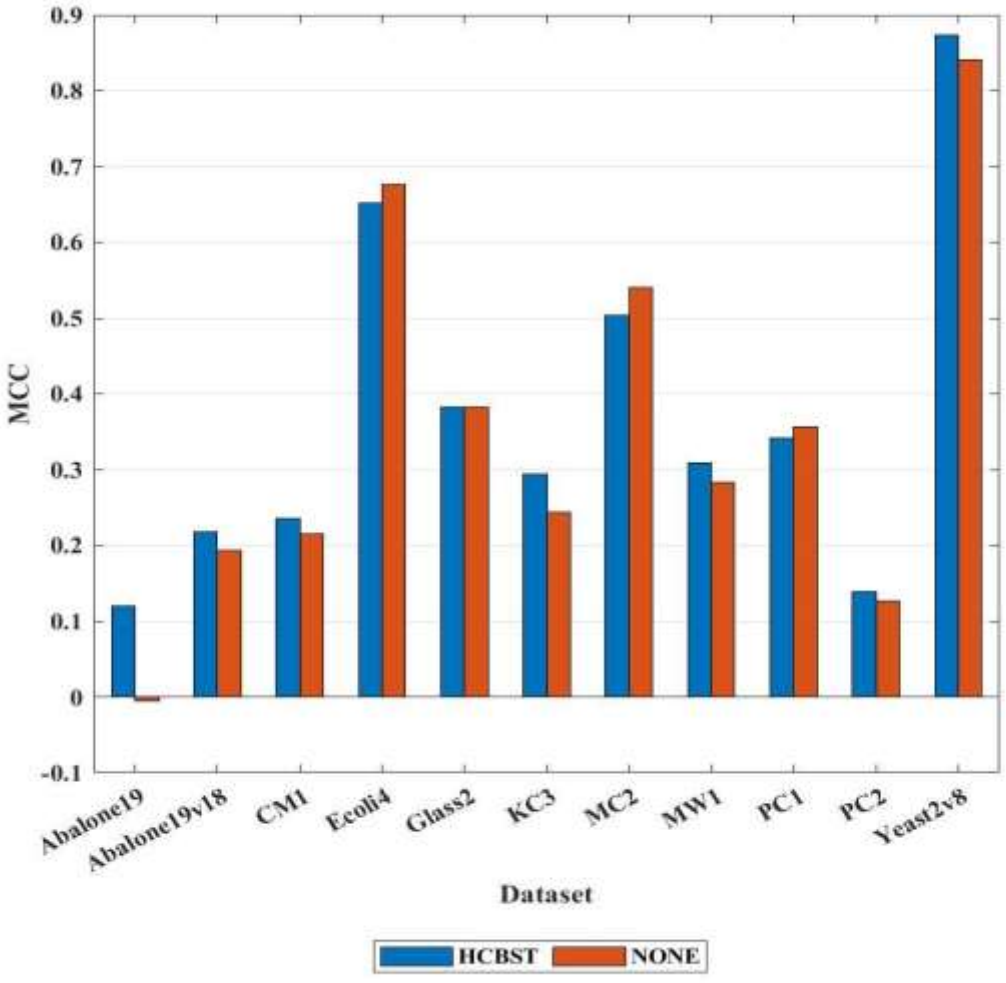


Figure 5.19 MCC of HCBST vs NONE Decision Tree Classifier

Figure 5.20 illustrates the performance of HCBST versus NONE in terms of MCC when using the Random Forest classifier HCBST. increased the performance of the classifier with a minimum increase of 0.054 representing 8.454% in the ecoli4 dataset and a maximum increase of 0.295 representing 3220.129% in the KC3 dataset. However, it fails in MC2 With a decrease in performance by 0.008, representing 1.456% respectively. An increase in performance from 0 to 0.119, 0.176 in PC2, abalone19 respectively.

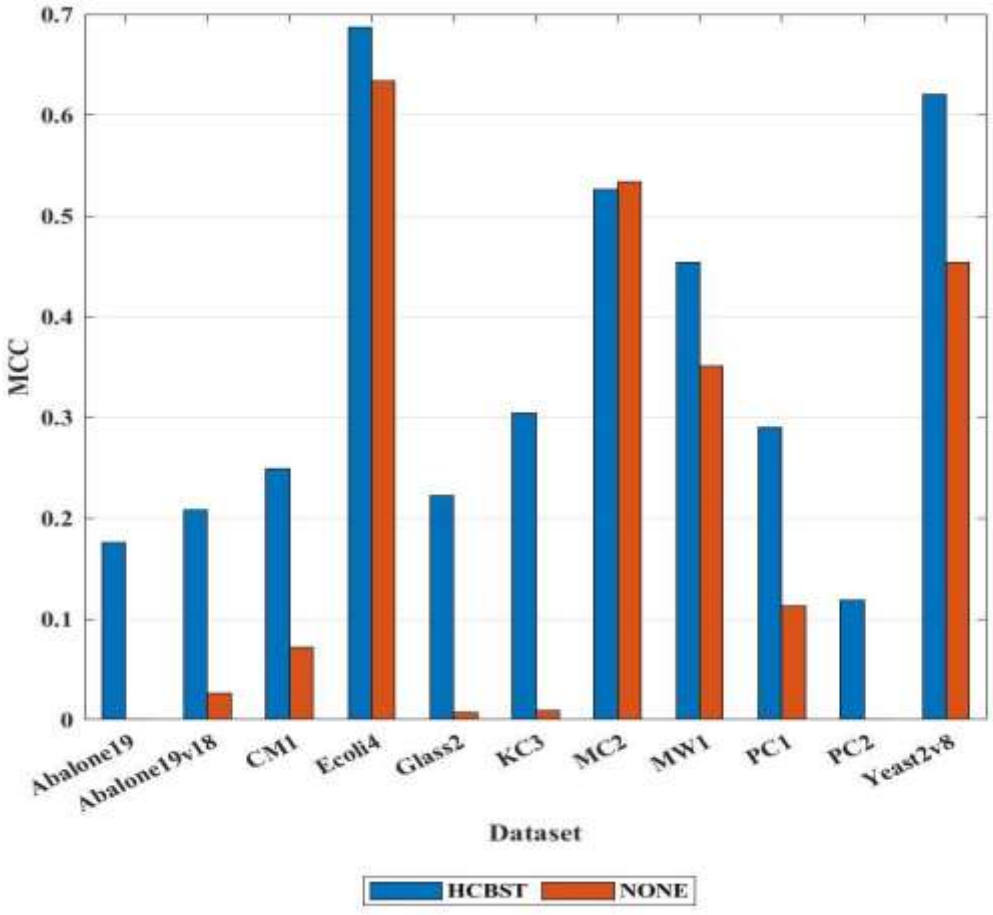


Figure 5.20 MCC of HCBST vs NONE using Random Forest Classifier

Figure 5.21 illustrates the performance of HCBST versus NONE in terms of MCC when using the Neural Net classifier HCBST. increased the performance of the classifier with a minimum increase of 0.015 representing 88.087% in the PC2 dataset and a maximum increase of 0.3 representing 42.918% in the yeast-2_vs_8 dataset. An increase in performance from 0 to 0.162, 0.496, 0.068 in abalone19, abalone9v18, glass2 respectively.

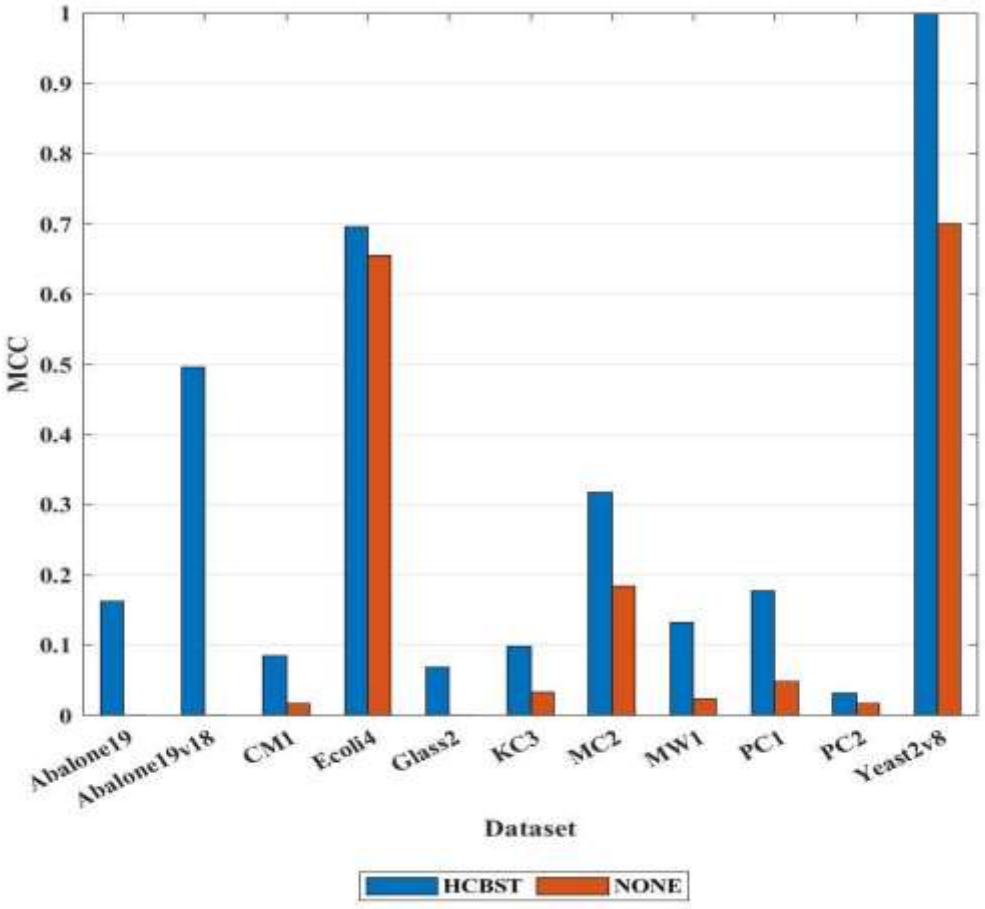


Figure 5.21 MCC of HCBST vs. NONE using Neural Network Classifier

Figure 5.22 illustrates the performance of HCBST versus NONE in terms of MCC when using the AdaBoost classifier. HCBST increased the performance of the classifier with a minimum increase of 0.035 representing 12.821% in the KC3 dataset and a maximum increase of 0.19 representing 3220.151% in the PC2 dataset. However, it fails in PC1, abalone9v18, ecoli4, glass2, yeast-2_vs_8 With a decrease in performance by 0.087, 0.149, 0.002, 0.003, 0.021 representing 21.012%, 34.185%, 0.334%, 0.509%, 3.268% respectively.

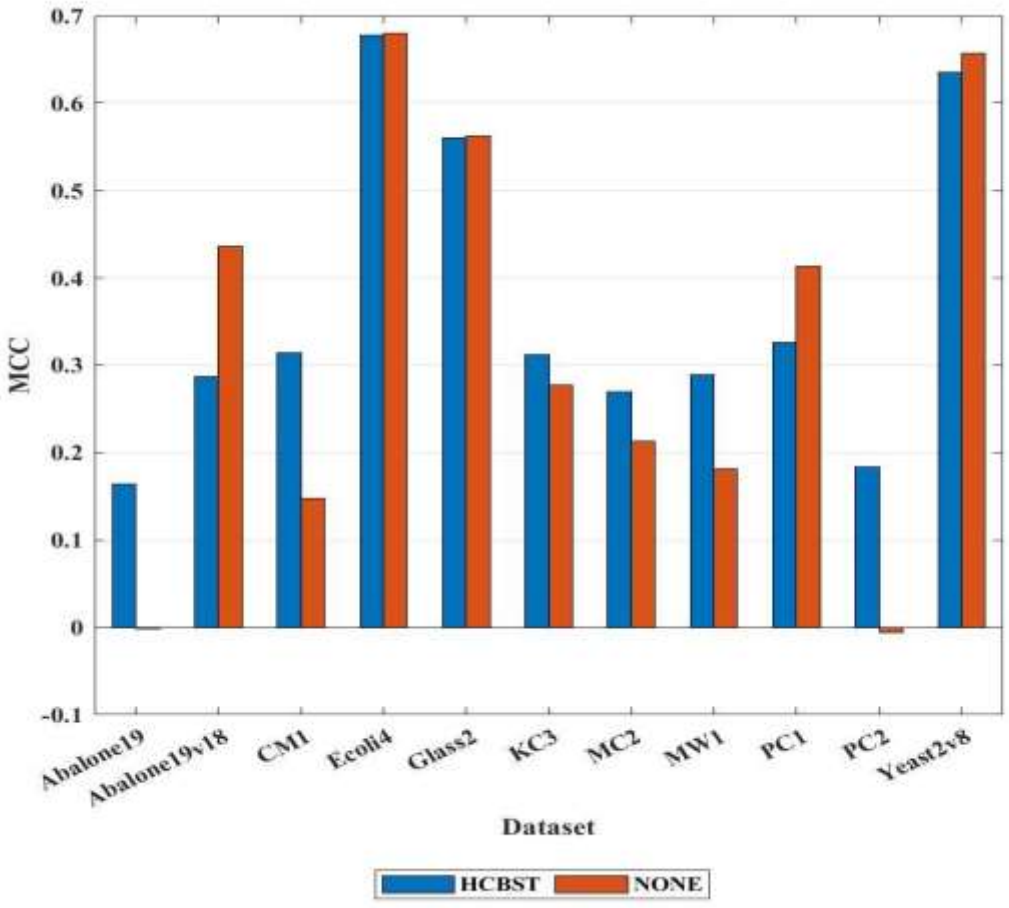


Figure 5.22 MCC of HCBST vs NONE using AdaBoost Classifier

Figure 5.23 illustrates the performance of HCBST versus NONE in terms of MCC when using the Naive Bayes classifier. HCBST increased the performance of the classifier with a minimum increase of 0.002 representing 0.415% in the MC2 dataset and a maximum increase of 0.677 representing 315.483% in the yeast-2_vs_8 dataset. However, it fails in PC2, abalone9v18 with a decrease in performance by 0.079, 0.036 representing 39.434%, 12.482% respectively.

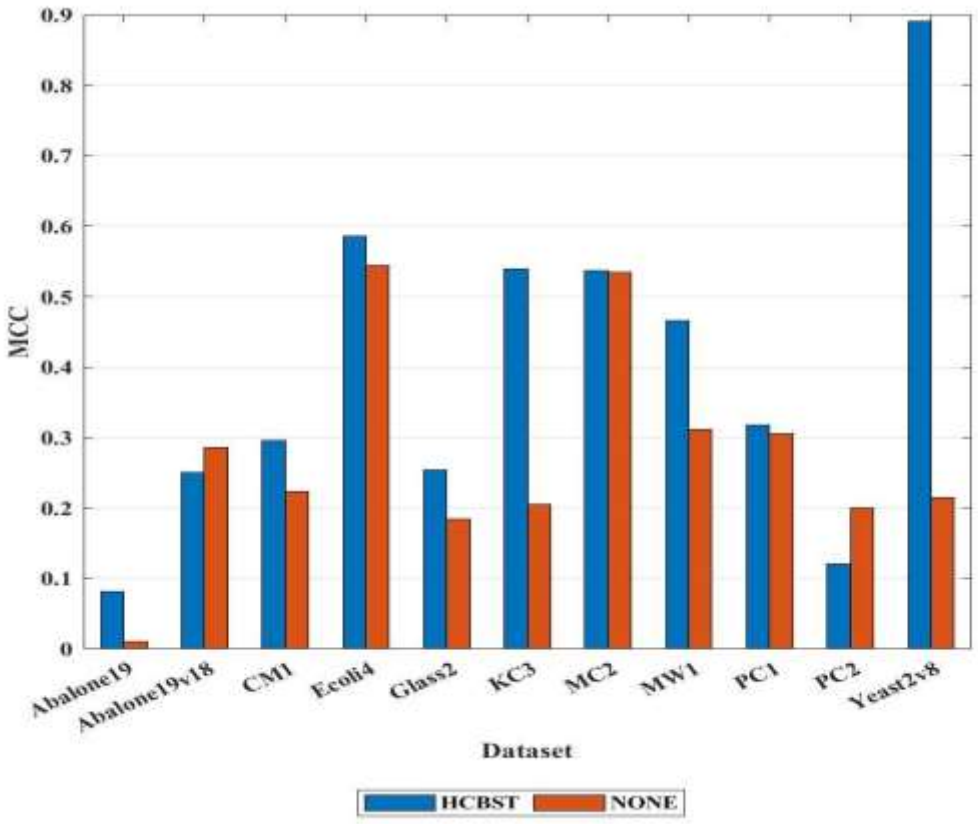


Figure 5.23 MCC of HCBST vs. NONE using Naïve Bayes Classifier

Figure 5.24 illustrates the performance of HCBST versus NONE in terms of MCC when using the QDA classifier. HCBST increased the performance of the classifier with a minimum increase of 0.043 representing 7.411% in the glass2 dataset and a maximum increase of 0.181 representing 71.22% in the yeast-2_vs_8 dataset. However, it fails in PC1, abalone9v18 With a decrease in performance by 0.006, 0.018 representing 1.707%, 3.178% respectively. An increase in performance from 0 to 0.15, 0.035, 0.281, 0.14, 0.072 in CM1, KC3, MC2, MW1, PC2 respectively.

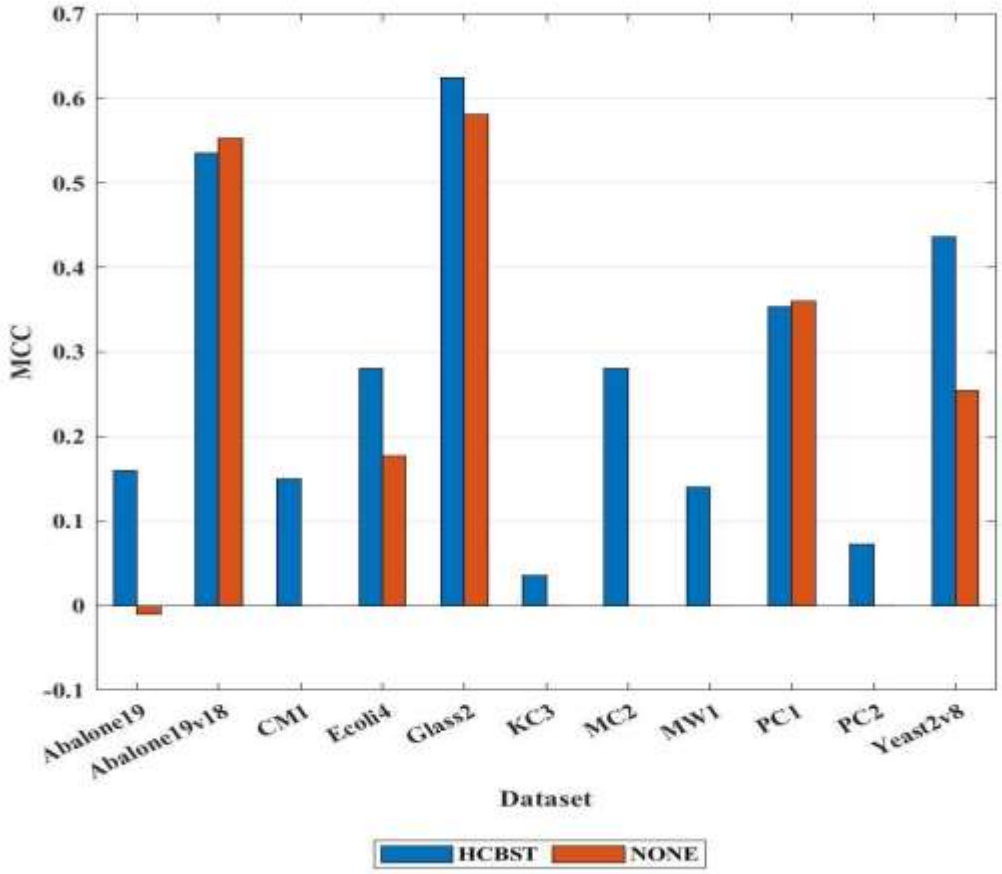


Figure 5.24 MCC of HCBST vs NONE using QDA Classifier

5.5 Overall Average Performance of HCBST vs. NONE Using AUC

Figure 5.25 showed the average AUC performances over all the datasets when HCBST was used to resample the data versus the performance when no sampling was done. The results show that HCBST provided a higher average performance concerning AUC in all the classifiers used in this study with a minimum improvement of 0.082 representing 12.19% with Naïve Bayes classifier and a maximum improvement of 0.16 representing 28.27% with Random Forest classifier.

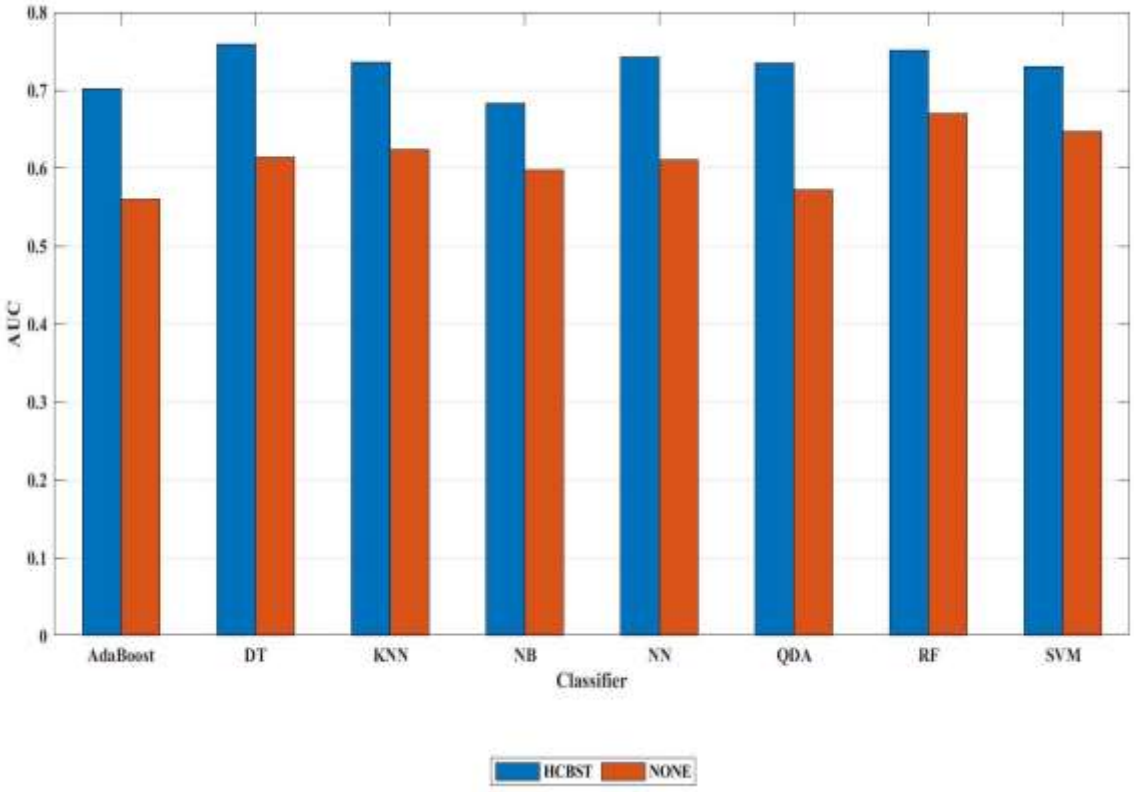


Figure 5.25 Average AUC Performance comparison of HCBST vs. NONE using all the datasets

5.6 Overall Average Performance of HCBST vs. NONE Using G-Mean

Figure 5.26 showed the average G-Mean performances over all the datasets when HCBST was used to resample the data versus the performance when no sampling was done. The results show that HCBST provided a higher average performance concerning G-Mean in all the classifiers used in this study with a minimum improvement of 0.147 representing 26.5% with Naïve Bayes classifier and a maximum improvement of 0.498 representing 233.2% with Random Forest Classifier.

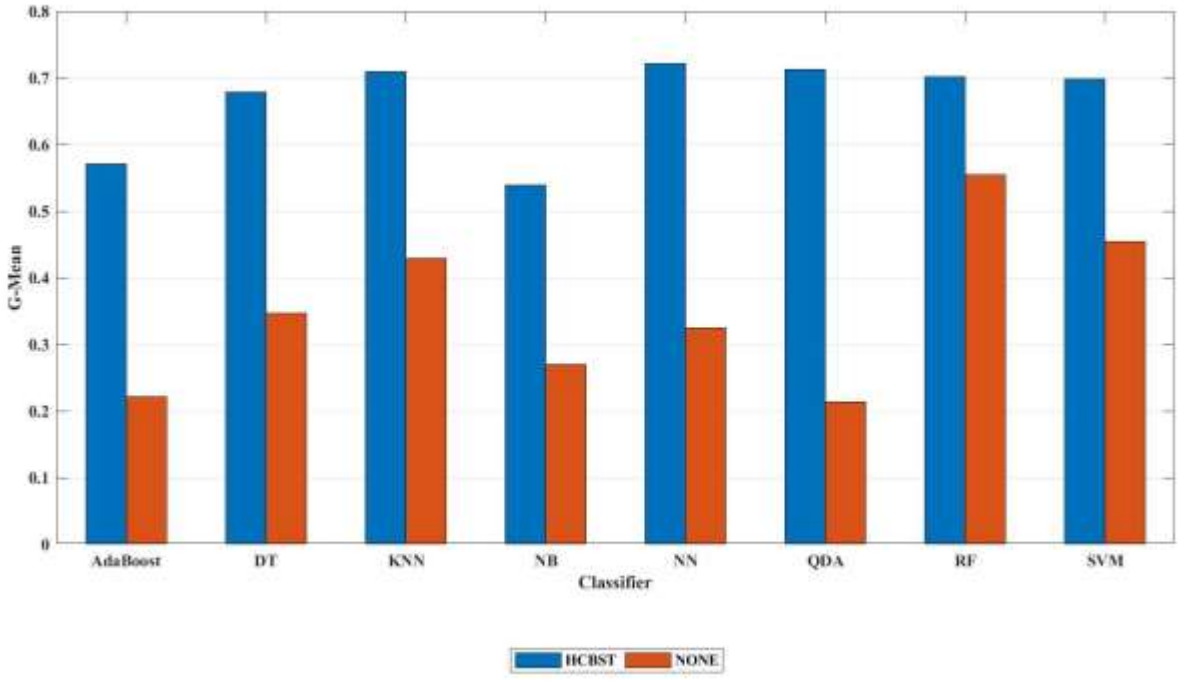


Figure 5.26 Average G-Mean Performance comparison of HCBST vs. NONE using all the datasets

5.7 Overall Average Performance of HCBST vs. NONE Using MCC

Figure 5.27 showed the average MCC performances over all the datasets when HCBST was used to resample the data versus the performance when no sampling was done. The results show that HCBST provided a higher average performance with respect to MCC in all the classifiers used in this study with a minimum improvement of 0.02 representing 5.62% with Decision Tree classifier and a maximum increase of 0.151 representing 75.4% with Random Forest Classifier.

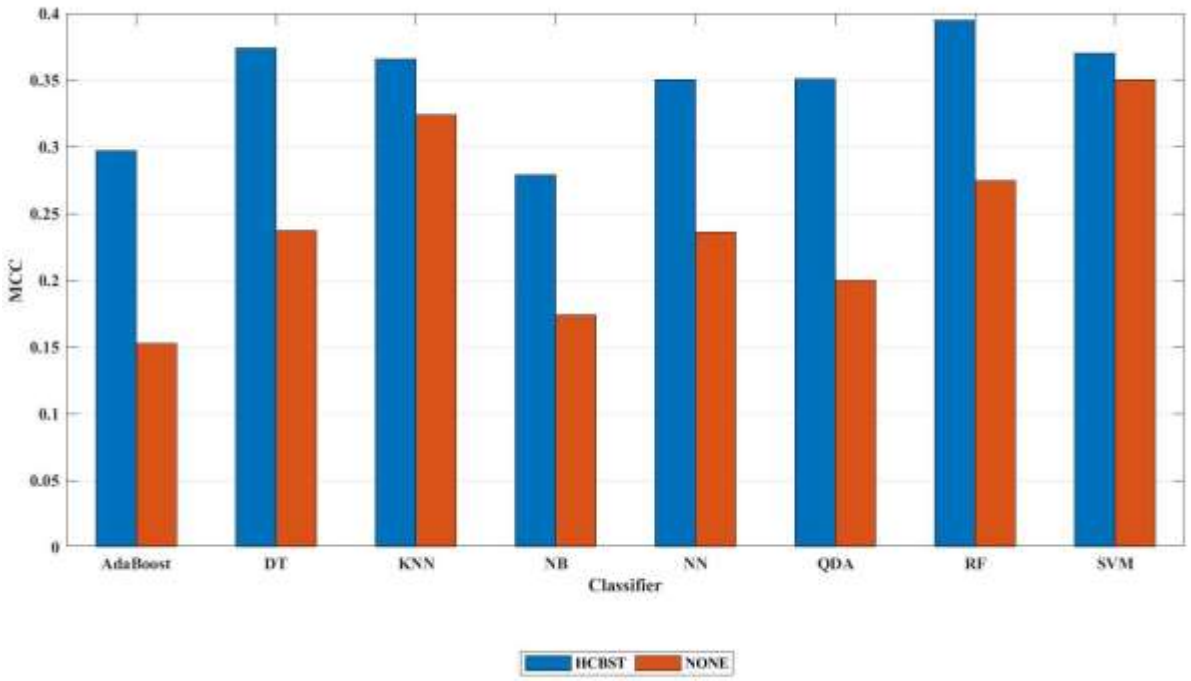


Figure 5.27 Average MCC Performance comparison of HCBST vs. NONE using all the datasets

5.8 HCBST vs ALL Using AUC

Figure 5.28 illustrates the performance of HCBST versus ALL in terms of AUC for Nearest Neighbours classifier. HCBST, NONE, RUS and SBC obtained a perfect AUC value of 1.0 for yeast-2_vs_8 dataset. Furthermore, HCBST, SBC had highest performance in 4 databases PC1, abalone19, glass2, yeast-2_vs_8 and KC3, MW1, ecoli4 and yeast-2_vs_8 respectively.

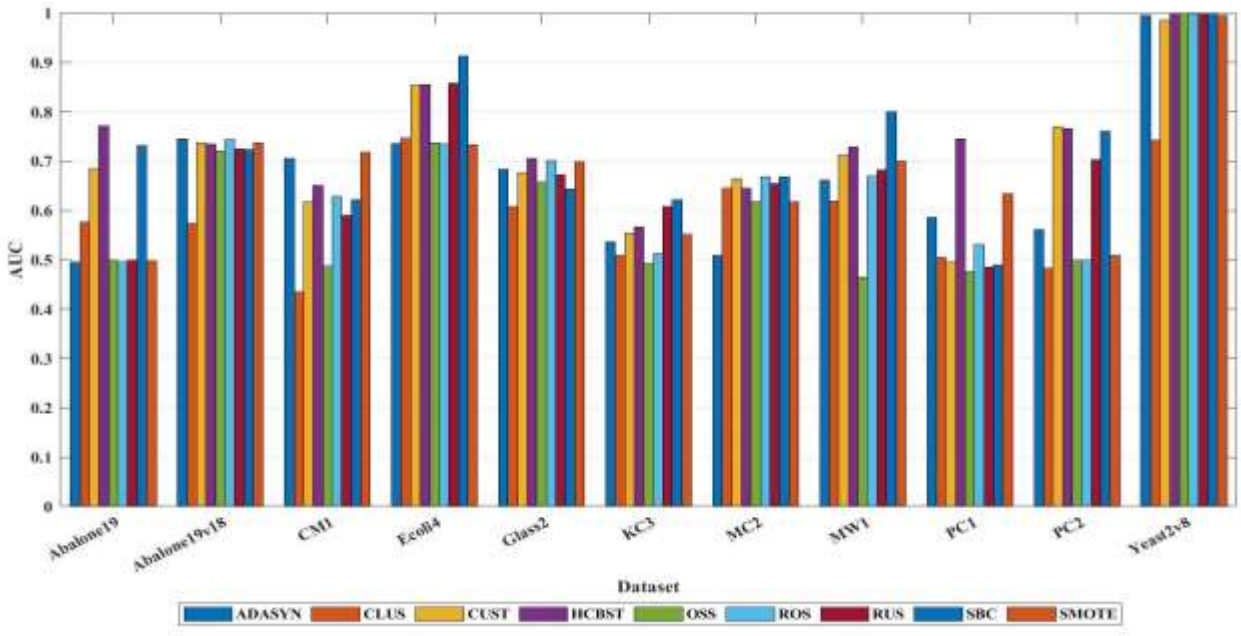


Figure 5.28 AUC Performance of HCBST vs All using KNN Classifier

Figure 5.29 illustrates the performance of HCBST versus ALL in terms of AUC for Linear SVM classifier. HCBST, NONE, ROS, RUS, SBC and SMOTE obtained a perfect AUC score of 1.0 for yeast-2_vs_8 dataset. Furthermore, HCBST had highest performance in 8 databases KC3, MC2, MW1, PC1, PC2, abalone19, ecoli4, yeast-2_vs_8 respectively indicating that HCBST dominated in the SVM classifier performance in terms of the AUC metric.

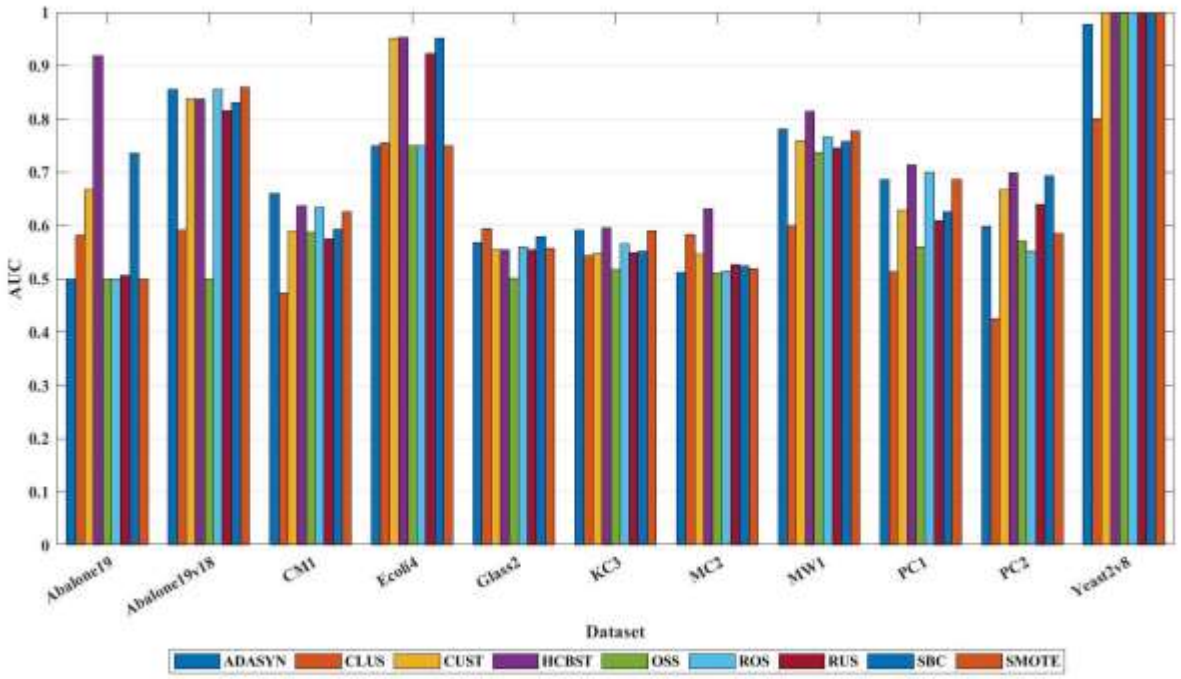


Figure 5.29 AUC Performance of HCBST vs All using SVM Classifier

Figure 5.30 illustrates the performance of HCBST versus ALL in terms of AUC for Decision Tree classifier. HCBST obtained the highest performance values of 0.674 and 0.781 for PC2 and abalone19 datasets respectively. Furthermore, SBC had the highest performance values of 0.722, 0.768 and 0.779 in 3 databases abalone9v18, ecoli4 and glass2 respectively. Again, the highest AUC values were recorded with the yeast-2_vs_8 database with a maximum value of 0. 977 by OSS.

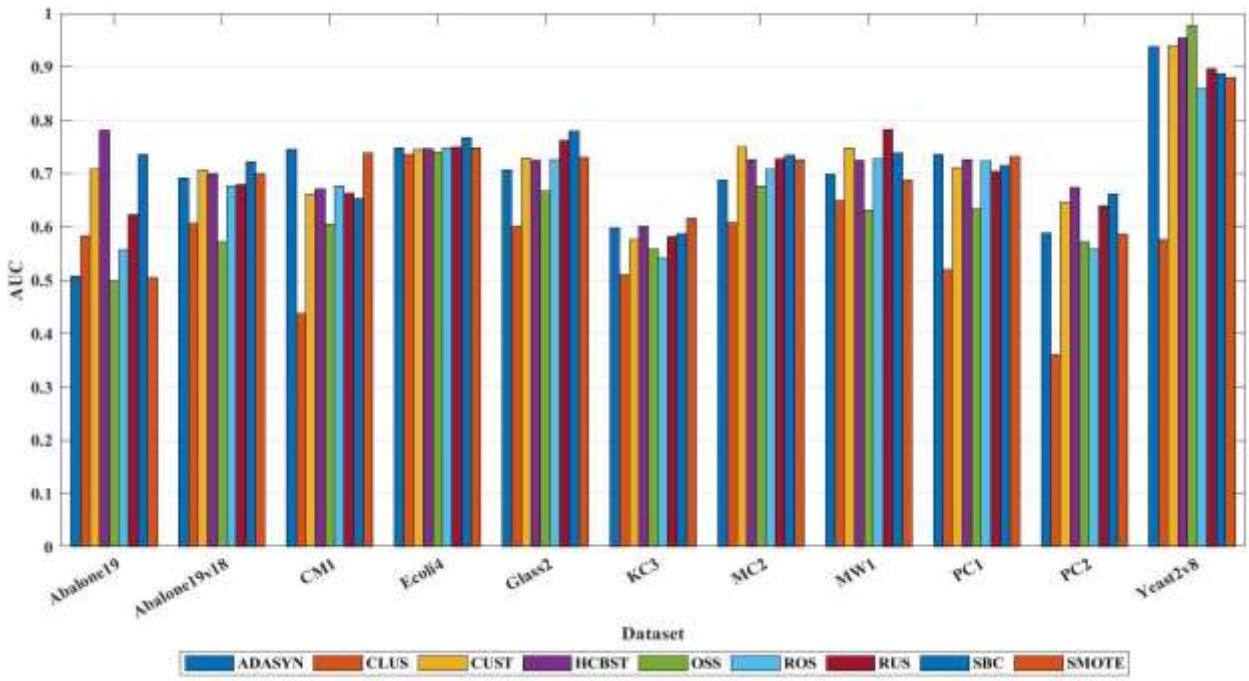


Figure 5.30 AUC Performance of HCBST vs All using Decision Tree Classifier

Figure 5.31 illustrates the performance of HCBST versus ALL in terms of AUC for Random Forest classifier. HCBST obtained the highest performance values of 0.749, 0.807 and 0.829 for PC1, abalone19 and yeast-2_vs_8 datasets respectively. Furthermore, SBC had highest performance values of 0.811, 0.718, 0.799 and 0.69 in MW1, abalone9v18, ecoli4 and glass2 respectively. The highest AUC values were recorded with the yeast-2_vs_8 database with a maximum value of 0. 829 by HCBST.

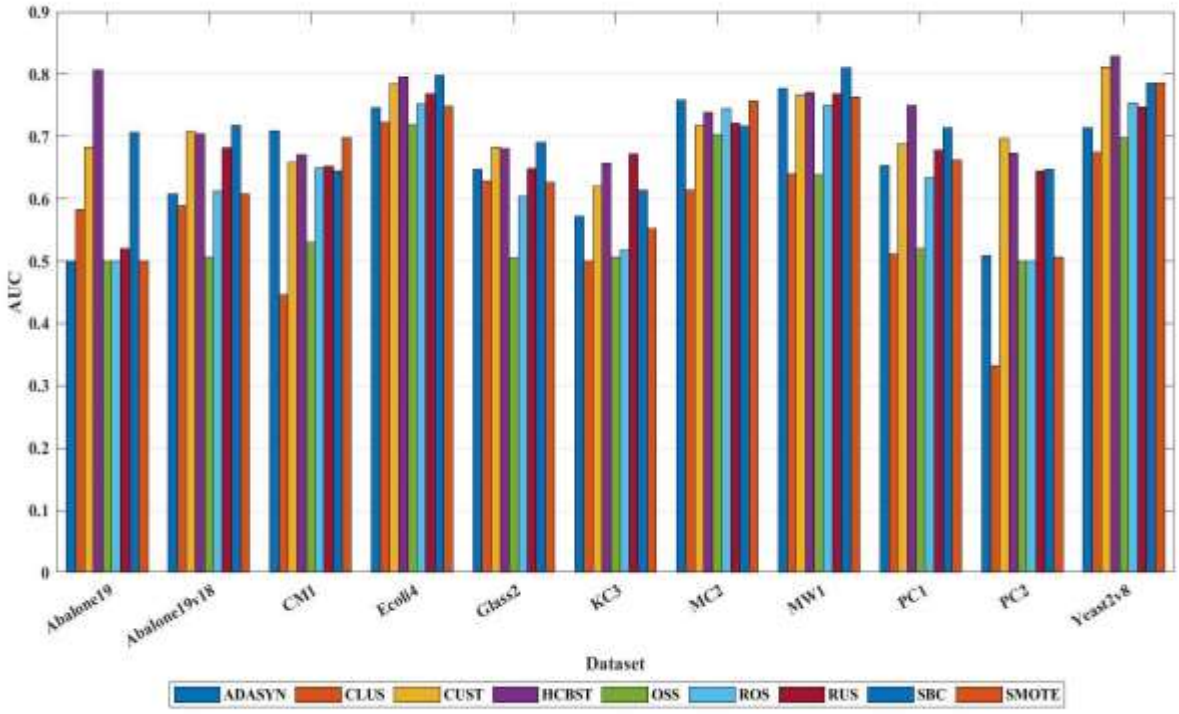


Figure 5.31 AUC Performance of HCBST vs All using Random Forest Classifier

Figure 5.32 illustrates the performance of HCBST versus ALL in terms of AUC for Neural Net classifier. HCBST, obtained the highest performance values of 0.645, 0.877 and 1.0 for MC2, abalone19 and yeast-2_v_8 datasets respectively. Furthermore, ROS had highest performance values of 0.575, 0.565, 0.566 and 1.0 in CM1, KC3, PC2 and yeast-2_vs_8 respectively. The highest AUC values were recorded with the yeast-2_vs_8 database with a maximum value of 1.0 by HCBST, SMOTE, SBC, ROS, OSS and CUST.

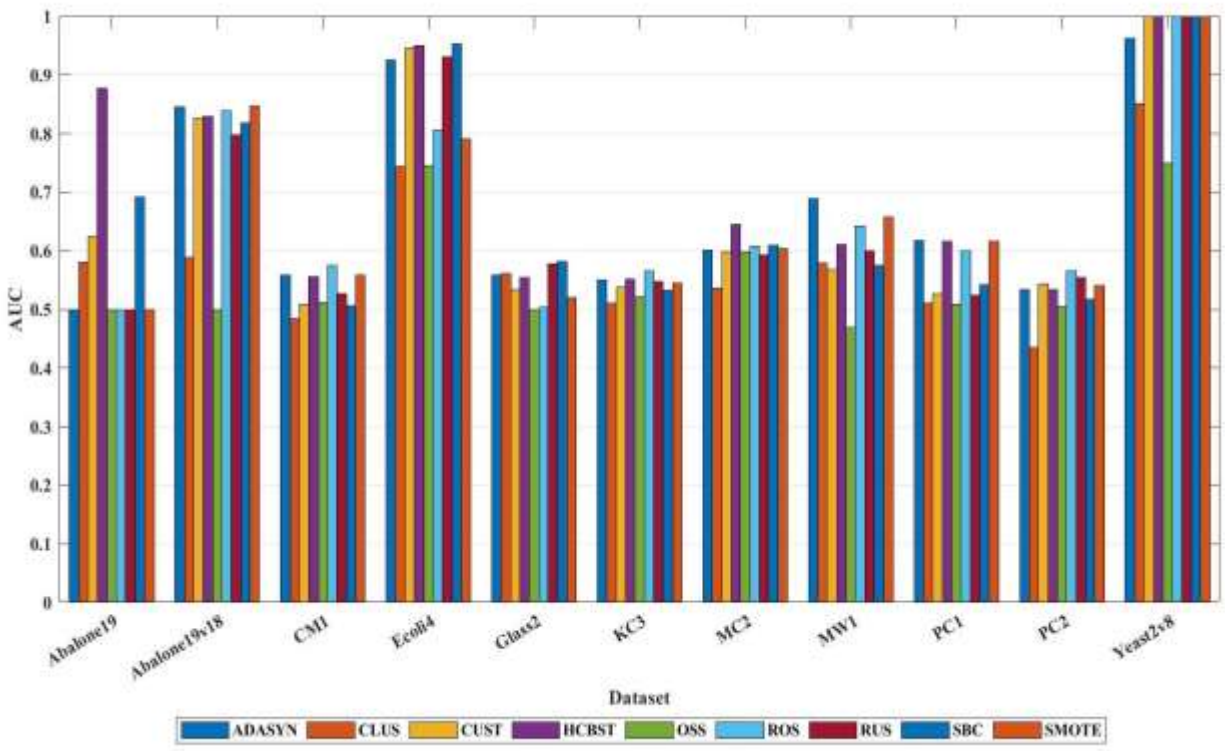


Figure 5.32 AUC Performance of HCBST vs All using Neural Network Classifier

Figure 5.33 illustrates the performance of HCBST versus ALL in terms of AUC for AdaBoost classifier. SMOTE, HCBST, SMOTE, RUS, HCBST, HCBST, HCBST, ROS, SBC, SMOTE, CUST, obtained the highest performance values; 0.736, 0.631, 0.637, 0.772, 0.759, 0.75, 0.795, 0.721, 0.804, 0.856, 0.884, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST had the highest performance in 4 databases KC3, PC1, PC2, abalone19respectively.

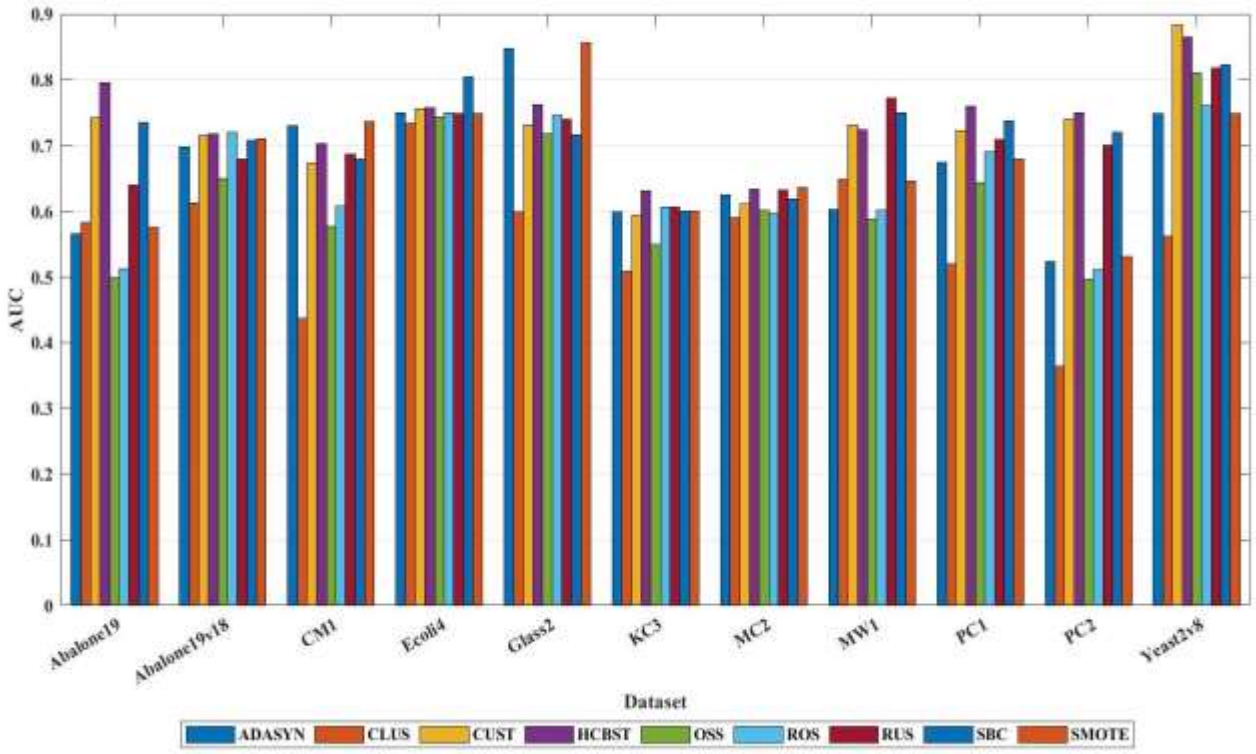


Figure 5.33 AUC Performance of HCBST vs All using AdaBoost Classifier

Figure 5.34 illustrates the performance of HCBST versus ALL in terms of AUC for Naive Bayes classifier. ADASYN, SBC, CUST, SBC, HCBST, CUST, ROS, HCBST, RUS, SMOTE, HCBST, CUST, obtained the highest performance values; 0.683, 0.746, 0.7, 0.7, 0.84, 0.643, 0.727, 0.735, 0.77, 0.935, 0.716, 0.994, for CM1, KC3, MC2, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, CUST and HCBST had highest performance in 3 databases MC2, PC1, yeast-2_vs_8 and MW1, abalone19, glass2 respectively.

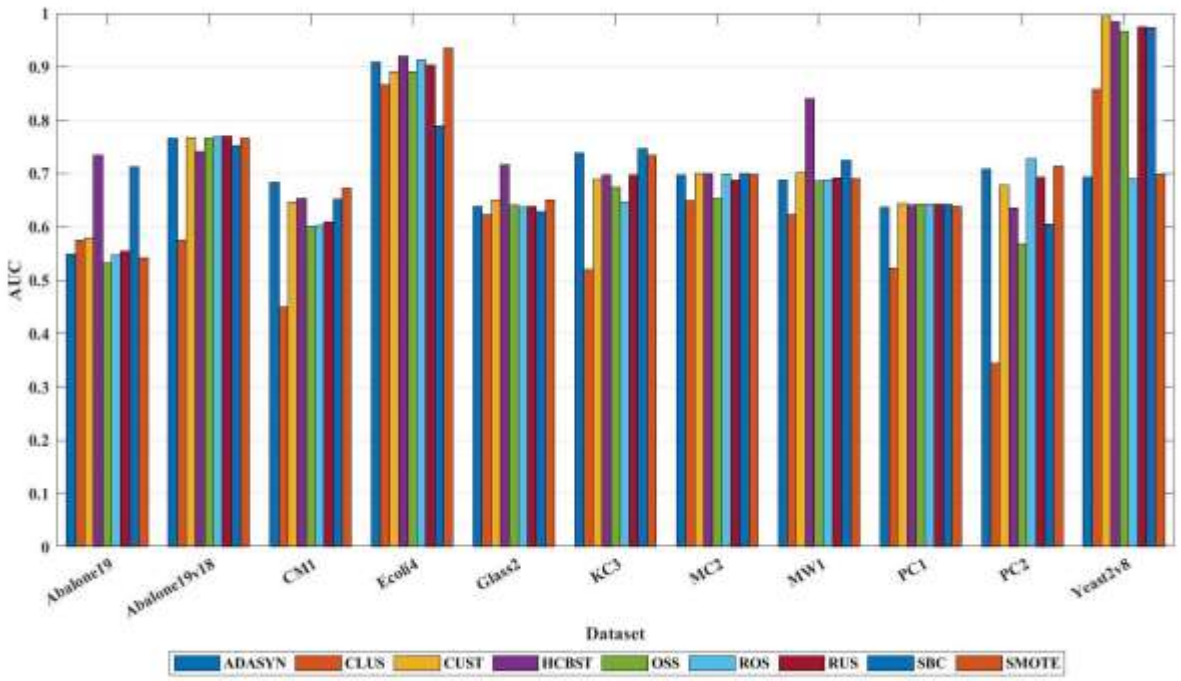


Figure 5.34 AUC Performance of HCBST vs All using Naïve Bayes Classifier

Figure 5.35 illustrates the performance of HCBST versus ALL in terms of AUC for QDA classifier. RUS, RUS, RUS, RUS, HCBST, SBC, SBC, ADASYN, SBC, SBC, HCBST, obtained the highest performance values; 0.612, 0.524, 0.622, 0.598, 0.692, 0.598, 0.777, 0.863, 0.706, 0.897, 0.811, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2 and yeast-2_vs_8 datasets respectively. Furthermore, RUS, SBC had highest performance in 4 databases CM1, KC3, MC2, MW1and PC2, abalone19, ecoli4, glass2 respectively.

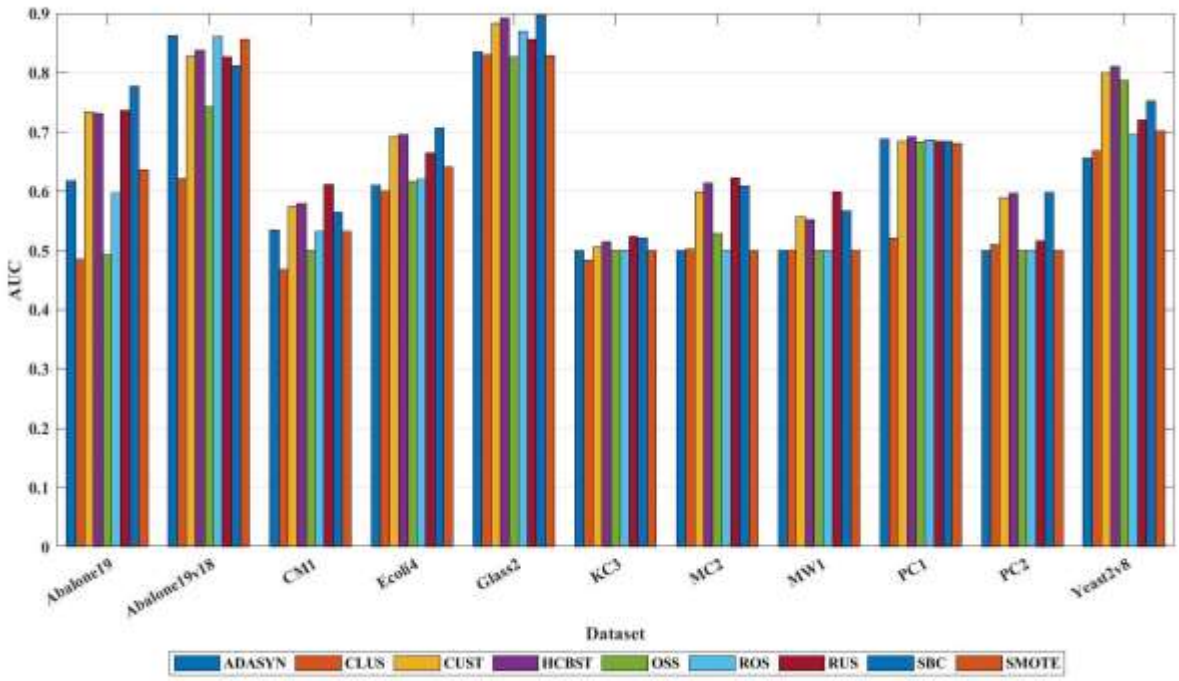


Figure 5.35 AUC Performance of HCBST vs All using QDA Classifier

5.9 HCBST vs ALL Using G-Mean

Figure 5.36 illustrates the performance of HCBST versus ALL in terms of G-Mean for Nearest Neighbours classifier. SMOTE, SBC, ROS, SBC, HCBST, CUST, HCBST, HCBST, SBC, ROS, HCBST, NONE, RUS, SBC, obtained the highest performance values; 0.71, 0.595, 0.659, 0.789, 0.725, 0.739, 0.756, 0.723, 0.908, 0.672, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8 datasets respectively. Furthermore, HCBST, SBC had highest performance in 4 databases PC1, abalone19, abalone9v18, yeast-2_vs_8 and KC3, MW1, ecoli4, yeast-2_vs_8 respectively.

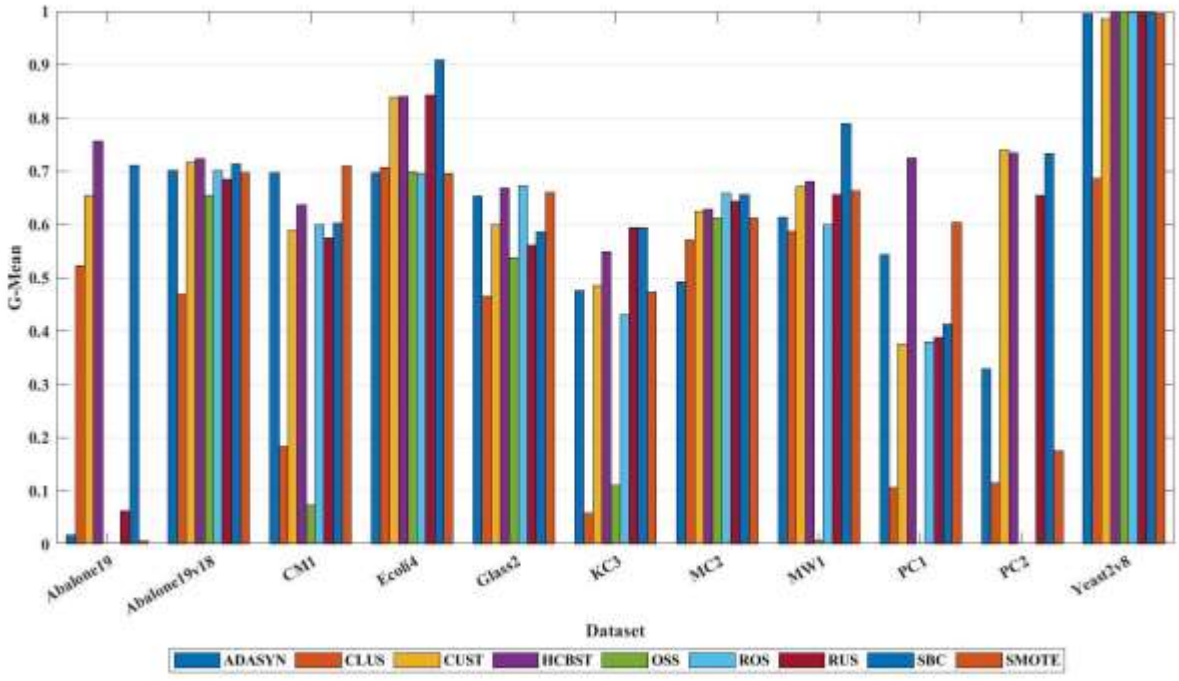


Figure 5.36 G-Mean Performance of HCBST vs All using KNN Classifier

Figure 5.37 illustrates the performance of HCBST versus ALL in terms of G-Mean for Linear SVM classifier. ADASYN, HCBST, HCBST, HCBST, HCBST, HCBST, HCBST, SMOTE, HCBST, CLUS, CUST, HCBST, NONE, ROS, RUS, SBC, SMOTE, obtained the highest performance values; 0.529, 0.405, 0.492, 0.789, 0.643, 0.626, 0.915, 0.852, 0.95, 0.445, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, datasets respectively. Furthermore, HCBST had highest performance in 8 databases KC3, MC2, MW1, PC1, PC2, abalone19, ecoli4 and yeast-2_vs_8 respectively.

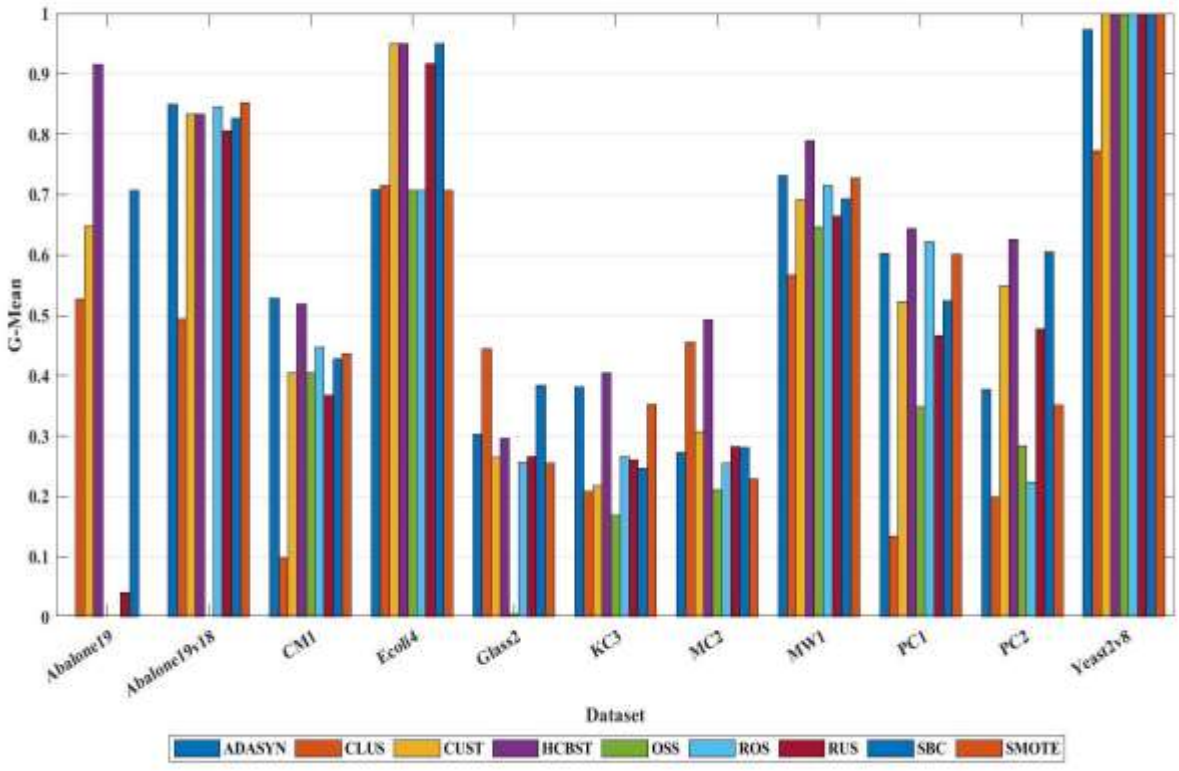


Figure 5.37 G-Mean Performance of HCBST vs All using SVM Classifier

Figure 5.38 illustrates the performance of HCBST versus ALL in terms of G-Mean for Decision Tree classifier. ADASYN, HCBST, HCBST, RUS, ADASYN, HCBST, HCBST, SBC, SBC, SBC, OSS, obtained the highest performance values; 0.739, 0.504, 0.714, 0.762, 0.729, 0.593, 0.767, 0.711, 0.739, 0.714, 0.973, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2 and yeast-2_vs_8 datasets respectively. Furthermore, HCBST had highest performance in 4 databases KC3, MC2, PC2 and abalone19 respectively.

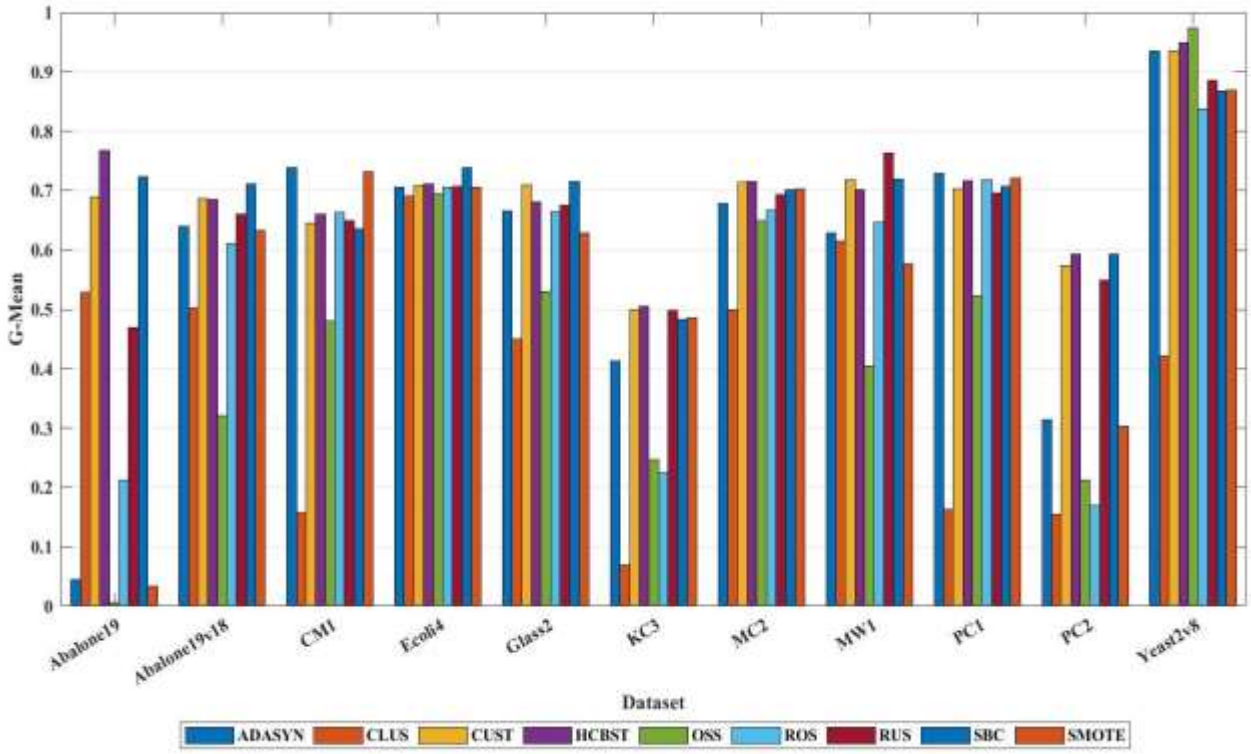


Figure 5.38 G-Mean Performance of HCBST vs All using Decision Tree Classifier

Figure 5.39 illustrates the performance of HCBST versus ALL in terms of G-Mean for Random Forest classifier. ADASYN, RUS, ADASYN, SBC, HCBST, CUST, HCBST, SBC, SBC, CUST, HCBST, obtained the highest performance values; 0.702, 0.614, 0.737, 0.797, 0.732, 0.658, 0.798, 0.71, 0.78, 0.653, 0.801, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST and SBC had highest performance in 3 databases PC1, abalone19, yeast-2_vs_8 and MW1, abalone9v18 and ecoli4 respectively.

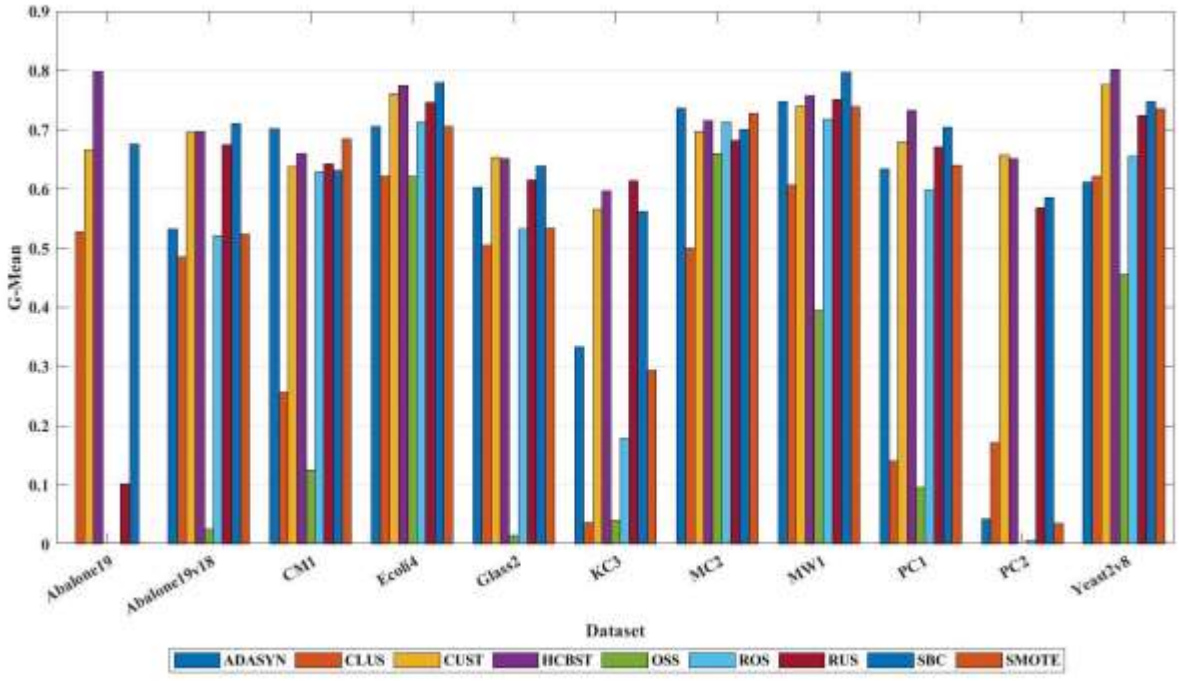


Figure 5.39 G-Mean Performance of HCBST vs All using Random Forest Classifier

Figure 5.40 illustrates the performance of HCBST versus ALL in terms of G-Mean for Neural Net classifier. ROS, ROS, HCBST, ADASYN, HCBST, RUS, HCBST, SMOTE, SBC, ADASYN, CUST, HCBST, ROS, RUS, SBC, SMOTE, obtained the highest performance values; 0.375, 0.332, 0.506, 0.553, 0.5, 0.321, 0.869, 0.841, 0.95, 0.472, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8 datasets respectively. Furthermore, HCBST had highest performance in 4 databases MC2, PC1, abalone19 and yeast-2_vs_8 respectively.

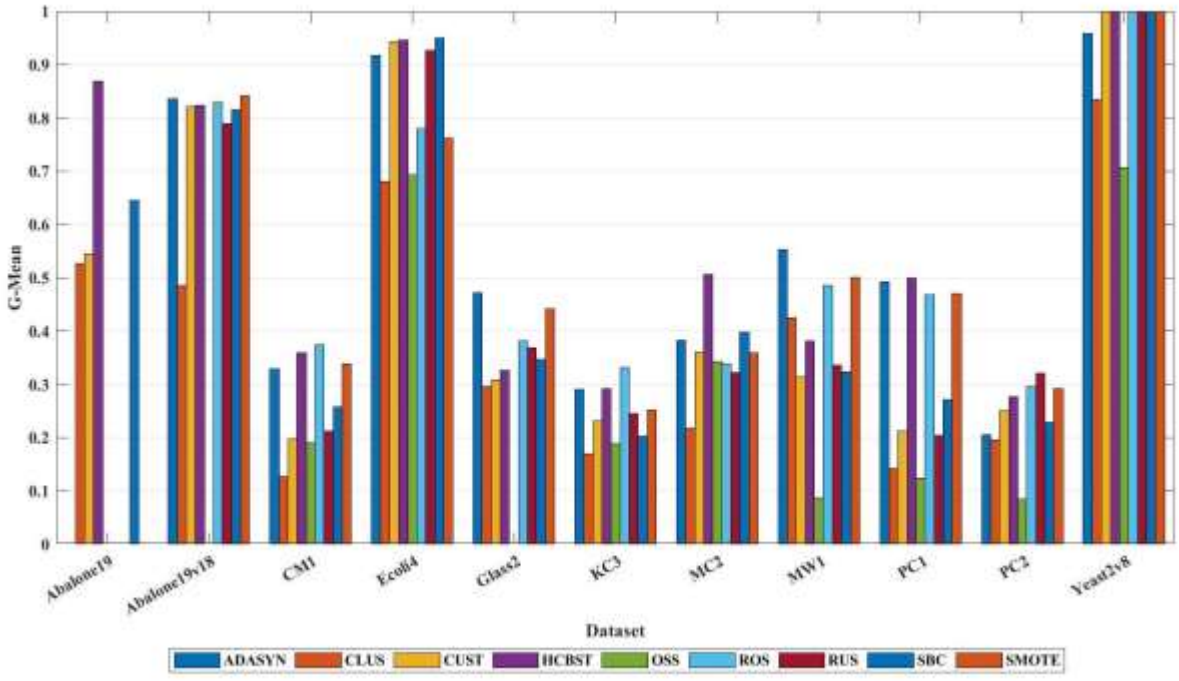


Figure 5.40 G-Mean Performance of HCBST vs All using Neural Network Classifier

Figure 5.41 illustrates the performance of HCBST versus ALL in terms of G-Mean for AdaBoost classifier. SMOTE, RUS, HCBST, SBC, HCBST, HCBST, HCBST, HCBST, SBC, SMOTE, CUST, obtained the highest performance values; 0.722, 0.575, 0.626, 0.736, 0.753, 0.711, 0.783, 0.708, 0.786, 0.83, 0.871, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST had highest performance in 5 databases MC2, PC1, PC2, abalone19 and abalone9v18 respectively.

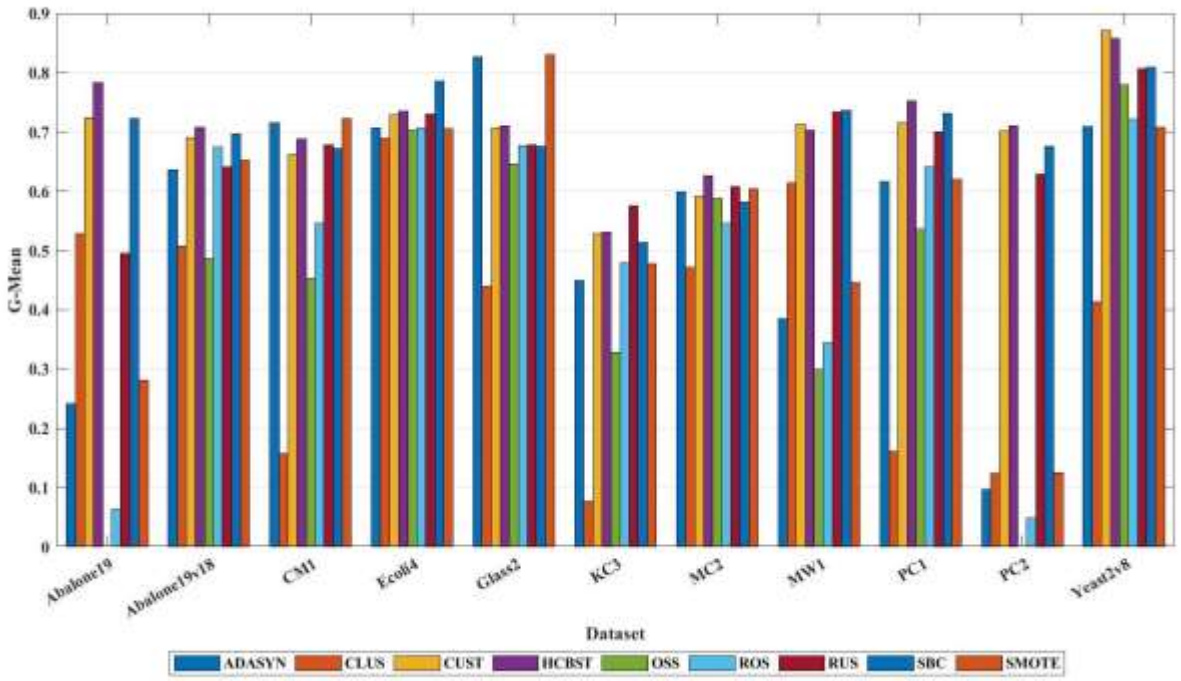


Figure 5.41 G-Mean Performance of HCBST vs All using AdaBoost Classifier

Figure 5.42 illustrates the performance of HCBST versus ALL in terms of G-Mean for Naive Bayes classifier. ADASYN, SBC, CUST, SBC, HCBST, HCBST, ROS, HCBST, RUS, SMOTE, HCBST, CUST, obtained the highest performance values; 0.641, 0.704, 0.632, 0.632, 0.829, 0.563, 0.693, 0.706, 0.77, 0.93, 0.665, 0.994, for CM1, KC3, MC2, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST had highest performance in 4 databases MW1, PC1, abalone19 and glass2 respectively.

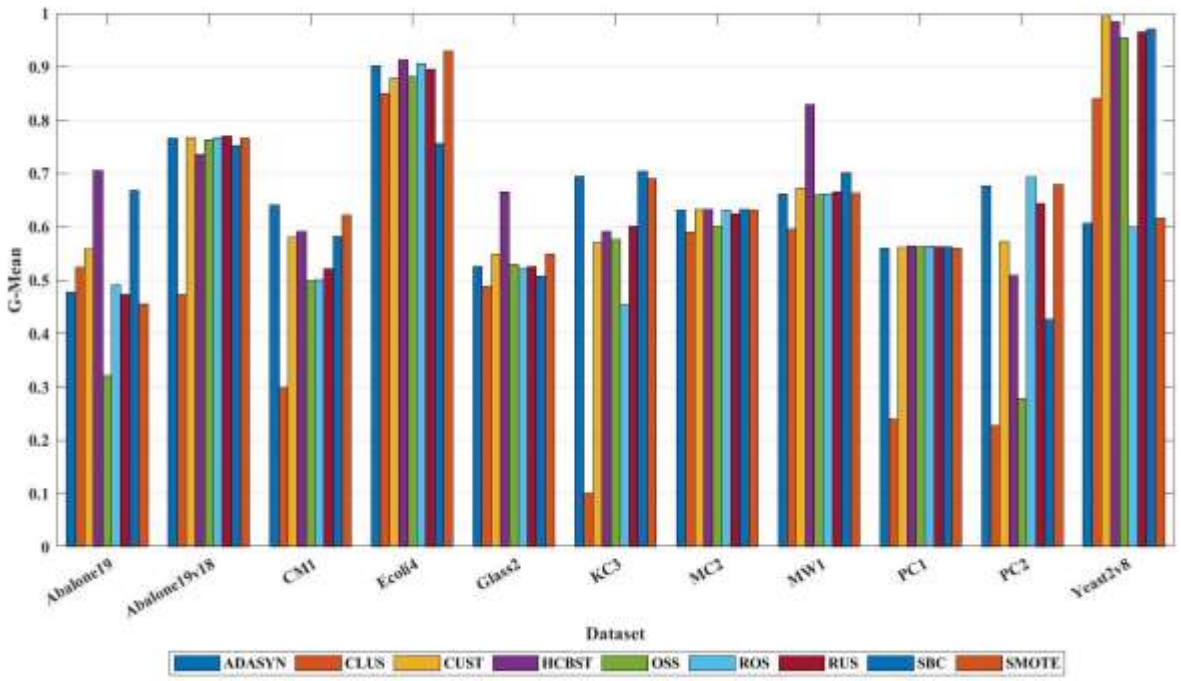


Figure 5.42 G-Mean Performance of HCBST vs All using Naïve Bayes Classifier

Figure 5.43 illustrates the performance of HCBST versus ALL in terms of G-Mean for QDA classifier. RUS, RUS, RUS, RUS, HCBST, SBC, SBC, ADASYN, SBC, SBC, HCBST, obtained the highest performance values; 0.504, 0.363, 0.567, 0.344, 0.679, 0.43, 0.764, 0.855, 0.562, 0.85, 0.726, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, RUS, SBC had highest performance in 4 databases CM1, KC3, MC2, MW1and PC2, abalone19, ecoli4 and glass2 respectively.

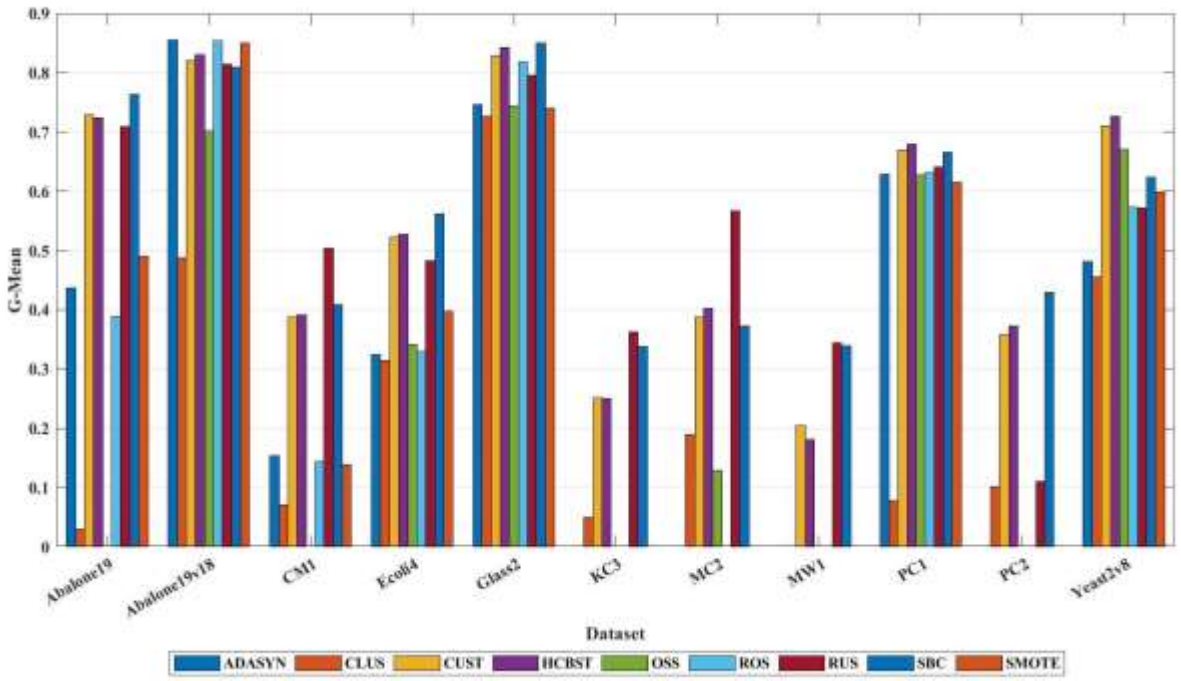


Figure 5.43 G-Mean Performance of HCBST vs All using QDA Classifier

5.10 HCBST vs ALL Using MCC

Figure 5.44 illustrates the performance of HCBST versus ALL in terms of MCC for Nearest Neighbours classifier. SMOTE, SBC, CUST, SBC, HCBST, CUST, HCBST, ADASYN, CUST, HCBST, HCBST, NONE, RUS, SBC, obtained the highest performance values; 0.305, 0.215, 0.452, 0.364, 0.28, 0.211, 0.144, 0.653, 0.594, 0.395, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, datasets respectively. Furthermore, HCBST had highest performance in 4 databases PC1, abalone19, glass2 and yeast-2_vs_8 respectively.

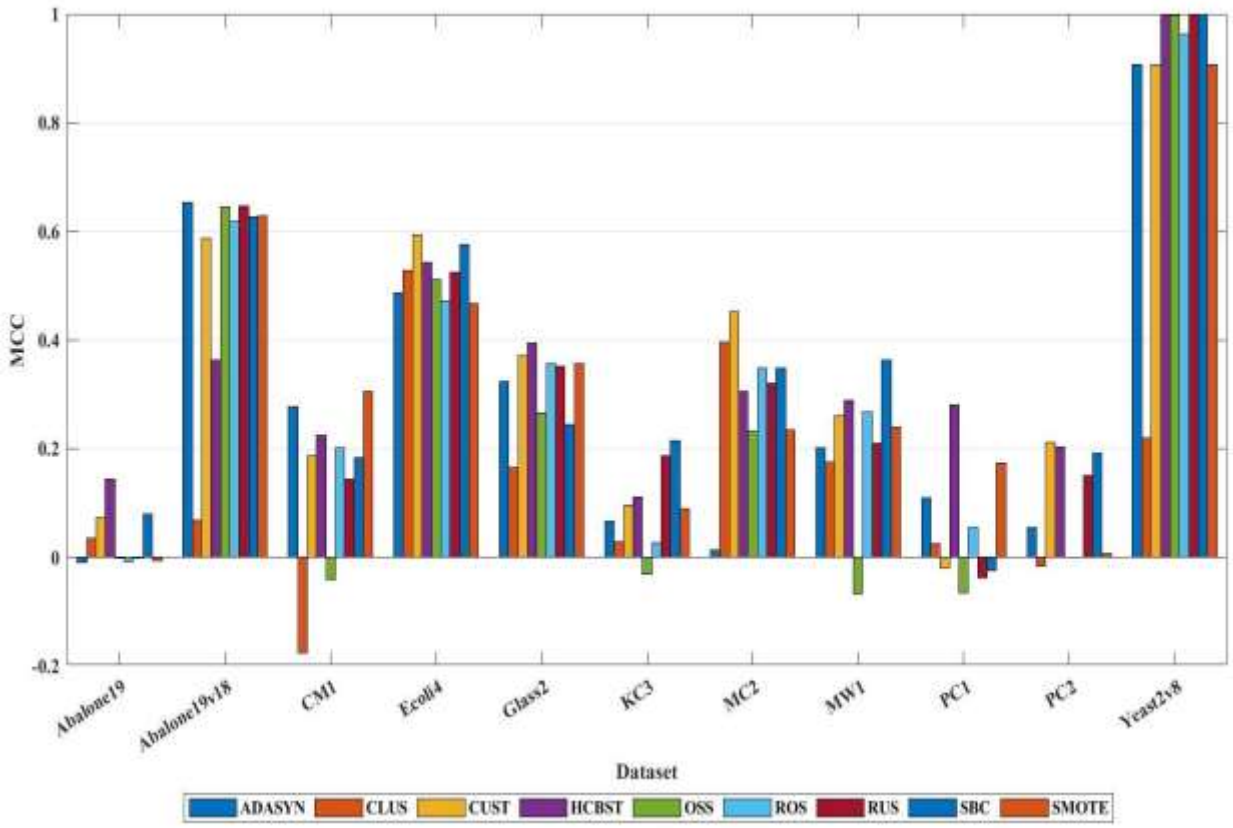


Figure 5.44 MCC Performance of HCBST vs All using KNN Classifier

Figure 5.45 illustrates the performance of HCBST versus ALL in terms of MCC for Linear SVM classifier. ADASYN, HCBST, CLUS, CUST, ROS, SBC, HCBST, CUST, ROS, RUS, SMOTE, ADASYN, CUST, HCBST, NONE, OSS, ROS, RUS, SMOTE, CLUS, CUST, HCBST, NONE, ROS, RUS, SBC, SMOTE, obtained the highest performance values; 0.211, 0.193, 0.291, 0.615, 0.295, 0.159, 0.212, 0.697, 0.697, 0.697, 0.697, 0.696, 0.696, 0.696, 0.696, 0.696, 0.696, 0.696, 0.696, 0.141, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, abalone9v18, abalone9v18, abalone9v18, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8 datasets respectively. Furthermore, CUST, HCBST and ROS had highest performance in 4 databases MW1, abalone9v18, ecoli4, yeast-2_vs_8 and KC3, abalone19, ecoli4, yeast-2_vs_8 and PC1, abalone9v18, ecoli4, yeast-2_vs_8 respectively.

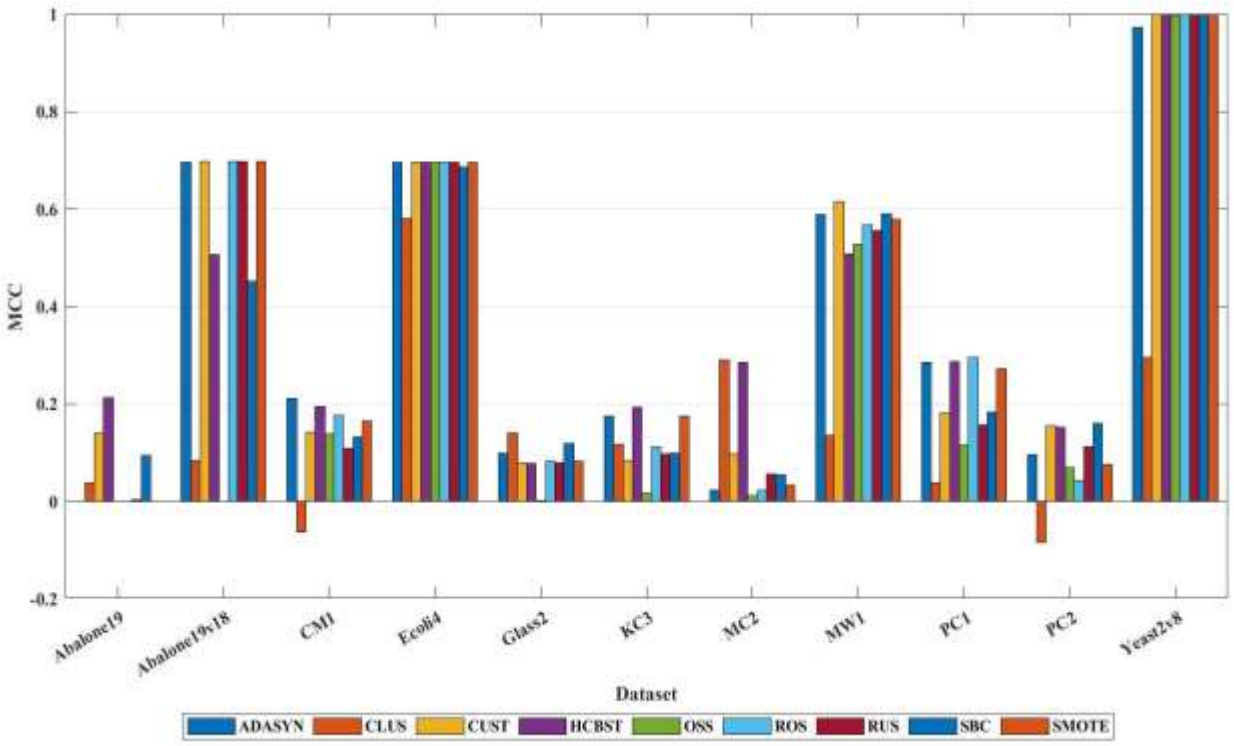


Figure 5.45 MCC Performance of HCBST vs All using SVM Classifier

Figure 5.46 illustrates the performance of HCBST versus ALL in terms of MCC for Decision Tree classifier. ADASYN, SMOTE, CUST, RUS, RUS, RUS, HCBST, SMOTE, ROS, SBC, OSS, obtained the highest performance values; 0.358, 0.404, 0.575, 0.434, 0.382, 0.212, 0.12, 0.322, 0.689, 0.475, 0.958, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2 and yeast-2_vs_8 datasets respectively. Furthermore, RUS had highest performance in 3 databases MW1, PC1 and PC2 respectively.

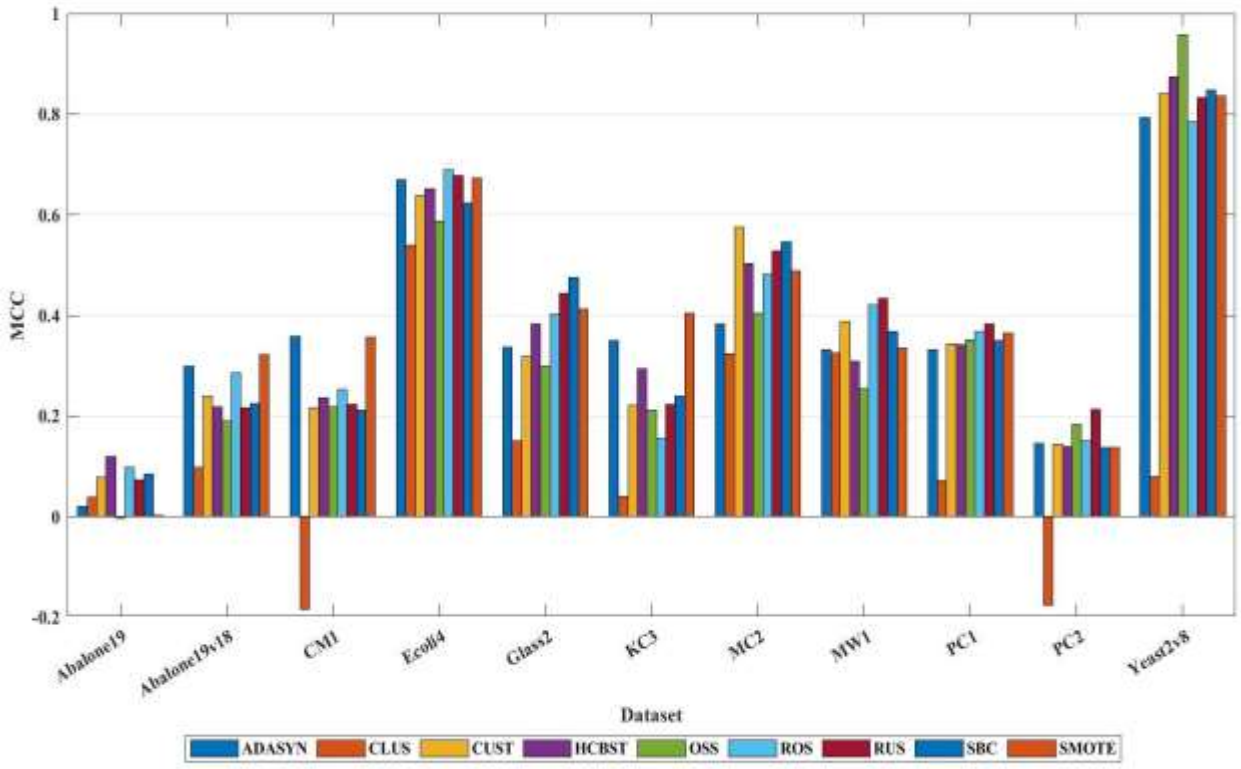


Figure 5.46 MCC Performance of HCBST vs All using Decision Tree Classifier

Figure 5.47 illustrates the performance of HCBST versus ALL in terms of MCC for Random Forest classifier. SMOTE, RUS, SMOTE, SMOTE, HCBST, CUST, HCBST, SMOTE, HCBST, ROS, ADASYN, SMOTE, obtained the highest performance values; 0.331, 0.314, 0.585, 0.558, 0.29, 0.148, 0.176, 0.222, 0.687, 0.687, 0.254, 0.675, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, ecoli4, glass2 and yeast-2_vs_8 datasets respectively. Furthermore, SMOTE had highest performance in 5 databases CM1, MC2, MW1, abalone9v18 and yeast-2_vs_8 respectively.

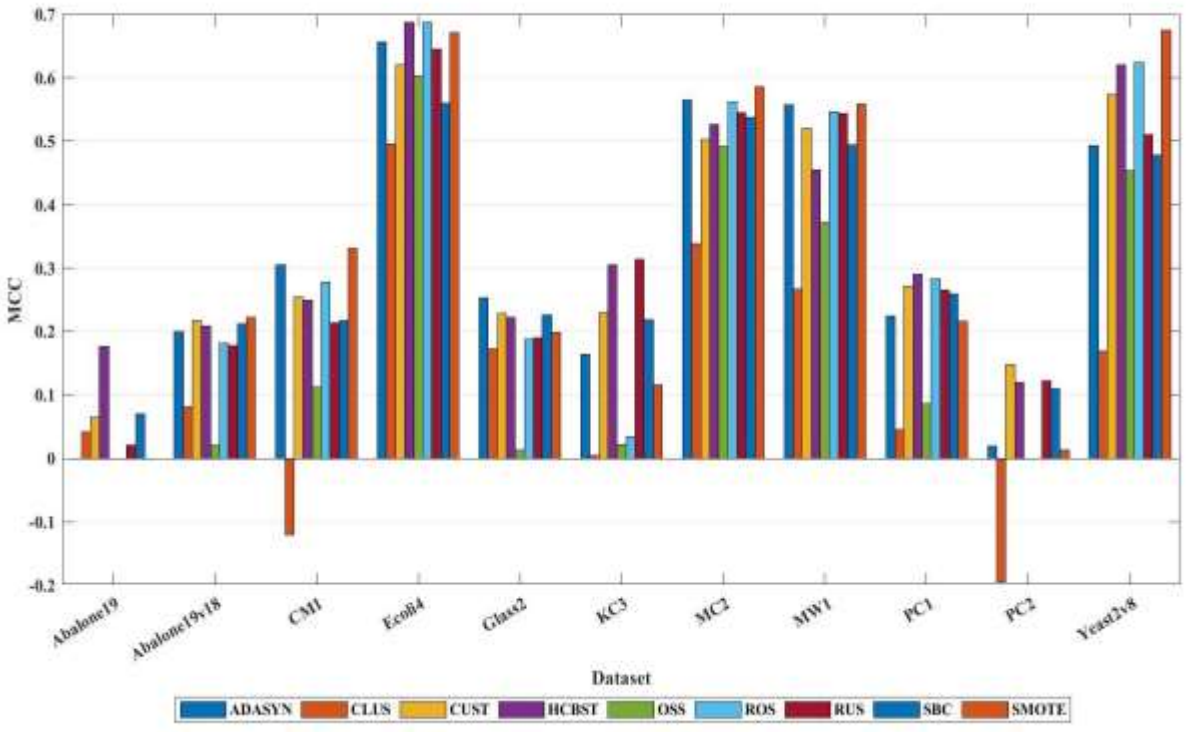


Figure 5.47 MCC Performance of HCBST vs. All using Random Forest Classifier

Figure 5.48 illustrates the performance of HCBST versus ALL in terms of MCC for Neural Net classifier. ROS, ROS, HCBST, ADASYN, HCBST, ROS, HCBST, SMOTE, ADASYN, CUST, HCBST, ROS, RUS, SMOTE, SBC, CUST, HCBST, ROS, RUS, SBC, SMOTE, obtained the highest performance values; 0.116, 0.134, 0.318, 0.253, 0.178, 0.063, 0.162, 0.761, 0.696, 0.696, 0.696, 0.696, 0.696, 0.696, 0.117, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, ecoli4, glass2, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, yeast-2_vs_8, and yeast-2_vs_8 datasets respectively. Furthermore, HCBST and ROS had highest performance in 5 databases MC2, PC1, abalone19, ecoli4, yeast-2_vs_8 and CM1, KC3, PC2, ecoli4 and yeast-2_vs_8respectively.

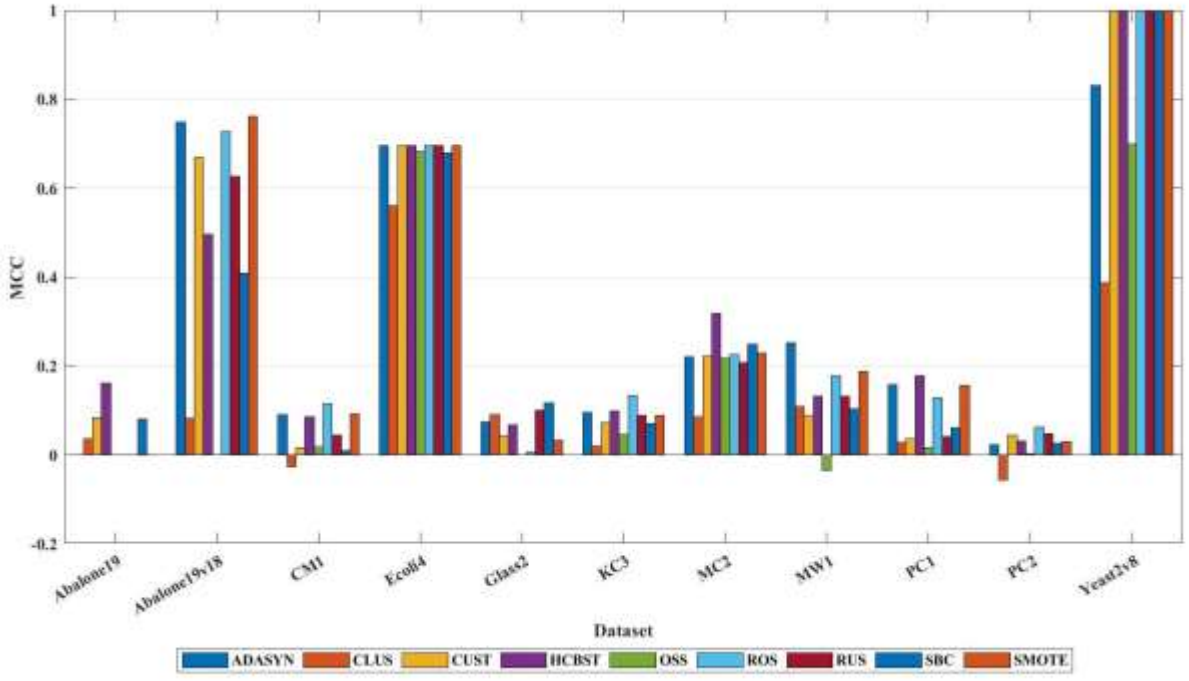


Figure 5.48 MCC Performance of HCBST vs. All using Neural Network Classifier

Figure 5.49 illustrates the performance of HCBST versus ALL in terms of MCC for AdaBoost classifier. SMOTE, HCBST, CLUS, RUS, SMOTE, HCBST, HCBST, ADASYN, ADASYN, ROS, SMOTE, OSS, obtained the highest performance values; 0.384, 0.312, 0.291, 0.388, 0.435, 0.184, 0.164, 0.469, 0.685, 0.685, 0.594, 0.69, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST and SMOTE had highest performance in 3 databases KC3, PC2, abalone19 and CM1, PC1 and glass2 respectively.

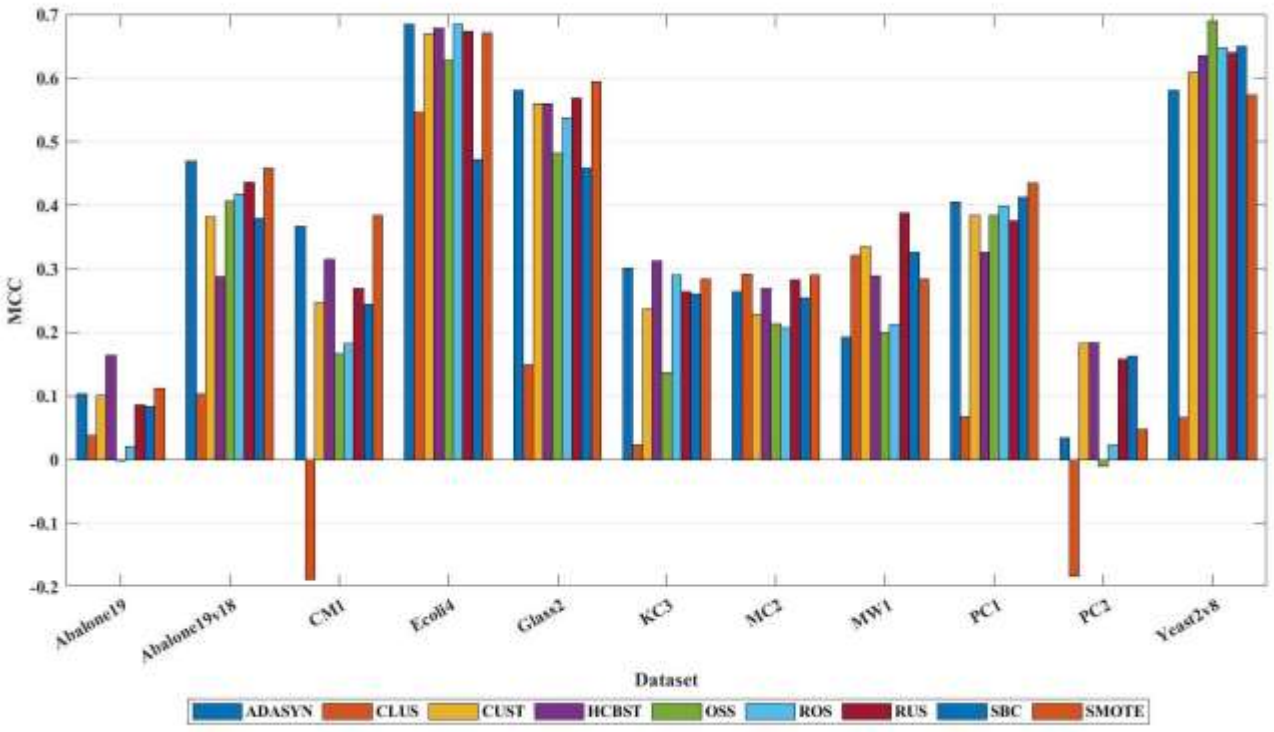


Figure 5.49 MCC Performance of HCBST vs. All using Adaboost Classifier

Figure 5.50 illustrates the performance of HCBST versus ALL in terms of MCC for Naive Bayes classifier. ADASYN, SBC, CUST, SBC, HCBST, SBC, ROS, HCBST, RUS, SMOTE, HCBST, CUST, obtained the highest performance values; 0.352, 0.644, 0.539, 0.539, 0.466, 0.328, 0.318, 0.082, 0.286, 0.623, 0.254, 0.934, for CM1, KC3, MC2, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8, datasets respectively. Furthermore, HCBST and SBC had highest performance in 3 databases MW1, abalone19, glass2 and KC3, MC2 and PC1 respectively.

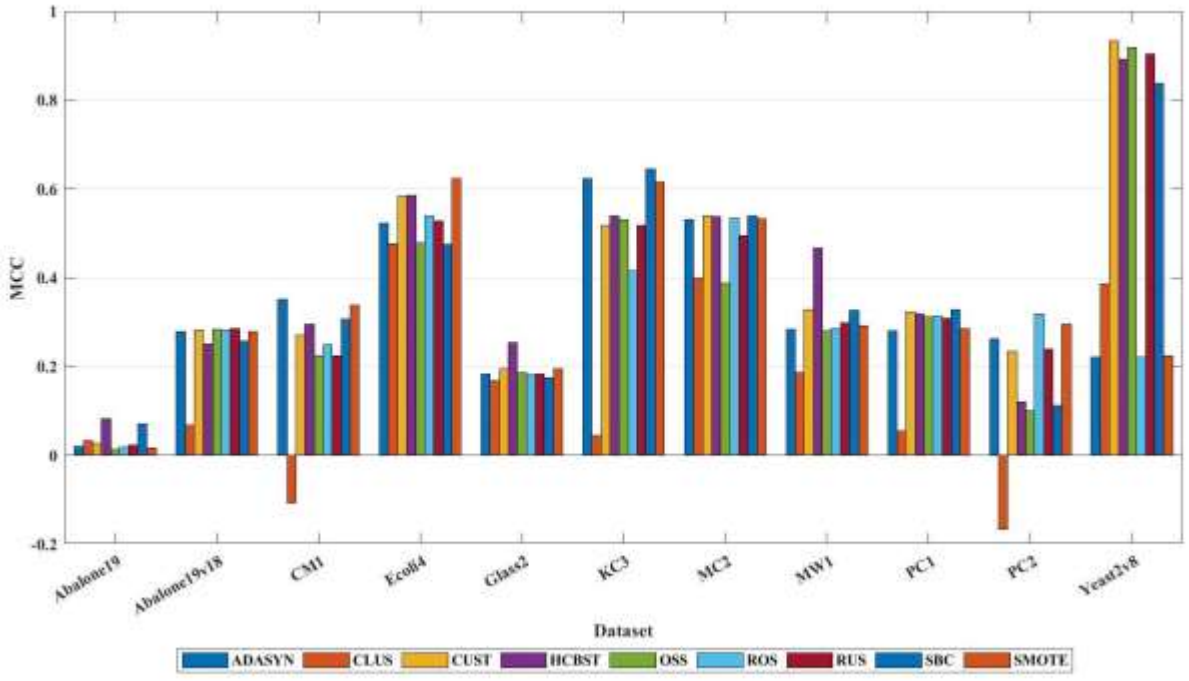


Figure 5.50 MCC Performance of HCBST vs. All using Naïve Bayes Classifier

Figure 5.51 illustrates the performance of HCBST versus ALL in terms of MCC for QDA classifier. RUS, SBC, SBC, RUS, ROS, SBC, HCBST, ADASYN, SBC, HCBST, HCBST, obtained the highest performance values; 0.158, 0.046, 0.299, 0.269, 0.386, 0.082, 0.159, 0.687, 0.301, 0.624, 0.436, for CM1, KC3, MC2, MW1, PC1, PC2, abalone19, abalone9v18, ecoli4, glass2, yeast-2_vs_8 datasets respectively. Furthermore, SBC had highest performance in 4 databases KC3, MC2, PC2 and ecoli4 respectively.

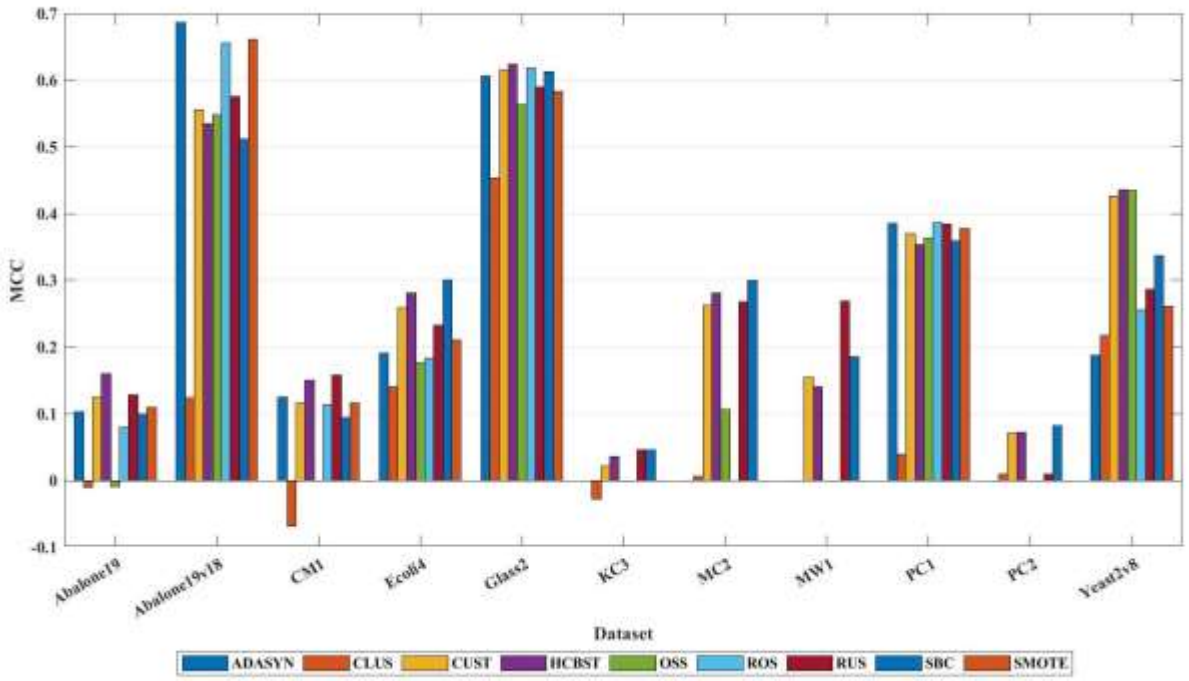


Figure 5.51 MCC Performance of HCBST vs. All using QDA Classifier

5.11 Overall Average Performance of HCBST vs. ALL Using AUC

Figure 5.52 showed the average AUC performances over all the datasets when HCBST was used to resample the data versus the performance vs. the other sampling techniques considered for this study. The results show that HCBST provided a higher average performance of 0.73 concerning AUC in all the classifiers used in this study with a minimum performance of 0.68 with QDA classifier and maximum performance of 0.76 with SVM classifier

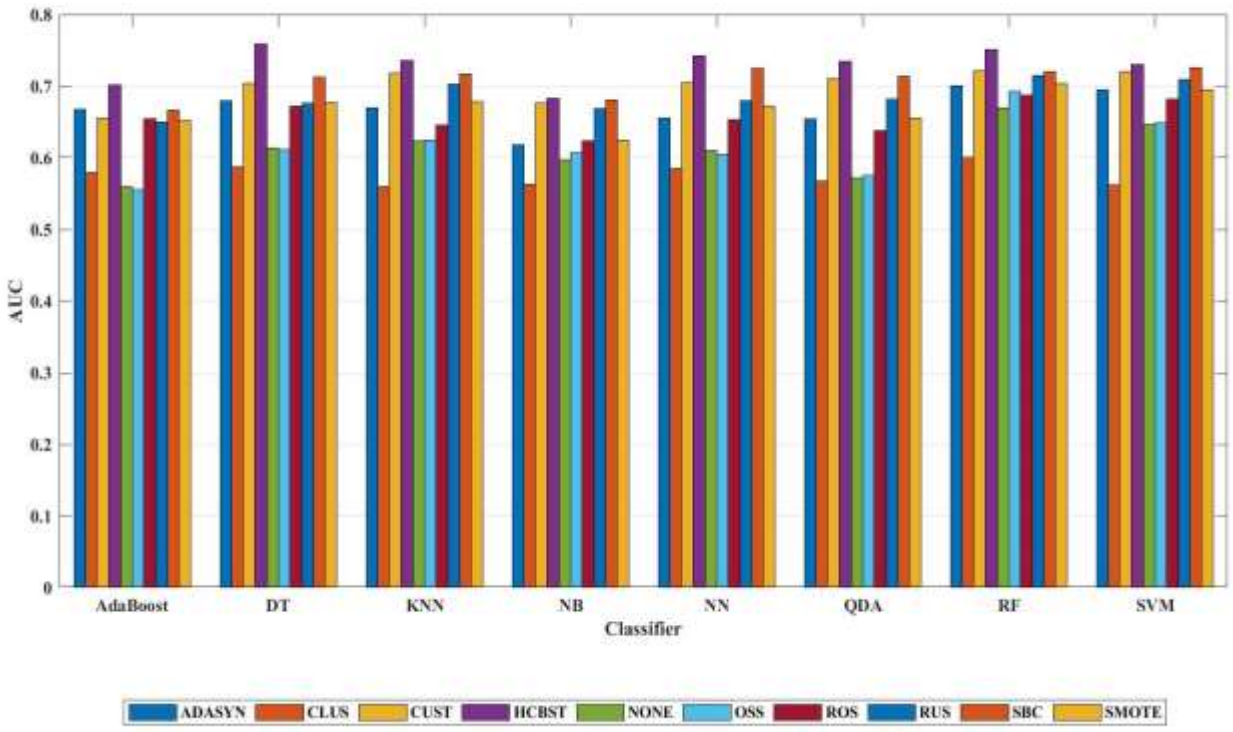


Figure 5.52 Average AUC Performance of HCBST vs. All using all the datasets

5.12 Overall Average Performance of HCBST vs. ALL Using G-Mean

Figure 5.53 illustrated the average G-Mean performances over all the datasets when HCBST was used to resample the data verse the performance vs. the other sampling techniques considered for this study. The results show that HCBST provided a higher average performance of 0.67 concerning G-Mean in all the classifiers used in this study with a minimum performance of 0.54 with QDA classifier and maximum performance of 0.72 with KNN classifier

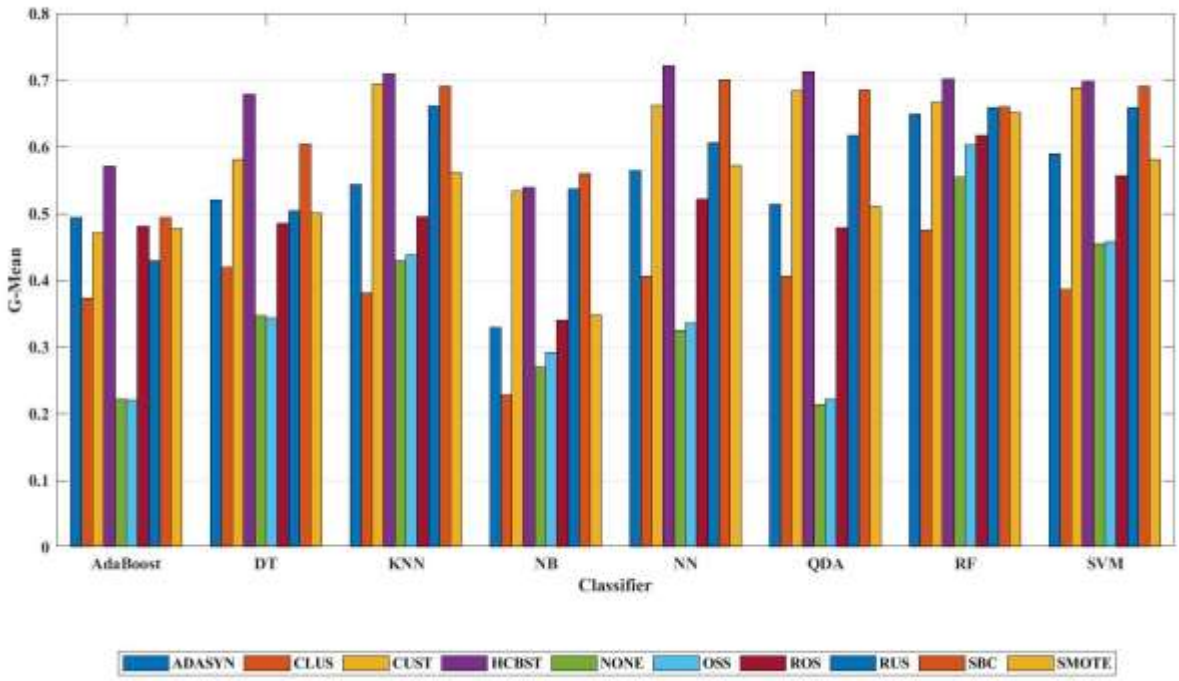


Figure 5.53 Average G-Mean Performance of HCBST vs. All using all the datasets

5.13 Overall Average Performance of HCBST vs. ALL Using MCC

Figure 5.54 illustrated the average MCC performances over all the datasets when HCBST was used to resample the data verse the performance vs. the other sampling techniques considered for this study. The results show that HCBST provided a higher average performance of 0.35 concerning G-Mean in all the classifiers used in this study with a minimum performance of 0.29 with QDA classifier and maximum performance of 0.39 with Naïve Bayes classifier.

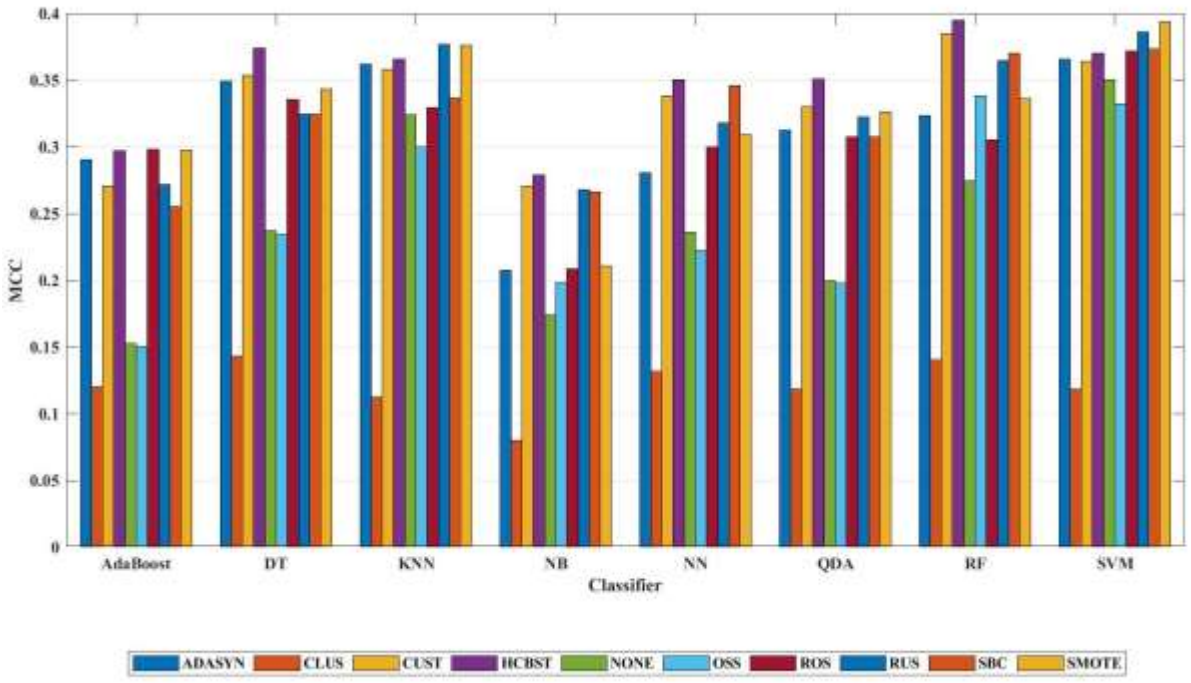


Figure 5.54 Average MCC Performance of HCBST vs. All using all the datasets

5.14 Overall CPU Time of HCBST vs. ALL

Figure 5.55 illustrates the computational times of all the sampling techniques across all the datasets considered for this study. The results show that RUS has the least sampling time with 0.00399 seconds and the highest being CLUS with a sampling time of 0.3281 seconds. It can be seen that the sampling techniques, CUST, SBC, and HCBST that employ k-means clustering using Euclidean distances are more computationally expensive.

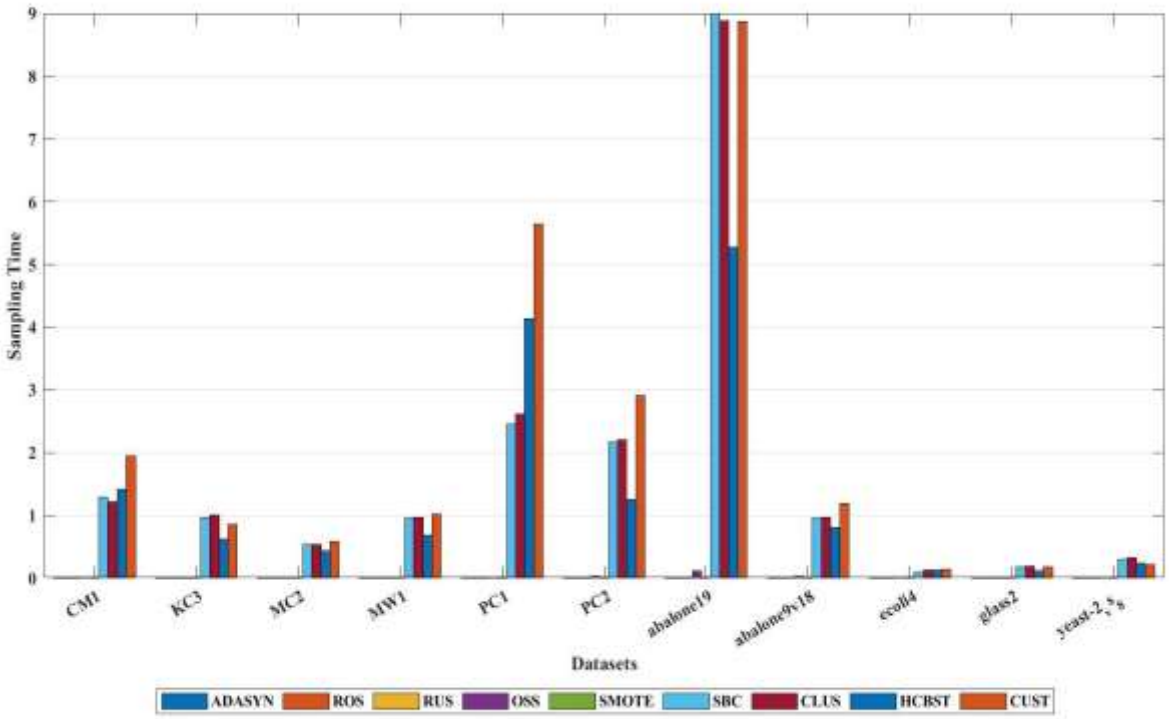


Figure 5.55 CPU Time of HCBST vs. ALL

5.15 Overall CPU Time of HCBST vs. SBC, CUST,

Figure 5.56 illustrates the computation time for HCBST, SBC, and CUST. Among the sampling techniques that employ the use of k-means clustering, given the same number of clusters and using the Euclidean distance metric, HCBST obtained the least sampling time for 7 out of 11 datasets used in this study. PC1, PC2, and ecol4 datasets were dominated by SBC while CUST obtained the least sampling time for the yeast2v8 dataset

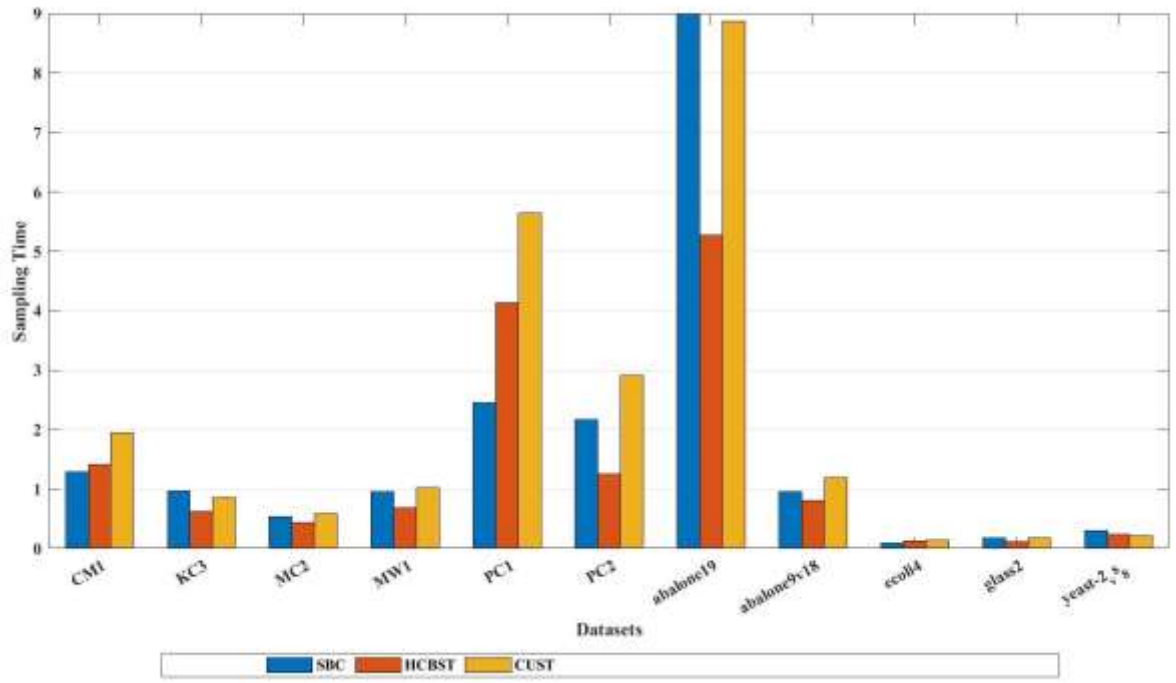


Figure 5.56 CPU Time of HCBST vs. SBC and CUST

5.16 ANOVA ANALYSIS

The results presented are further analyzed to determine their statistical significance. Table 5.1, Table 5.2 and Table 5.3 show the ANOVA analysis of the models in terms of AUC, G-Mean and MCC respectively. The columns of tables represent the classifier and the corresponding sum of squares (SS), degree of freedom (DF), mean squares (MS), F-values (F) and P values of the models.

Table 5.1 AUC ANOVA Analysis

CLASSIFIER	SS	DF	MS	F	P
KNN	0.2703	9	0.0300	1.4881	0.1626
SVM	0.2694	9	0.0299	1.2588	0.2689
Decision Tree	0.2571	9	0.0286	2.8906	0.0044
Random Forest	0.3794	9	0.0422	5.4855	0.0000
Neural Net	0.2527	9	0.0281	1.1855	0.3126
AdaBoost	0.2952	9	0.0328	4.4270	0.0001
Naïve Bayes	0.1604	9	0.0178	1.3369	0.2276
QDA	0.1676	9	0.0186	1.1893	0.3101

Table 5.2 G-Mean ANOVA Analysis

CLASSIFIER	SS	DF	MS	F	P
KNN	2.0458	9	0.2273	3.2688	0.0016
SVM	1.1507	9	0.1279	1.5457	0.1424
Decision Tree	1.2416	9	0.1380	3.4750	0.0009
Random Forest	3.2553	9	0.3617	7.9404	0.0000
Neural Net	1.3690	9	0.1521	1.8668	0.0656
AdaBoost	1.4994	9	0.1666	4.6303	0.0000
Naïve Bayes	0.4294	9	0.0477	1.7115	0.0961
QDA	1.6576	9	0.1842	2.3417	0.0193

Table 5.3 MCC ANOVA Analysis

CLASSIFIER	SS	DF	MS	F	P
KNN	0.4554	9	0.0506	0.5870	0.8051
SVM	0.5270	9	0.0586	0.5705	0.8184
Decision Tree	0.6402	9	0.0711	1.3449	0.2238
Random Forest	0.5859	9	0.0651	1.3565	0.2182
Neural Net	0.4918	9	0.0546	0.5752	0.8146
AdaBoost	0.6045	9	0.0672	1.5822	0.1308
Naïve Bayes	0.5376	9	0.0597	1.3535	0.2196
QDA	0.3577	9	0.0397	0.8718	0.5530

The results from Table 5.1 show that the ANOVA models of Decision Tree, Random Forest and AdaBoost classifiers have p values of 0.0044, 0.0000 and 0.0001 respectively. This means that, at 5% confidence level, the choice of sampling technique significantly affects the performance of the classifiers when using the AUC performance metric. Table 5.2 also indicates that the ANOVA models of KNN, Decision Tree, Random Forest, AdaBoost, and QDA have p values of 0.0016, 0.009, 0.0000, 0.0000, 0.0193 respectively. This means that, at 5% confidence level, the choice of sampling technique significantly affects the performance of these classifiers when using G-Mean as a performance metric.

The results from Table 5.3 shows that none of the sampling techniques significantly outperforms any of the other sampling techniques at 5% confidence level when using MCC as the performance metric. However, at 25%confidence level, the choice of the sampling technique marginally affects the performance of Decision Tree, Random Forest, AdaBoost, and Naïve Bayes classifiers.

In summary, the results also imply that the performance of at least one sampling technique is significantly different from the rest when using AUC and G-Mean performance metric and marginally different when using the MCC performance metric.

The Tukey's Honestly Significance Difference (HSD) test was used to carry out a post-hoc comparison to determine which of the sampling technique resulted in a performance that is significantly different from the others.

Table 5.4, Table 5.5, Table 5.6,

Table 5.7, Table 5.8 and Table 5.9 show the HSD Tests for the classifiers and performance metrics that passed the significance tests in the ANOVA Analysis. In each table, the columns show the sampling techniques, the mean performance and the rank of the sampling techniques. Two groups are statistically significant, then there will be no common letters in their respective columns [72][73].

The Decision Tree AUC performances of the sampling techniques are shown in Table 5.4. It can be observed that ADASYN was outperformed by HCBST, CLUS, RUS, and SBC but statistically the same as the rest of the sampling techniques at 5% confidence level.

Table 5.5 shows the Random Forest AUC performances of the sampling techniques. It can be shown that at 5% confidence level, the performances of HCBST, CLUS and SBC statistically outperformed ADASYN, NONE and OSS but statistically the same as the rest of the sampling techniques.

Table 5.4 HSD test for Decision Tree using AUC

Technique	AUC	
	Mean	HSD
HCBST	**0.7300**	A
ADASYN	0.5626	B
CLUS	0.7199	A
CUST	0.6951	AB
NONE	0.6468	AB
OSS	0.6484	AB
ROS	0.6820	AB
RUS	0.7098	A
SBC	0.7254	A
SMOTE	0.6954	AB

Table 5.5 HSD test for Random Forest using AUC

Technique	AUC	
	Mean	HSD
HCBST	**0.7342**	A
ADASYN	0.5672	B
CLUS	0.7105	A
CUST	0.6539	AB
NONE	0.5724	B
OSS	0.5749	B
ROS	0.6381	AB
RUS	0.6822	AB
SBC	0.7133	A

SMOTE	0.6550	AB

Table 5.6 outlines the AdaBoost AUC performances of the sampling techniques at 5% confidence level. HCBST performed statistically the same as the rest of the sampling techniques except ADASYN. However, CUS, NONE, OSS, ROS failed to outperform ADASYN which was outperformed by HCBST, CLUS, RUS, SBC, and SMOTE.

The Decision Tree G-Mean performance of the sampling techniques is shown in

Table 5.7. It shows that the performances of HCBST, CLUS, and SBC statistically outperformed ADASYN but perform statistically the same as the rest of the sampling techniques at 5% confidence level.

Table 5.6 HSD test for AdaBoost using AUC

Technique	AUC	
	Mean	HSD
HCBST	**0.7357**	A
ADASYN	0.5598	B
CLUS	0.7180	A
CUST	0.6694	AB
NONE	0.6240	AB
OSS	0.6251	AB
ROS	0.6457	AB
RUS	0.7029	A
SBC	0.7174	A
SMOTE	0.6788	A

Table 5.7 HSD test for Decision Tree using G-Mean

Technique	G-Mean	
	Mean	HSD
HCBST	**0.6982**	A
ADASYN	0.3862	B
CLUS	0.6888	A
CUST	0.5899	AB
NONE	0.4543	AB
OSS	0.4580	AB
ROS	0.5562	AB
RUS	0.6585	AB
SBC	0.6902	A
SMOTE	0.5808	AB

Table 5.8 shows the Random Forest G-Mean performances of the sampling techniques. It can be shown that at 5% confidence level, the proposed technique, HCBST statistically outperformed ADASYN, NONE and OSS but statistically the same as CLUS, CUST, ROS, RUS and SBC. However, none of the sampling techniques statistically outperformed any other sampling technique at 5% confidence level.

Table 5.9 shows the AdaBoost AUC performances of the sampling techniques. It can be shown that at 5% confidence level, the performances of HCBST, CLUS and SBC statistically outperformed ADASYN, NONE and OSS but statistically the same as CLUS, CUST, ROS, RUS

SBC and SMOTE, however, none of the sampling techniques statistically outperformed any other sampling technique at 5% confidence level.

Table 5.8 HSD test for Random Forest using G-Mean

Technique	G-Mean	
	Mean	HSD
HCBST	**0.7122**	A
ADASYN	0.4066	BC
CLUS	0.6844	AB
CUST	0.5133	AB
NONE	0.2137	C
OSS	0.2210	C
ROS	0.4784	ABC
RUS	0.6172	AB
SBC	0.6847	AB
SMOTE	0.5106	AB

Table 5.9 HSD test for AdaBoost using G-Mean

Technique	G-Mean	
	Mean	HSD
HCBST	**0.7094**	A
ADASYN	0.3803	BC
CLUS	0.6937	AC
CUST	0.5439	ABC
NONE	0.4294	BC
OSS	0.4383	C
ROS	0.4952	ABC

RUS	0.6614	AC
SBC	0.6911	AC
SMOTE	0.5611	ABC

A box plot is an exploratory data analysis tool that helps to summarize data obtained from an experimental process[74]. The box plot usually consists of variable length boxes divided into quartiles with lines extending from the boxes usually called “whiskers”[75]. The mean is represented by the marker inside the box while the upper and lower quartiles are represented by the top and bottom edges of the box respectively. The “whiskers” represent the maximum and minimum values that are not outliers.

Figure 5.57 illustrates the box plot of Decision Tree AUC performances of the sampling techniques across all the datasets. It can be observed that HCBST, ADASYN, CLUS, CUST, and OSS have one outlier each which indicates that one AUC performance recorded for each of the above sampling techniques has a value that significantly deviates from the rest of the data. However, this does not indicate an error in measurement for this experiment since the outliers for HCBST, CLUS, CUST, and OSS all occurred for the same dataset (yeast2v8) where they outperformed all the other sampling techniques for the study.

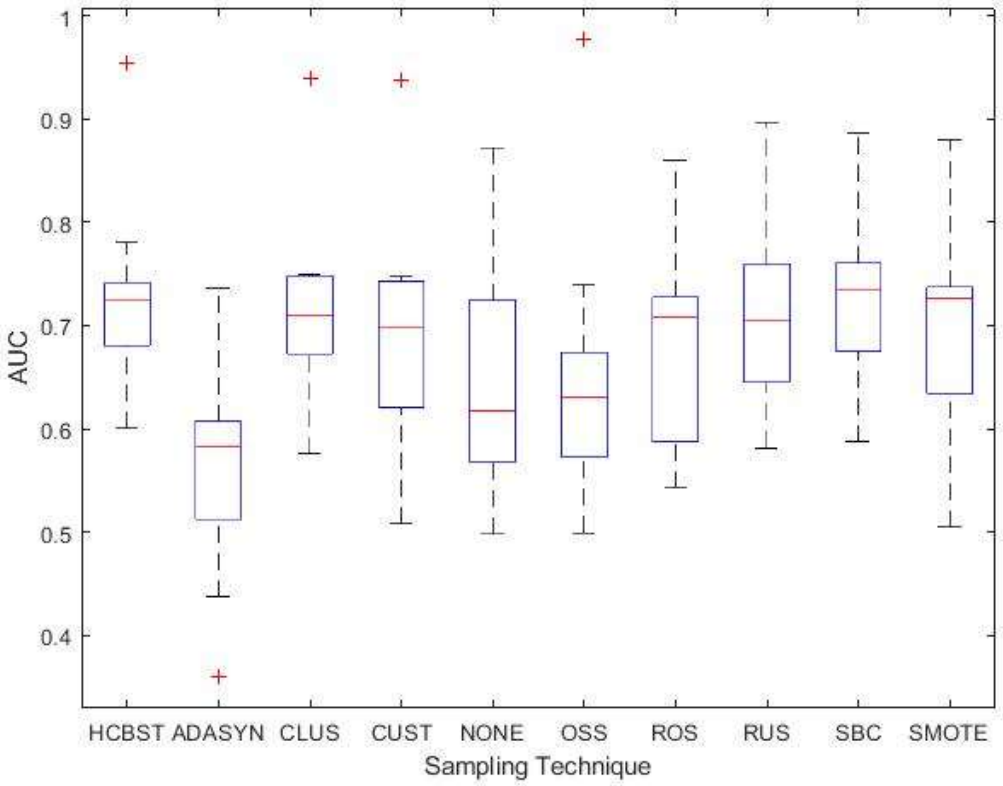


Figure 5.57 Box plot AUC Performance of Decision Tree Classifier

It can also be observed that the means of HCBST and SBC are approximately equal and outperform most of the other techniques. Also, SBC has a higher maximum AUC performance value compared to HCBST. However, the size of their respective boxes indicates that there are higher variations in the AUC performance metric of SBC across all the datasets compared to HCBST.

This can be visualized in Figure 5.58 and Figure 5.59 which shows the variations in the AUC performances of SBC and HCBST respectively. The datasets Abalone19, Abalone19v8, Ecoli4, Glass2, and Yeast2v8 have been represented by A19, A19v8, E4, G2, and Y28 respectively for brevity. Furthermore, HCBST has 75% of its AUC results above 0.674 while SBC has 75% of its AUC results above 0.61. Consequently, relatively small variations about the mean AUC

performance coupled with relatively large representations in the upper quartiles makes HCBST a more robust and reliable AUC performance of the **Decision Tree Classifier** compared to SBC.

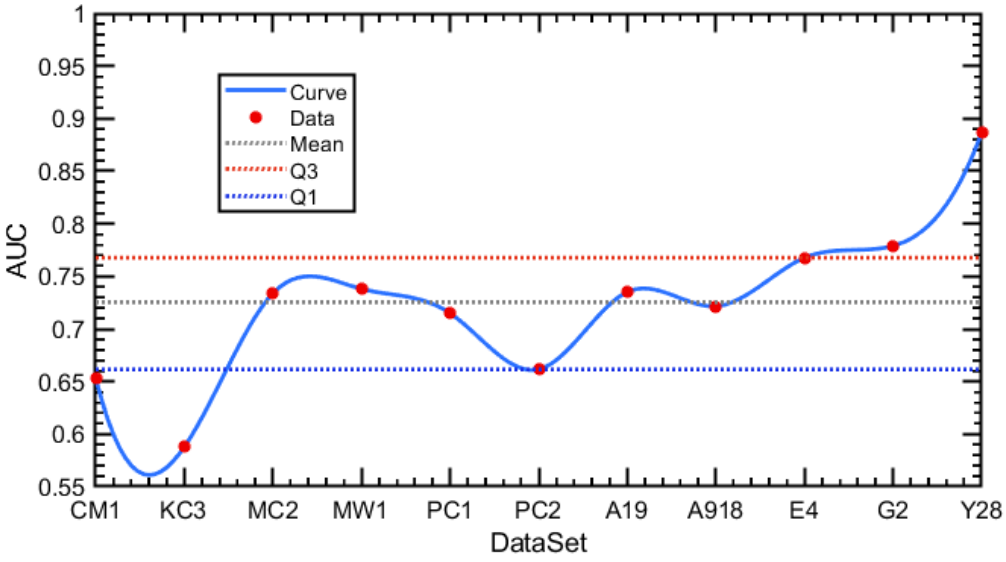


Figure 5.58 A plot of AUC performance of SBC showing the quartiles

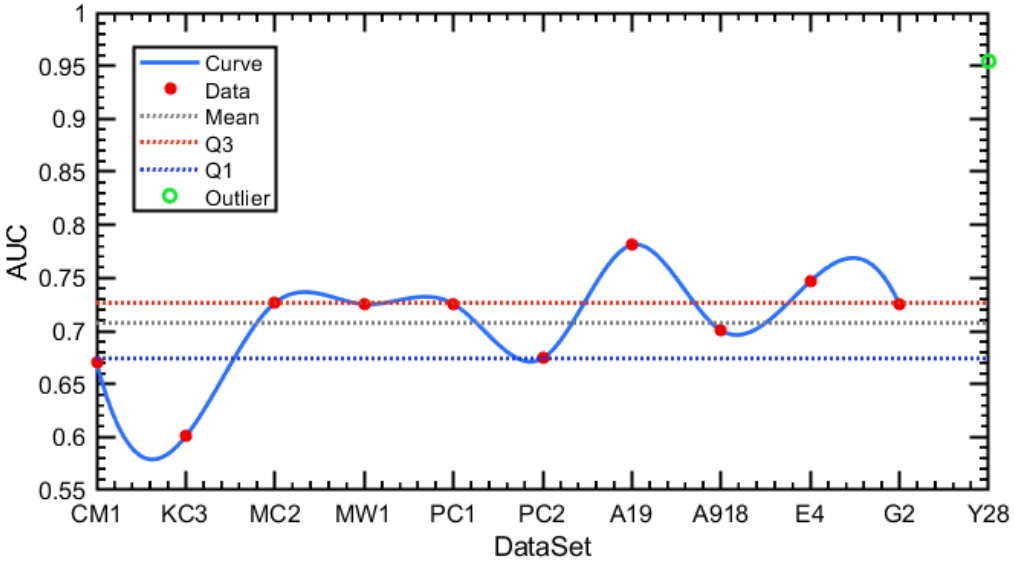


Figure 5.59 A plot of AUC performance of HCBST showing the quartiles

Figure 5.57 illustrates the Boxplot of Random Forest AUC performances of the sampling techniques across all the datasets. It can be observed there are no outliers which indicate that none of the AUC performances recorded for each of the sampling techniques have values that significantly deviate from the rest of the data. Furthermore, HCBST has the highest mean, maximum and minimum values compared to the sampling techniques. It can also be observed that CLUS has the smallest variation about the mean, however, the mean is located close to the lower quartile which indicates that the majority of the AUC values fall in the region near bottom 25% of the entire set of values. The location of the box corresponding to HCBST suggests that 25% of the AUC values of HCBST lie above all the upper quartiles of the rest of the sampling techniques.

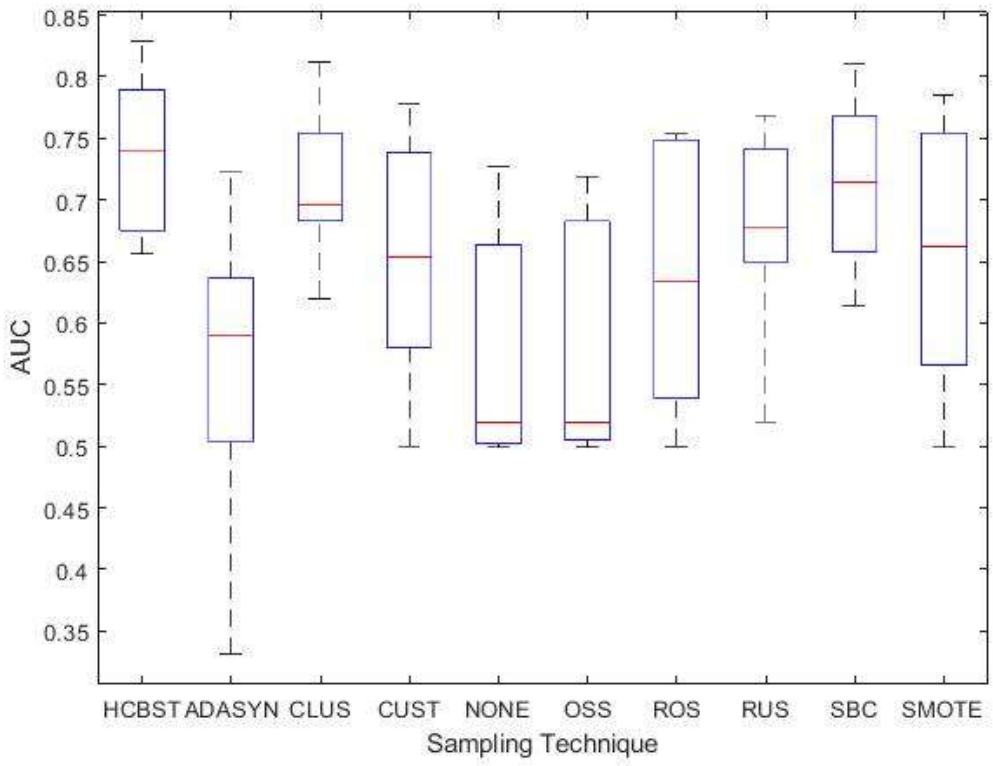


Figure 5.60 A box plot AUC performance of Random Forest Classifier

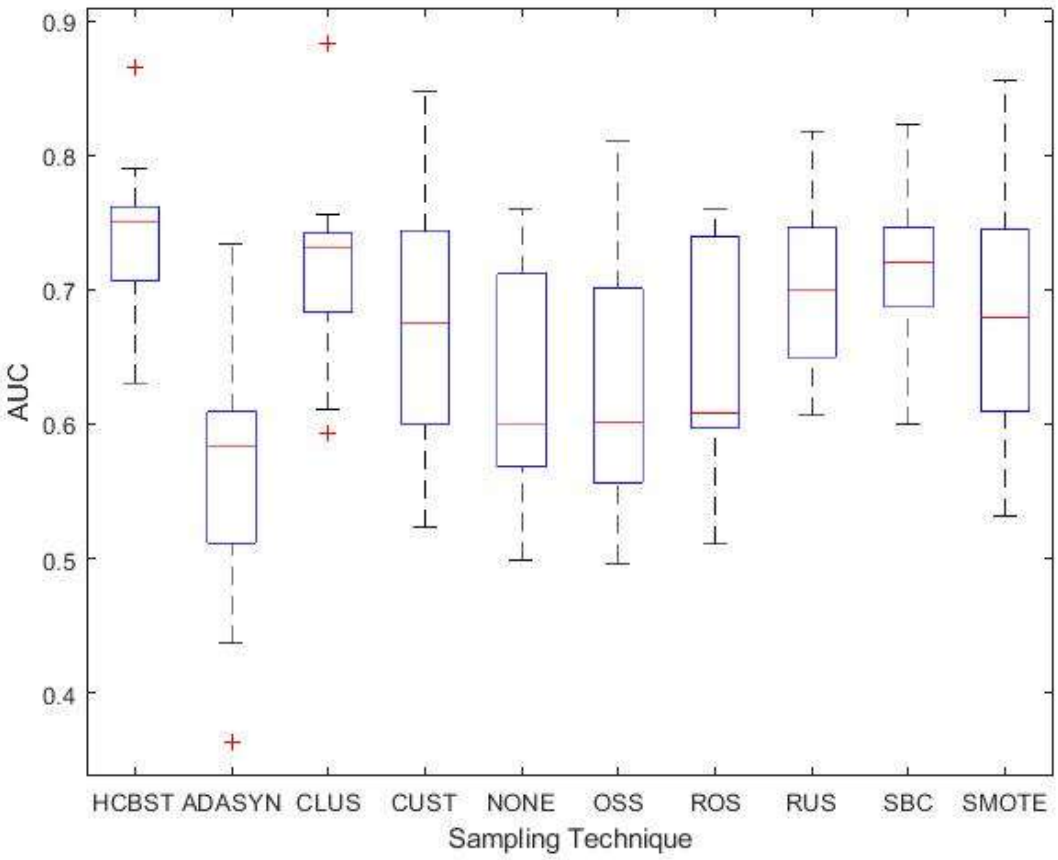


Figure 5.61 A box plot AUC Performance of AdaBoost Classifier

5.17 Practical Application of HCBST

To examine the practical feasibility of the proposed technique and its contribution to the scientific community, two activities were carried out. First, HCBST was used to resample a popular real-world dataset that wasn't used in the experimental process in this study. The results were imported into MATLAB and used as input to the Classification Learner App for further analysis. The ROC and Confusion Matrix plots were observed for both. The same process was repeated using the Classification Learner App without resampling, and the results were compared.

Secondly, a GUI interface to the proposed technique was developed to enable end-users to apply the technique without programming knowledge or adequate technical expertise to work within the runtime environment of HCBST.

5.18 HCBST vs. NONE Using MATLAB on Heart disease Cleveland Dataset

MATLAB Classification Learner App is a GUI tool that allows the use of several classification algorithms without writing a single line of code. It provides an option to run all classification algorithms available in the tool after which the one with the highest performance in terms of Accuracy is highlighted. The sampling parameters used were {7,5, 5} for the oversampling ration, number of clusters, and undersampling ratio respectively. The number of negative samples before and after resampling with HCBST were 160 and 94 respectively, and the number of positive samples before and after resampling with HCBST are 13 and 74 respectively.

Figure 5.62 and Figure 5.63 illustrate the ROC plot for NONE and HCBS, respectively. The red marker on the plots indicates the performances of the selected classifiers, as indicated in the figure below, Fine Tree Classifier. For NONE, the point the market point (0.54,0.94) indicates that 54% of the positive classes were incorrectly classified as negative (FPR) and 94% positive samples were correctly classified as positive (TPR). HCBST, on the other hand, had 3% of positive classes incorrectly classified as negative, and 89% positive samples were correctly classified as positive. In terms of AUC, HCBST had an AUC score of 0.94 while NONE had an AUC score of 0.71. A large area under the curve (AUC) corresponds to better classifier performance.

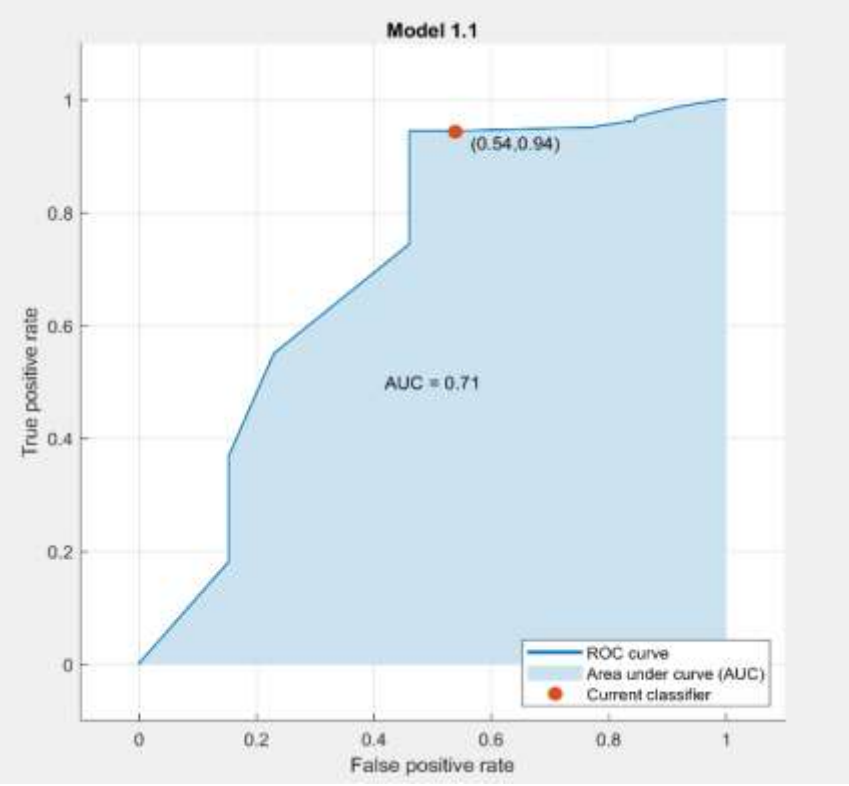


Figure 5.62 ROC plot Using MATLAB Classification Learner App on raw Cleveland Dataset

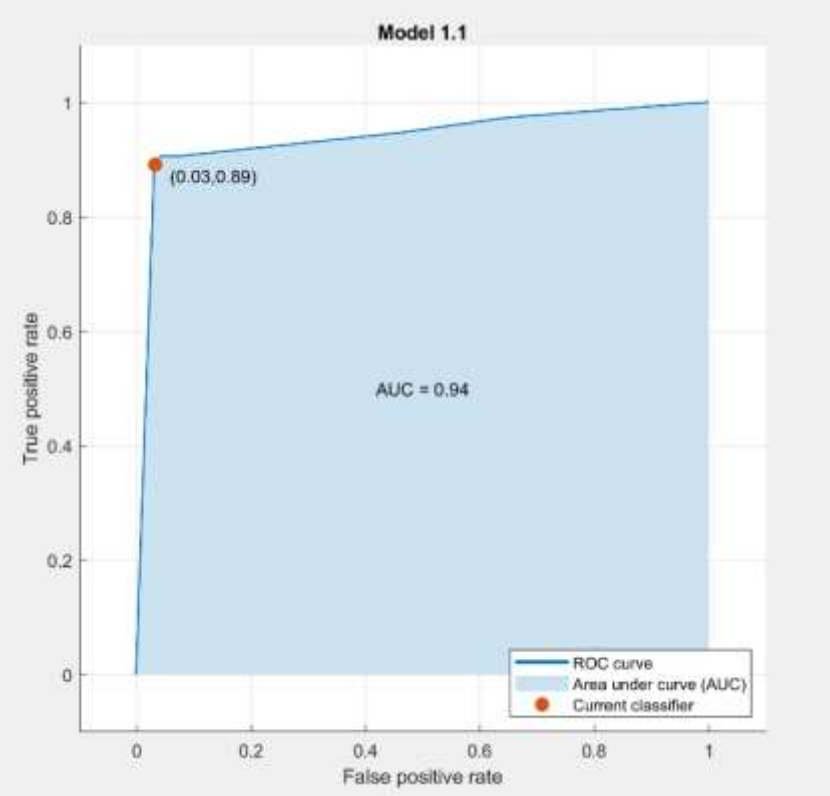


Figure 5.63 ROC plot using MATLAB Classification Learner App on Cleveland Dataset after sampling with HCBST

Figure 5.64 and Figure 5.65 illustrate the confusion matrix plots of NONE and HCBST, respectively. The negative class is indicated by 0, and the positive class is indicated by 1. According to the confusion matrix, 151, 6, 7, and 9 correspond to TN, TP, FN, and FP respectively for NONE while 91, 66, 8, and 3 correspond to TN, TP, FN, and FP respectively for HCBST.

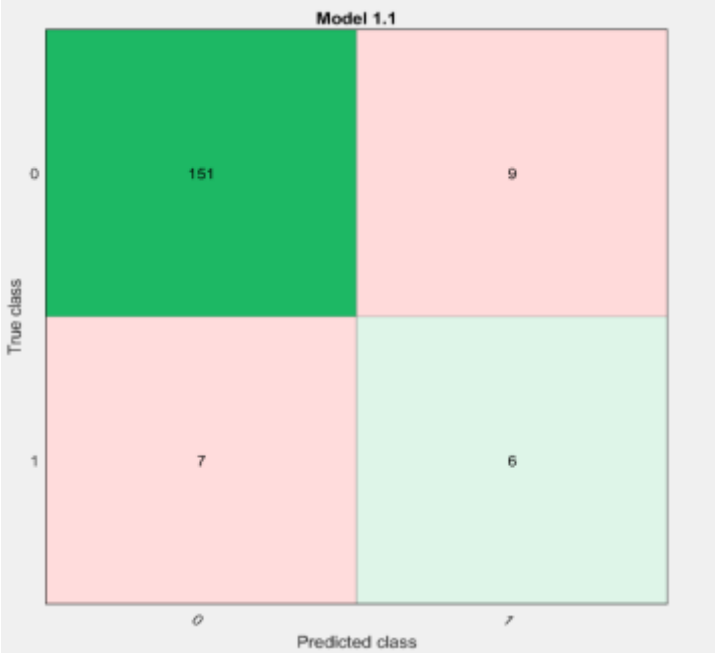


Figure 5.64 Confusion Matrix plot using MATLAB Classification Learner App on Cleveland dataset

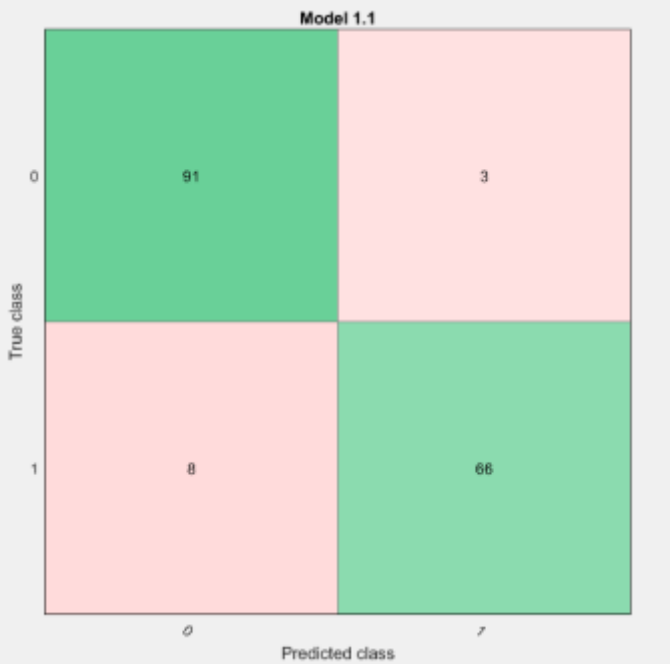


Figure 5.65 Confusion Matrix plot using MATLAB Classification Learner App on Cleveland dataset after sampling with HCBST

5.19 UI Application for HCBST

In addition to the design and implementation of the proposed technique, an application was developed to provide a user interface for the algorithm.

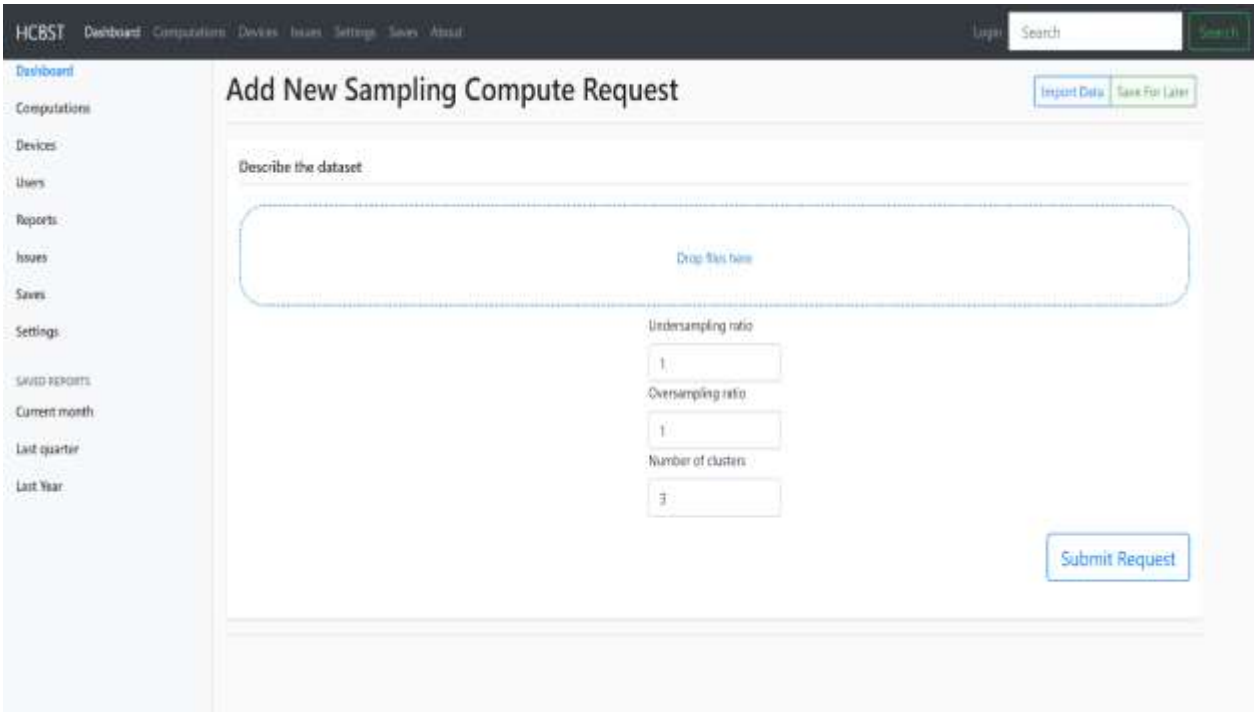
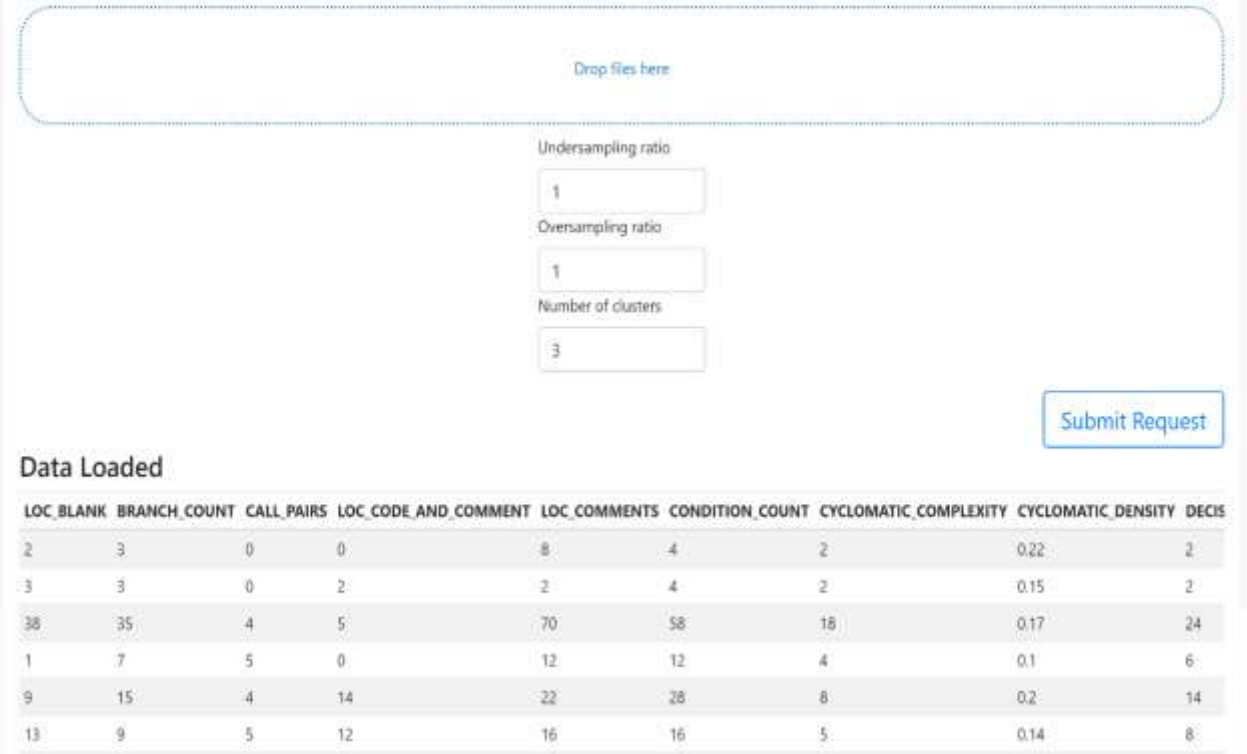


Figure 5.66 Web GUI New Sampling Request



LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMATIC_COMPLEXITY	CYCLOMATIC_DENSITY	DECIS
2	3	0	0	8	4	2	0.22	2
3	3	0	2	2	4	2	0.15	2
38	35	4	5	70	58	18	0.17	24
1	7	5	0	12	12	4	0.1	6
9	15	4	14	22	28	8	0.2	14
13	9	5	12	16	16	5	0.14	8

Figure 5.67 Web GUI Load Data

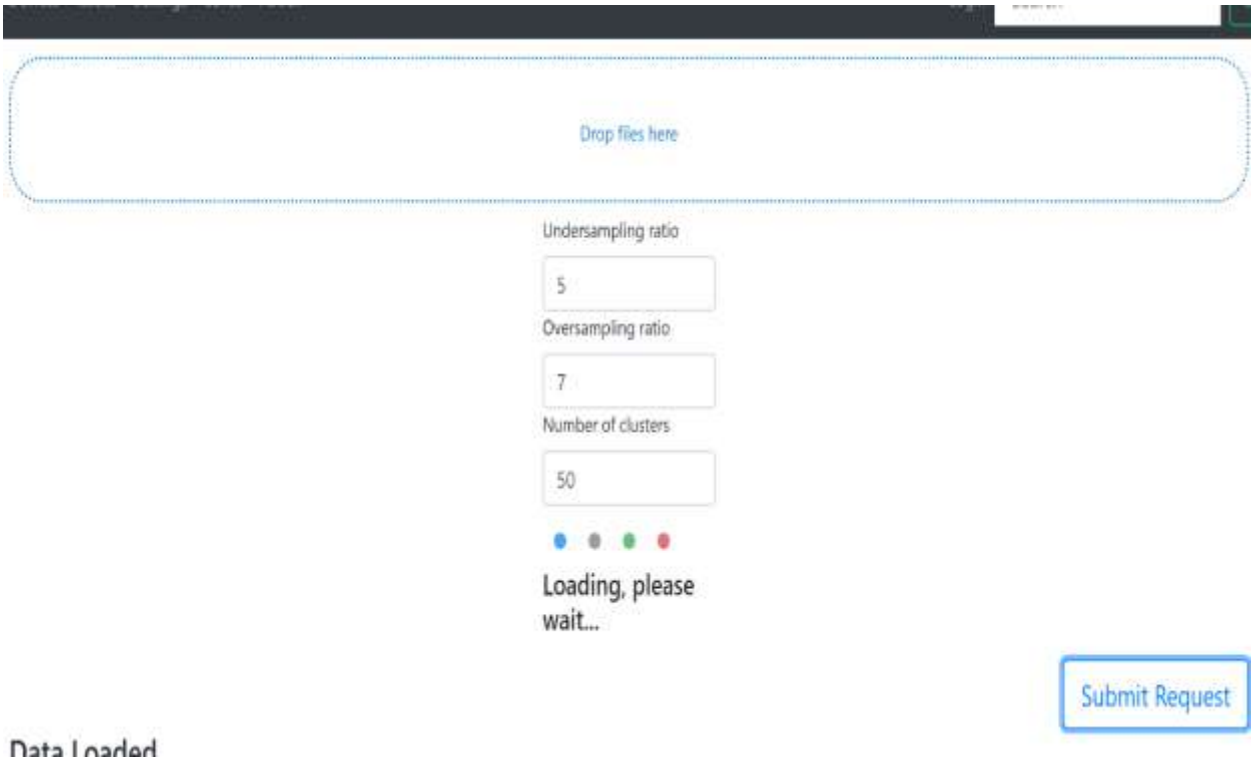


Figure 5.68 Web GUI Perform Resampling

CHAPTER 6

CONCLUSION AND RECOMMENDATION

6.0 Introduction

The chapter summarizes the findings from this research and presents the conclusions and some recommendations as well as some possible further future research directions. The Chapter is organized to highlight the Conclusion, Contribution to knowledge, Observation, and Recommendations for further studies.

6.0 Conclusion

The concept of class imbalance is increasingly becoming an area of interest for many researchers. This phenomenon causes traditional classification algorithms to perform poorly when predicting positive classes with new examples.

Researchers have proposed several techniques to solve this problem at both the data level and the algorithmic level. Data resampling is among the most commonly used methods used to deal with this problem. It involves the manipulation of the training data before applying standard classification techniques. At the data level, recent sampling techniques such as SNOCC and CUST have been shown from previous work to improve the performance of classification algorithms when compared to other standard sampling techniques. CUST uses k-means clustering to under-sample majority instances removing duplicate/repeated instances. However, it does take into consideration, the presence of minority class instances, hence overlapping classes might still be persisted after the sampling process. SNOCC uses an oversampling technique based on SMOTE to oversample minority class instances such that the new synthetic samples are generated in the distribution of the minority samples using a convex combination, unlike SMOTE where the new

samples are generated on the line segmented between the seed samples. Although SNOCC, generates samples that are more representative of the minority class, it does not consider the presence of majority class instances. Hence the problem of overlap in classes is not alleviated.

The combination of class imbalance and class overlap lead to further deterioration of the performance of standard machine learning classifiers. The proposed technique, which is based on CUST on SNOCC attempts to address this issue by taking into consideration the local proximity of minority and majority class instances. Several experiments were carried out to compare to the performance of the proposed technique to eight other sampling methods using KNN, SVM, Decision Tree, Random Forest, Neural Network, AdaBoost, Naïve Bayes, and Quadratic Discriminant Analysis classifiers.

In Summary, the first part of the ANOVA analysis revealed that AdaBoost, Decision Tree, and Random Forest Classifiers were most significantly affected by the sampling technique used to sample the data before classification. Furthermore, the performance metrics that were also most significantly affected by the choice of sampling technique are AUC and G-Mean. Additionally, a post hoc analysis was carried out to determine which of the sampling technique resulted in a performance that is significantly different from the others. The results from the post-hoc analysis is confirmed by further box plot analysis which showed that HCBST had the highest mean performance values and the smallest variations of performance about the means for AdaBoost, Decision Tree and Random Forest Classifier. This confirms the robustness of the proposed technique in terms of overall average performance which was highlighted in sections 5.11 to 5.13 of chapter 5. However, it is worth mentioning that the performance of HCBST was closely followed by SBC, RUS, and CLUS in a significant number of cases

6.1 Contribution to Knowledge

This research designed and implemented a novel resampling technique, Hybrid Cluster-Based Undersampling Technique (HCBST) that takes into account the presence class overlap. The technique was evaluated by eight classification algorithms on datasets from the UCI repository and NASA MDP repository. The performance of the classifiers after resampling the datasets with HCBST was compared to the performance with no sampling performed. The results showed that HCBST significantly improved the performance of the classifiers concerning AUC and G-Mean, and marginally with MCC obtaining the highest average scores of 0.73, 0.67 and 0.35 in AUC, G-Mean and MCC respectively across all the classifiers used for this study. Also, the performance of eight other sampling techniques was compared to HCBST. The results from the experiments showed that HCBST produced better overall results compared to the other sampling techniques used in this study.

Furthermore, to assess the real-world application of this study, an experiment was carried out on a well-known scientific tool (MATLAB) to compare the generalizability of the proposed technique. A real-world dataset was resampled using HCBST and classified using the in-built MATLAB Classification Learner App and the results compared to applying the tool when no resampling is carried out on the dataset. The results from the experiment showed that HCBST significantly improves the classification algorithms available in the tool. Additionally, a web portal was created to serve as a user interface to the proposed technique to enable experimenters with no technical knowledge or programming skills to easily apply the technique on datasets.

6.2 Observations

HCBST may further improve a classification algorithm by tuning the input parameters such as the number of majority samples to keep, the number of minority samples to generate or the number of

clusters to use. However, manually finetuning these parameters can be time-consuming depending on the size of the dataset. This problem is predominant in other sampling techniques that make use of k-means clustering or expensive distance computations. Consequently, the optimum set of parameters may not be attained. Furthermore, HCBST performed marginally better on average in terms of MCC, however, it was observed that this performance metric was the least significantly affected by the choice of sampling technique.

6.3 Recommendations

Based on the above observations, it is therefore of research interest to investigate some heuristic approaches to select the optimum set of parameters. Furthermore, a thorough investigation is needed to explain why MCC is least significantly affected by the choice of sampling technique.

REFERENCES

[1] R. U. Saravanan and P. U. Sujatha, "A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2018, pp. 945–949, doi: 10.1109/ICCONS.2018.8663155.

[2] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, "Unsupervised Learning Based On Artificial Neural Network: A Review," in *2018 IEEE International Conference on Cyborg and Bionic Systems, CBS 2018*, 2019, doi: 10.1109/CBS.2018.8612259.

[3] M. Goldstein and S. Uchida, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data.," *PLoS One*, 2016, doi: 10.1371/journal.pone.0152173.

[4] C. Doersch and A. Zisserman, "Multi-task Self-Supervised Visual Learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, doi: 10.1109/ICCV.2017.226.

[5] H. Pan and Z. Kang, "Robust Graph Learning for Semi-Supervised Classification," in *Proceedings - 2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2018*, 2018, doi: 10.1109/IHMSC.2018.00068.

[6] A. Singh, N. Thakur, and A. Sharma, "A Review of Supervised Machine Learning Algorithms," *2016 3rd Int. Conf. Comput. Sustain. Glob. Dev.*, 2016.

[7] L. Demidova and I. Klyueva, "Data classification based on the hybrid versions of the particle swarm optimization algorithm," in *2018 7th Mediterranean Conference on Embedded Computing, MECO 2018 - Including ECYPS 2018, Proceedings*, 2018, doi:

10.1109/MECO.2018.8406069.

[8] Z. Y. Han and J. Wang, "Fault detection based on Sensitive Marginal Fisher Analysis for class imbalance," in *CGNCC 2016 - 2016 IEEE Chinese Guidance, Navigation and Control Conference*, 2017, doi: 10.1109/CGNCC.2016.7828774.

[9] Z. Jin, Q. Li, D. Zeng, and L. Wang, "Filtering spam in Weibo using ensemble imbalanced classification and knowledge expansion," in *2015 IEEE International Conference on Intelligence and Security Informatics: Securing the World through an Alignment of Technology, Intelligence, Humans and Organizations, ISI 2015*, 2015, doi: 10.1109/ISI.2015.7165952.

[10] G. Schaefer and T. Nakashima, "Strategies for addressing class imbalance in ensemble classification of thermography breast cancer features," in *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, 2015, doi: 10.1109/CEC.2015.7257177.

[11] M. S. Reza and J. Ma, "Imbalanced histopathological breast cancer image classification with convolutional neural network," in *International Conference on Signal Processing Proceedings, ICSP*, 2019, doi: 10.1109/ICSP.2018.8652304.

[12] S. Subudhi and S. Panigrahi, "Effect of Class Imbalanceness in Detecting Automobile Insurance Fraud," in *Proceedings - 2nd International Conference on Data Science and Business Analytics, ICDSBA 2018*, 2018, doi: 10.1109/ICDSBA.2018.00104.

[13] R. A. Bauder and T. M. Khoshgoftaar, "Medicare fraud detection using random forest with class imbalanced big data," in *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science, IRI 2018*, 2018, doi:

10.1109/IRI.2018.00019.

[14] R. A. Mohammed, K. W. Wong, M. F. Shiratuddin, and X. Wang, "Scalable machine learning techniques for highly imbalanced credit card fraud detection: A comparative study," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, doi: 10.1007/978-3-319-97310-4_27.

[15] G. E. Melo-Acosta, F. Duitama-Munoz, and J. D. Arias-Londono, "Fraud detection in big data using supervised and semi-supervised learning techniques," in *2017 IEEE Colombian Conference on Communications and Computing, COLCOM 2017 - Proceedings*, 2017, doi: 10.1109/ColComCon.2017.8088206.

[16] S. Rodda and U. S. R. Erothi, "Class imbalance problem in the Network Intrusion Detection Systems," in *International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016*, 2016, doi: 10.1109/ICEEOT.2016.7755181.

[17] R. A. Sowah, M. A. Agebure, A. M. Godfrey, K. M. Koumadi, and S. Y. Fiawoo, "New Cluster Undersampling Technique for Class Imbalance Learning," *Int. J. Mach. Learn. Comput.*, vol. 6, no. 3, 2016.

[18] N. Ofek, L. Rokach, R. Stern, and A. Shabtai, "Fast-CBUS: A fast clustering-based undersampling method for addressing the class imbalance problem," *Neurocomputing*, 2017, doi: 10.1016/j.neucom.2017.03.011.

[19] M. Gao, X. Hong, S. Chen, C. J. Harris, and E. Khalaf, "PDFOS: PDF estimation based over-sampling for imbalanced two-class problems," *Neurocomputing*, 2014, doi: 10.1016/j.neucom.2014.02.006.

[20] S. J. Yen and Y. S. Lee, “Cluster-based under-sampling approaches for imbalanced data distributions,” *Expert Syst. Appl.*, 2009, doi: 10.1016/j.eswa.2008.06.108.

[21] Z. Zheng, Y. Cai, and Y. Li, “Oversampling method for imbalanced classification,” *Comput. Informatics*, 2015.

[22] X.-Y. Liu, J. Wu, and Z.-H. Zhou, “Exploratory Undersampling for Class Imbalance Learning,” *IEEE Trans. Syst. Man Cybern.*, 2009, doi: 10.1109/TSMCB.2008.2007853.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, 2002, doi: 10.1613/jair.953.

[24] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, “MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction,” *IEEE Trans. Softw. Eng.*, 2018, doi: 10.1109/TSE.2017.2731766.

[25] M. Elter, R. Schulz-Wendtland, and T. Wittenberg, “The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process,” *Med. Phys.*, 2007, doi: 10.1118/1.2786864.

[26] Y.-H. Liu and Y.-T. Chen, “Face recognition using total margin-based adaptive fuzzy support vector machines.,” *IEEE Trans. Neural Netw.*, 2007, doi: 10.1109/TNN.2006.883013.

[27] V. García, R. Alejo, J. S. Sánchez, J. M. Sotoca, and R. A. Mollineda, “Combined Effects of Class Imbalance and Class Overlap on Instance-Based Classification,” 2006.

[28] B. Tang and H. He, “KernelADASYN: Kernel based adaptive synthetic data generation

for imbalanced learning," in *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, 2015, doi: 10.1109/CEC.2015.7256954.

[29] C. Drummond and R. C. Holte, "C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," *Work. Learn. from Imbalanced Datasets II*, 2003, doi: 10.1.1.68.6858.

[30] P. Xenopoulos, "Introducing DeepBalance: Random deep belief network ensembles to address class imbalance," in *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, 2018, doi: 10.1109/BigData.2017.8258364.

[31] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explor. Newsl.*, 2004, doi: 10.1145/1007730.1007735.

[32] H. HE and E. a. Garcia, "Learning from Imbalanced Data Sets.," *IEEE Trans. Knowl. Data Eng.*, 2010, doi: 10.1109/TKDE.2008.239.

[33] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, 2010, doi: 10.1109/TSMCA.2009.2029559.

[34] J. Gong and H. Kim, "RHSBoost: Improving classification performance in imbalance data," *Comput. Stat. Data Anal.*, 2017, doi: 10.1016/j.csda.2017.01.005.

[35] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, 2009, doi: 10.1109/TSMCA.2009.2027131.

[36] "A Novel Boundary Oversampling Algorithm Based on Neighborhood Rough Set Model: NRSBoundary-SMOTE : Figure 2[Image]." [Online]. Available: https://www.hindawi.com/journals/mpe/2013/694809/fig2/. [Accessed: 29-Sep-2018].

[37] S. Chen, H. He, and E. A. Garcia, "RAMOBoost: Ranked minority oversampling in boosting," *IEEE Trans. Neural Networks*, 2010, doi: 10.1109/TNN.2010.2066988.

[38] S. Barua, M. M. Islam, and K. Murase, "A novel synthetic minority oversampling technique for imbalanced data set learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, doi: 10.1007/978-3-642-24958-7_85.

[39] W. Juanjuan, X. Mantao, W. Hui, and Z. Jiwu, "Classification of imbalanced data by using the SMOTE algorithm and locally linear embedding," in *International Conference on Signal Processing Proceedings, ICSP*, 2007, doi: 10.1109/ICOSP.2006.345752.

[40] R. Blagus and L. Lusa, "SMOTE for high-dimensional class-imbalanced data," *BMC Bioinformatics*, 2013, doi: 10.1186/1471-2105-14-106.

[41] N. V. (Universit. of N. D. Chawla, A. H. I. Lazarevic, L. O. Hall, and K. B. of N. D. Bowyer, "SMOTEBoost: Improving Prediction of the Minority Class in Boosting," in *7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2003, pp. 107–119.

[42] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2008, doi: 10.1109/IJCNN.2008.4633969.

[43] M. Kubat and S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One Sided Selection," *Icml*, 1997, doi: 10.1007/s13398-014-0173-7.2.

[44] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *Am. Stat.*, 1992, doi: 10.1080/00031305.1992.10475879.

[45] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Inf. Theory*, 1967, doi: 10.1109/TIT.1967.1053964.

[46] P. Hall, B. U. Park, and R. J. Samworth, "Choice of neighbor order in nearest-neighbor classification," *Ann. Stat.*, 2008, doi: 10.1214/07-AOS537.

[47] C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn.*, 1995, doi: 10.1023/A:1022627411411.

[48] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, 2008, doi: 10.1007/s10115-007-0114-2.

[49] T. K. Ho, "Random decision forests," in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 1995, doi: 10.1109/ICDAR.1995.598994.

[50] T. Hastie, *Elements of Statistical Learning*. 2009.

[51] A. A. Mullin and F. Rosenblatt, "Principles of Neurodynamics.," *Am. Math. Mon.*, 2006, doi: 10.2307/2312103.

[52] D. E. Ruineihart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations Error Propagation," *Cogn. Sci.*, 1986.

[53] M. H. Popel, K. M. Hasib, S. A. Habib, and F. M. Shah, "A Hybrid Under-Sampling Method (HUSBoost) to Classify Imbalanced Data," in *2018 21st International Conference*

of Computer and Information Technology, ICCIT 2018, 2019, doi: 10.1109/ICCITECHN.2018.8631915.

[54] G. F. Hughes, "On the Mean Accuracy of Statistical Pattern Recognizers," *IEEE Trans. Inf. Theory*, 1968, doi: 10.1109/TIT.1968.1054102.

[55] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attribtues to Learn Defect Predictors," *TSE*, 2007, doi: 10.1109/TSE.2007.10.

[56] G. M. Weiss, "Mining with Rarity : A Unifying Framework," *ACM SIGKDD Explor. Newsl. 6.1*, 2004.

[57] J. S. Akosa, "Predictive Accuracy : A Misleading Performance Measure for Highly Imbalanced Data," *SAS Glob. Forum*, 2017.

[58] S. H. Park, J. M. Goo, and C. H. Jo, "Receiver operating characteristic (ROC) curve: Practical review for radiologists," *Korean Journal of Radiology*. 2004, doi: 10.3348/kjr.2004.5.1.11.

[59] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, 2006, doi: 10.1016/j.patrec.2005.10.010.

[60] D. M. W. Powers and Ailab, "Evaluation : From Precision , Recall and F-Factor to ROC , Informedness , Markedness & Correlation," *J. Mach. Learn. Technol.*, 2007.

[61] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric," *PLoS One*, 2017, doi: 10.1371/journal.pone.0177678.

[62] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA

MDP data sets," *IET Softw.*, 2012, doi: 10.1049/iet-sen.2011.0132.

[63] A. A. A.Frank, "UCI Machine Learning Repository: Data Sets," *Univ. Calif. Irvine Sch. Inf.*, 2007.

[64] C. Tantithamthavorn, "GitHub - klainfo/NASADefectDataset: NASA Cleaned Defect Datasets," 2016. [Online]. Available: https://github.com/klainfo/NASADefectDataset/blob/master/README.md.

[65] D. Gray, D. Bowes, N. Davey, Yi Sun, and B. Christianson, "The misuse of the NASA Metrics Data Program data sets for automated software defect prediction," in *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 2011, doi: 10.1049/ic.2011.0012.

[66] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, 2013, doi: 10.1109/TSE.2013.11.

[67] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the NASA software defect data sets," 2016, doi: 10.1145/2915970.2916007.

[68] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot," *J. Mach. Learn. Res.*, 2011.

[69] G. van (C. voor W. en I. (CWI)) Rossum, "Python tutorial," *Python*, 1995.

[70] D. M. Rocke and D. L. Woodruff, "Identification of Outliers in Multivariate Data," *J. Am. Stat. Assoc.*, 1996, doi: 10.1080/01621459.1996.10476975.

[71] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster*

Analysis (Wiley Series in Probability and Statistics). 1990.

[72] R. A. Cribbie and H. J. Keselman, "Pairwise multiple comparisons: A model comparison approach versus stepwise procedures," *Br. J. Math. Stat. Psychol.*, 2003, doi: 10.1348/000711003321645412.

[73] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H. P. Piepho, and R. Schmid, "Algorithms for compact letter displays: Comparison and evaluation," *Comput. Stat. Data Anal.*, 2007, doi: 10.1016/j.csda.2006.09.035.

[74] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: A simple visual method to interpret data," *Ann. Intern. Med.*, 1989, doi: 10.7326/0003-4819-110-11-916.

[75] K. Potter, "Methods for Presenting Statistical Information: The Box Plot," *Vis. Large Unstructured Data Sets*, 2006.

APPENDICES

APPENDIX A

Code Listing 1

```
for i in range(len(self.Min)):
    term = self.Min.iloc[i, 0:self.Min.shape[1]-1]
    term = pd.DataFrame(term)
    knn = iknn.KNN(o, kn)
   [pred, neigh, neigh_dist] = knn.fit(self.Min.iloc[:, 0:self.Min.shape[1]-1].values, y.values, term)
    m_i.append(np.mean(neigh_dist))
    neighs.append(neigh)
    neigh_dists.append(neigh_dist)
sigma = np.mean(m_i) + np.std(m_i)
```

Code Listing 2

```
for m in range(No):
    sigma_neigh =[]
    max_search = 20
    search_count = 0
    s_i = 0
    x = 0
    y = 0
    z = 0
    while len(sigma_neigh) < 2 and search_count < max_search:
        a = round(random.uniform(0, 1), 1)
```

```
x = random.uniform(0, a)
y = random.uniform(0, 1 - a)
z = 1 - (x + y)
s_i = choice(self.Min.values)
index_s_i = np.bincount(np.where(self.Min.values == s_i)[0]).argmax()
neighd_s_i = neigh_dists[index_s_i]  # neigh_dists[index_s_i]
neigh_s_i = neighs[index_s_i]
sigma_neigh =[]
for i in range(len(neighd_s_i)):
    if len(sigma_neigh) > 2:
        continue
    if neighd_s_i[i] < sigma:
        sigma_neigh.append(neigh_s_i[i])

search_count += 1

#print(sigma_neigh)
if len(sigma_neigh) >= 2:
    s_i_new = x * s_i + y * self.Min.values[sigma_neigh[0]] + z * self.Min.values[sigma_neigh[1]]
```

Code Listing 3

```
u = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[0]]])
h = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[1]]])
g = (u+h)/2
f = self.check_min_sample(self.Maj.values,[s_i_new])
```

```
if f > g:
    final_samples.append(s_i_new)
if self.pm is None and self.ru == self.Maj.shape[0]/self.Min.shape[0]:
    return self.Maj
if self.pm is not None:
    return self.stack_undersample()
_Min = self.Min[self.Min.columns[0:self.Min.shape[1]-1]].values
_Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1]-1]].values
_Maj_new = {}
[_Min, _Maj] = self.remove_inconsistent(_Min, _Maj)
samples = 0
km = ik_means.K_Means(k=self.km)
km.fit(_Maj)
clusters = km.get_clusters()


def oversample(self):
    _Min = self.Min[self.Min.columns[0:self.Min.shape[1]-1]].values
    _Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1]-1]].values
    y = self.Min.iloc[:, self.Min.shape[1] - 1]
    No = self.ro * (len(_Min)) - (len(_Min))
    No = int(np.round(No))
    print(No)
    if No == 0:
        return self.Min.values
    o = choice(self.Min.iloc[:, 0:self.Min.shape[1]-1].values)
    o = o.reshape(1, self.Min.iloc[:, 0:self.Min.shape[1]-1].shape[1])
```

```
final_samples = self.Min.values.copy().tolist()
m_i =[]
neighs =[]
neigh_dists =[]
for i in range(len(self.Min)):
    term = self.Min.iloc[i, 0:self.Min.shape[1]-1]
    term = pd.DataFrame(term)
    knn = iknn.KNN(o, kn)
   [pred, neigh, neigh_dist] = knn.fit(self.Min.iloc[:, 0:self.Min.shape[1]-1].values, y.values, term)
    m_i.append(np.mean(neigh_dist))
    neighs.append(neigh)
    neigh_dists.append(neigh_dist)
sigma = np.mean(m_i) + np.std(m_i)
# generating alphas
for m in range(No):
    sigma_neigh =[]
    max_search = 20
    search_count = 0
    s_i = 0
    x = 0
    y = 0
    z = 0
    while len(sigma_neigh) < 2 and search_count < max_search:
        a = round(random.uniform(0, 1), 1)
        x = random.uniform(0, a)
        y = random.uniform(0, 1 - a)
```

```
    z = 1 - (x + y)
    s_i = choice(self.Min.values)
    index_s_i = np.bincount(np.where(self.Min.values == s_i)[0]).argmax()
    neighd_s_i = neigh_dists[index_s_i]  # neigh_dists[index_s_i]
    neigh_s_i = neighs[index_s_i]
    sigma_neigh =[]
    for i in range(len(neighd_s_i)):
        if len(sigma_neigh) > 2:
            continue
        if neighd_s_i[i] < sigma:
            sigma_neigh.append(neigh_s_i[i])

    search_count += 1

#print(sigma_neigh)
if len(sigma_neigh) >= 2:
    s_i_new = x * s_i + y * self.Min.values[sigma_neigh[0]] + z * self.Min.values[sigma_neigh[1]]

    u = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[0]]])
    h = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[1]]])
    g = (u+h)/2
    #print(g)
    f = self.check_min_sample(self.Maj.values,[s_i_new])
    #print(f)
    #print("-----------------")
    if f > g:
```

```
                final_samples.append(s_i_new)
                # print("added")
            else:
                # print("Not added")
                pass
    final_samples = np.array(final_samples)
    return np.concatenate((self.Min.values, final_samples), axis=0)
```

Code Listing 4

```
max_search = 50
for i in range(self.km):
    _Maji = self.ru * (len(_Min) / len(_Maj)) * len(clusters[i])
    sc = 1
    giveup = False
    while _Maji > 0:
        if len(clusters[i]) == 0:
            break
        if sc >= max_search:
            #print("max search")
            giveup = True
```

```
        np.random.shuffle(clusters[i])
        new_sample = clusters[i][0].copy()
        new_sample.append(0)
        indices = self.check_maj_sample(self.data.values,[new_sample])
        overlap = False
        for index in indices:
            if self.data.iloc[index]['class'] == 1:
                sc += 1
                overlap = True
        if not overlap or giveup:
            _Maj_new[samples] = clusters[i][0]
            samples += 1
            for j in range(len(clusters[i]) - 1):
                if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                    np.delete(clusters[i], j + 1, 0)
            np.delete(clusters[i], 0, 0)
            _Maji -= 1
            giveup = False
            sc = 1
_Maj_new = np.array(list(_Maj_new.values()))
```

Code Listing 5

```
def stack_undersample(self):
    print("stack_triggered")
    _Min = self.Min[self.Min.columns[0:self.Min.shape[1] - 1]].values
    _Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1] - 1]].values
    _MajDf = _Maj
    _Maj_new = {}
   [_Min, _Maj] = self.remove_inconsistent(_Min, _Maj)
    keys = list(self.data.columns.values)
    temp = np.zeros((len(_Maj), _Maj.shape[1]+1))
    temp[:, :-1] = _Maj
    _MajDf = pd.DataFrame(data=temp, columns=keys)
    origin = np.zeros(shape=(1, _Maj.shape[1]))
    km = ik_means.K_Means(originPoint=origin[0], k=self.km)
    km.fit(_Maj)
    clusters = km.get_clusters()
    max_search = 25
    for i in range(self.km):
        _Maji = self.pm * len(clusters[i])
        sc = 1
        giveup = False
        while _Maji > 0:
            if len(clusters[i]) == 0:
                break
            if sc >= max_search:
                giveup = True
```

```
else:
    giveup = False
np.random.shuffle(clusters[i])
new_sample = clusters[i][0].copy()
new_sample.append(0)
indices = self.check_maj_sample(self.data.values,[new_sample])
overlap = False
for index in indices:
    if self.data.iloc[index]['class'] == 1:
        overlap = True
if overlap:
    sd = pd.DataFrame(data=[new_sample], columns=keys)
    sr = _MajDf[_MajDf.set_index(keys).index.isin(sd.set_index(keys).index)]
    if sr.shape[0] > 0:
        d_index = sr.index.values[0]
        _MajDf = _MajDf.drop(d_index)


        for j in range(len(clusters[i]) - 1):
            if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                np.delete(clusters[i], j + 1, 0)
        np.delete(clusters[i], 0, 0)
        _Maji -= 1
        sc = 1
    else:
        sc += 1
if not overlap and giveup:
```

```
            sd = pd.DataFrame(data=[new_sample], columns=keys)
            sr = _MajDf[_MajDf.set_index(keys).index.isin(sd.set_index(keys).index)]
            if sr.shape[0] > 0:
                d_index = sr.index.values[0]
                _MajDf = _MajDf.drop(d_index)

                for j in range(len(clusters[i]) - 1):
                    if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                        np.delete(clusters[i], j + 1, 0)
                np.delete(clusters[i], 0, 0)
                _Maji -= 1
                sc = 1
            else:
                sc += 1
        if not overlap:
            sc += 1
    if _MajDf.shape[0] == 0:
        raise ValueError("No Majority samples selected check your parameters")
    return _MajDf.values
```

I. EXPERIMENTAL PROCESS IMPLEMENTATION

```
import json
import time
```

```
import pandas as pd
import numpy as np
import sys
import os
import matplotlib.pyplot as plt
from sklearn.exceptions import UndefinedMetricWarning
from sklearn.metrics import *
from sklearn.neighbors import KNeighborsClassifier
from ugutils.adasyn import ADASYN
from ugutils.clus import CLUS
from ugutils.ehcbst import EHCBST
from ugutils.none import NONE
from ugutils.oss import OSS
from ugutils.ros import ROS
from ugutils.rus import RUS
from ugutils.sbc import SBC
from ugutils.smote import SMOTE

sys.path.append("..")
sys.path.append("../..")
from ugutils import *
from ugutils.hcbst import HCBST
from ugutils.cust import CUST

from sklearn import svm, metrics, tree
from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit, train_test_split, cross_val_score
from imblearn.metrics import geometric_mean_score
from sklearn import model_selection
from math import sqrt
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import f1_score
from sklearn.metrics import fbeta_score
import coloredlogs, logging
import warnings
#warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)


coloredlogs.install(level=logging.DEBUG, fmt='[%(levelname)s]%(asctime)s:%(message)s')
aucs =[]
mccs =[]
rocs =[]
specs =[]
sens =[]
accs =[]
gmeans =[]
algos =[]
tps =[]
```

```
fps =[]

tns =[]

fns =[]

fitted_models =[]

stage = 1


h = .02  # step size in the mesh


figure = plt.figure(figsize=(27, 9))



def val(X_train, y_train, train_index, test_index, r, keys, d, step, resampling_technique, resampling_technique_name):
    global cts, mccs, rocs, specs, sens, accs, gmeans, aucs, algos, fitted_models, tps, fps, tns, fns, f_measures
    global dataset_count, st_count, dataset_name


    X_train_val, X_test_val = X_train[train_index], X_train[test_index]
    y_train_val, y_test_val = y_train[train_index], y_train[test_index]
    T = np.zeros((X_train_val.shape[0], X_train_val.shape[1] + 1))
    T[:, :-1] = X_train_val
    T[:, -1] = y_train_val
    T = pd.DataFrame(T, columns=keys)
    mi = T.loc[T['class'] == 1]  # minority samples
    ma = T.loc[T['class'] == 0]  # majority samples
    start_time = time.time()
    s_min, s_maj = resampling_technique.resample(ma, mi)
    ct = time.time() - start_time
    if s_min is None or s_maj is None:
        print('sampling failed')
        s_maj, s_min = ma, mi


    final_train_sample = np.concatenate((s_maj, s_min), axis=0)
```

```
    final_train_sample = pd.DataFrame(final_train_sample,
                                      columns=keys)
    X_train_val = final_train_sample[final_train_sample.columns[0:d - 1]].values
    y_train_val = final_train_sample[final_train_sample.columns[d - 1]].values
    y_train_val = y_train_val.astype('int32')
    y_test_val = y_test_val.astype('int32')
    # iterate over classifiers
    for name, clf in zip(classifier_names, classifiers):
        logging.info("getting metrics for {4}--{2}--{0} val step {1} iter: {3}".format(name, step, resampling_technique_name,
i_step+1, dataset_name))
        percentage = ((d_step / dataset_count) * 10000 + (t_step / st_count) * 1000 +
                      ((i_step / 10) * 100) + ((v_step / 10) * 10)) / 100
        percentage = np.round(percentage, 2)
        logging.info("Percentage done {0}%".format(percentage))
        clf = clf.fit(X_train_val, y_train_val)
        pred = clf.predict(X_test_val.tolist())
        y = y_test_val.tolist()
        acc = accuracy_score(y, list(pred))
        tn, fp, fn, tp = confusion_matrix(y, pred).ravel()
        # mcc = (tp * tn - fp * fn) / (np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)))
        # matthews_corrcoef(y, pred)
        roc_auc = roc_auc_score(y, list(pred))
        fpr, tpr, thresholds = roc_curve(y, list(pred), pos_label=1)
        auc = metrics.auc(fpr, tpr)
        gmean = geometric_mean_score(y, list(pred))

        # mcc = matthews_corrcoef(y, list(pred))

        denom = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
        if denom == 0:
            mcc = 0
```

```
        else:
            mcc = (tp * tn - fp * fn) / denom

        sensitivity = tp / (tp + fn)
        specificity = tn / (tn + fp)
        f_measure = fbeta_score(y, list(pred), average='weighted, beta=0.5)

        logging.info("{7} --- acc: {0}, mcc:{1}, auc:{2}, gmean:{3}, sen:{4}, spec:{5}, roc:{6}"
                     .format(round(acc, 2), round(mcc, 2), round(auc, 2), round(gmean, 2),
                             round(sensitivity, 2), round(specificity, 2), round(roc_auc, 2),
                             resampling_technique_name))
        mccs.append(mcc)
        accs.append(acc)
        aucs.append(auc)
        gmeans.append(gmean)
        sens.append(sensitivity)
        specs.append(specificity)
        rocs.append(roc_auc)
        algos.append(name)
        fns.append(fn)
        tns.append(tn)
        fps.append(fp)
        tps.append(tp)
        fitted_models.append(clf)
        f_measures.append(f_measure)
        cts.append(ct)


def test(X_test, y_test, resampling_technique_name):
    global test_mccs, test_rocs, test_specs, test_sens, test_accs, \
        test_gmeans, test_aucs, test_algos, test_tps, test_fps, test_tns, test_fns, \
```

```
        test_f_measures
    global d_step, t_step, i_step, v_step, dataset_name


    step = 1
    for name, clf in zip(classifier_names, fitted_models):
        logging.info("getting metrics for {4}--{2}--{0} test step {1} iter: {3}".format(name, v_step, resampling_technique_name,
i_step+1, dataset_name))
        percentage = ((d_step / dataset_count) * 10000 + (t_step / st_count) * 1000 +
                      ((i_step / 10) * 100) + ((v_step / 10) * 10)) / 100
        percentage = np.round(percentage, 2)
        logging.info("Percentage done {0}%".format(percentage))
        step += 1
        pred = clf.predict(X_test.tolist())
        y = y_test.tolist()
        acc = accuracy_score(y, list(pred))
        tn, fp, fn, tp = confusion_matrix(y, pred).ravel()
        # mcc = (tp * tn - fp * fn) / (np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)))
        # matthews_corrcoef(y, pred)
        roc_auc = roc_auc_score(y, list(pred))
        fpr, tpr, thresholds = roc_curve(y, list(pred), pos_label=1)
        auc = metrics.auc(fpr, tpr)
        gmean = geometric_mean_score(y, list(pred))
        denom = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
        if denom == 0:
            mcc = 0
        else:
            mcc = (tp * tn - fp * fn) / denom


        sensitivity = tp / (tp + fn)
        specificity = tn / (tn + fp)
        f_measure = fbeta_score(y, list(pred), average='weighted, beta=0.5)
```

```
        test_mccs.append(mcc)
        test_accs.append(acc)
        test_aucs.append(auc)
        test_gmeans.append(gmean)
        test_sens.append(sensitivity)
        test_specs.append(specificity)
        test_rocs.append(roc_auc)
        test_algos.append(name)
        test_fns.append(fn)
        test_tns.append(tn)
        test_fps.append(fp)
        test_tps.append(tp)
        test_f_measures.append(f_measure)


def cross_val(data, ds_name, resampling_technique, resampling_technique_name):
    global cts, mccs, rocs, specs, sens, accs, gmeans, aucs, algos, fitted_models, tps, fps, tns, fns, f_measures
    global test_mccs, test_rocs, test_specs, test_sens, test_accs, test_gmeans, test_aucs, test_algos, \
        test_tps, test_fps, test_tns, test_fns, test_f_measures
    global stage, global_step
    global d_step, t_step, i_step, v_step
    global data_name, dataset_name
    dataset_name = ds_name
    i_step, v_step = 0, 0
    test_aucs =[]
    test_mccs =[]
    test_rocs =[]
    test_specs =[]
    test_sens =[]
    test_accs =[]
```

```
test_gmeans =[]

test_algos =[]

test_tps =[]

test_fps =[]

test_tns =[]

test_fns =[]

test_f_measures =[]

cts =[]

aucs =[]

mccs =[]

rocs =[]

specs =[]

sens =[]

accs =[]

gmeans =[]

algos =[]

tps =[]

fps =[]

tns =[]

fns =[]

f_measures =[]

Maj = data.loc[data['class'] == 0]  # Majority instances

Min = data.loc[data['class'] == 1]  # Minority instances

X = data[data.columns[0:data.shape[1] - 1]].values

y = data[data.columns[data.shape[1] - 1]].values

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.1, random_state=0)

sss = StratifiedShuffleSplit(test_size=0.1, random_state=0)

sss.get_n_splits(X, y)


counter = 0

keys = list(data.columns.values)
```

```
r = 9
d = data.shape[1]  # the dimension of the dataset

train_indices =[]
test_indices =[]
global_step = 1
seed = 456
i_step = 0
for i in range(10):
    print("Global step {0} for {1}".format(global_step, dataset_name))
    skf = StratifiedKFold(n_splits=10, random_state=global_step * seed, shuffle=True)
    step = 1
    v_step = 0
    for train_index, test_index in skf.split(X_train, y_train):
        train_indices.append(train_index)
        test_indices.append(test_index)
        logging.info(
            "Processing validation for dataset: {0} at global step: {1}, local step: {2}".format(dataset_name,
                                                                    global_step, step))
        percentage = ((d_step / dataset_count) * 10000 + (t_step / st_count) * 1000 +
                      ((i_step / 10) * 100) + ((v_step / 10) * 10)) / 100
        percentage = np.round(percentage, 2)
        logging.info("Percentage done {0}%".format(percentage))

        val(X_train, y_train, train_index, test_index, r, keys, d, step, resampling_technique,
            resampling_technique_name)
        logging.info(
            "Evaluating testing for dataset: {0} at global step: {1}, local step: {2}".format(dataset_name,
                                                                global_step, step))
        test(X_test, y_test, resampling_technique_name)
        step += 1
```

```
        v_step += 1
    global_step += 1
    i_step += 1

logging.info("Saving results to file for {0}__{1}".format(resampling_technique_name, dataset_name))
val_result = pd.DataFrame(algos, columns=['Classifier'])
# val_result = pd.DataFrame(accs, columns=['Accuracy'])
val_result['Accuracy'] = accs
val_result['AUC'] = aucs
val_result['ROC'] = rocs
val_result['Sensitivity'] = sens
val_result['Specificity'] = specs
val_result['F-measure'] = f_measures
val_result['G-Mean'] = gmeans
val_result['MCC'] = mccs
val_result['TP'] = tps
val_result['TN'] = tns
val_result['FP'] = fps
val_result['FN'] = fns
val_result['Sampling Time'] = cts

test_result = pd.DataFrame(test_algos, columns=['Classifier'])
test_result['Accuracy'] = test_accs
test_result['AUC'] = test_aucs
test_result['ROC'] = test_rocs
test_result['Sensitivity'] = test_sens
test_result['Specificity'] = test_specs
test_result['F-measure'] = test_f_measures
test_result['G-Mean'] = test_gmeans
test_result['MCC'] = test_mccs
test_result['TP'] = test_tps
```

```
    test_result['TN'] = test_tns
    test_result['FP'] = test_fps
    test_result['FN'] = test_fns

    with pd.ExcelWriter(
        {0}result_{1}_{2}.xlsx'.format(data_dir, resampling_technique_name, dataset_name)) as writer:  # doctest: +SKIP
        val_result.to_excel(writer, sheet_name='val_results')
        test_result.to_excel(writer, sheet_name='test_results')
        writer.close()

    summary_result = pd.DataFrame([], columns=[])
    result_stats = pd.DataFrame([], columns=[])
    classifiers =[]
    header = False if stage > 1 else True

    for classifier in classifier_names:
        metrics_c = val_result.loc[val_result['Classifier'] == classifier]
        means = metrics_c.mean(skipna=True)
        means['Dataset'] = dataset_name
        means['Sampling Technique'] = resampling_technique_name
        means['Classifier'] = classifier
        summary_result = summary_result.append(means, ignore_index=True)

        stats = metrics_c.describe()
        stats['Dataset'] = dataset_name
        stats['Sampling Technique'] = resampling_technique_name
        stats['Classifier'] = classifier
        result_stats = result_stats.append(stats)

    append_df_to_excel('{0}result_summary.xlsx'.format(data_dir), summary_result, sheet_name='val_summary,
header=header)
```

```
    append_df_to_excel('{0}result_summary.xlsx'.format(data_dir), result_stats, sheet_name='val_stats, header=header)

    summary_result = pd.DataFrame([], columns=[])
    result_stats = pd.DataFrame([], columns=[])
    for classifier in classifier_names:
        metrics_c = test_result.loc[test_result['Classifier'] == classifier]
        means = metrics_c.mean(skipna=True)
        means['Dataset'] = dataset_name
        means['Sampling Technique'] = resampling_technique_name
        means['Classifier'] = classifier
        summary_result = summary_result.append(means, ignore_index=True)

        stats = metrics_c.describe()
        stats['Dataset'] = dataset_name
        stats['Sampling Technique'] = resampling_technique_name
        stats['Classifier'] = classifier
        result_stats = result_stats.append(stats)

    append_df_to_excel('{0}result_summary.xlsx'.format(data_dir), summary_result, sheet_name='test_summary,
header=header)
    append_df_to_excel('{0}result_summary.xlsx'.format(data_dir), result_stats, sheet_name='test_stats, header=header)

    logging.info("Done working on {0}".format(dataset_name))
    # print("done!")
    stage += 1


classifier_names =["Nearest Neighbors", "Linear SVM",
                   "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
                   "Naive Bayes", "QDA"]
```

```
classifiers =[
    KNeighborsClassifier(3),
    # SVC(kernel="linear", C=0.025),
    # SVC(gamma=2, C=1),
    LinearSVC(random_state=0, tol=1e-5),
    # GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
resampling_technique_names =['HCBST']
resampling_techniques =[
    #NONE(),
    #ADASYN({'r': 10}),
    #ROS({'r': 10}),
    #RUS({'r': 0.05}),
    #OSS({'k_neigh': 1}),
    #SMOTE({'r': 10,k_neigh': 5}),
    #SBC({'r': 50,k_clusters': 50}),
    #CLUS({'k_neigh': 5}),
    EHCBST({'ru': 10,'km': 15,'ro': 10,'pm': 0.1,'rmin': 10}),
    #CUST({'r': 50,k_clusters': 50})
]

d_step, t_step, i_step, v_step = 0, 0, 0, 0
dataset_count, st_count = 0, 0

run_id = 63
data_dir = "D:\\data_run_{0}\\".format(run_id)
```

```
def main():
    global stage, data_dir, resampling_techniques, d_step, t_step, i_step, v_step, dataset_count, st_count
    datasets =[]
    stage = 1
    if not os.path.exists(data_dir):
        os.makedirs(data_dir)
    f = open("{0}params.txt".format(data_dir), "w+")
    params =and .join(json.dumps(e.get_params()) for e in resampling_techniques)
    #params = str(resampling_techniques[0])
    #f.write("{0}\n".format(param_txt))
    f.write("{0}\n".format(params))
    f.close()

    dataset_names =['yeast-2_vs_8,glass2,ecoli4,CM1,KC3,MC2,MW1,PC1,PC2,abalone9v18,abalone19']
    logging.info("Preparing Datasets!")

    DATA_PATH = os.path.join('../../data, "yeast-2_vs_8.csv")
    data = pd.read_csv(DATA_PATH)
    data.loc[data['class'] ==negative,class'] = 0
    data.loc[data['class'] ==positive,class'] = 1
    datasets.append(data)
    DATA_PATH = os.path.join('../../data, "glass2.csv")
    data = pd.read_csv(DATA_PATH)
    data.loc[data['class'] ==negative,class'] = 0
    data.loc[data['class'] ==positive,class'] = 1
    datasets.append(data)
    DATA_PATH = os.path.join('../../data, "ecoli4.csv")
    data = pd.read_csv(DATA_PATH)
    data.loc[data['class'] ==negative,class'] = 0
    data.loc[data['class'] ==positive,class'] = 1
    datasets.append(data)
```

```
DATA_PATH = os.path.join('../../data, "CM1.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "KC3.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "MC2.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "MW1.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "PC1.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "PC2.csv")
data = pd.read_csv(DATA_PATH)
datasets.append(data)
DATA_PATH = os.path.join('../../data, "abalone9v18.csv")
data = pd.read_csv(DATA_PATH)
data.loc[data['class'] ==negative,class'] = 0
data.loc[data['class'] ==positive,class'] = 1
datasets.append(data)
DATA_PATH = os.path.join('../../data, "abalone19.csv")
data = pd.read_csv(DATA_PATH)
data.loc[data['sex'] ==M,sex'] = 0.3
data.loc[data['sex'] ==F,sex'] = 0.2
data.loc[data['sex'] ==I,sex'] = 0.1
data.loc[data['class'] ==negative,class'] = 0
data.loc[data['class'] ==positive,class'] = 1
datasets.append(data)
```

```
    dataset_count = len(datasets)
    st_count = len(resampling_techniques)
    d_step = 0
    for name, data in zip(dataset_names, datasets):
        t_step = 0
        for resampling_technique_name, resampling_technique in zip(resampling_technique_names, resampling_techniques):
            logging.info("Begin dataset: {0}, resampling technique: {1}".format(name, resampling_technique_name))
            cross_val(data, name, resampling_technique, resampling_technique_name)
            t_step += 1
        d_step += 1

    # cross_val(data)


def append_df_to_excel(filename, df, sheet_name='Sheet1, startrow=None,
                       truncate_sheet=False, header=False,
                       retry_count=0, **to_excel_kwargs):
    """
    Append a DataFrame[df] to existing Excel file[filename]
    into[sheet_name] Sheet.
    If[filename] doesn't exist, then this function will create it.

    Parameters:
      filename : File path or existing ExcelWriter
                 (Example:/path/to/file.xlsx')
      df : dataframe to save to workbook
      sheet_name : Name of sheet which will contain DataFrame.
                   (default:Sheet1')
      startrow : upper left cell row to dump data frame.
                 Per default (startrow=None) calculate the last row
```

```
            in the existing DF and write to the next row...
    truncate_sheet : truncate (remove and recreate)[sheet_name]
                before writing DataFrame to Excel file
    to_excel_kwargs : arguments which will be passed to `DataFrame.to_excel()`
                [can be dictionary]

    Returns: None
    """
    from openpyxl import load_workbook

    # ignore[engine] parameter if it was passed
    if'engine'in to_excel_kwargs:
        to_excel_kwargs.pop('engine')

    writer = pd.ExcelWriter(filename, engine='openpyxl')

    try:
        # try to open an existing workbook
        writer.book = load_workbook(filename)

        # get the last row in the existing Excel sheet
        # if it was not specified explicitly
        if startrow is None and sheet_name in writer.book.sheetnames:
            startrow = writer.book[sheet_name].max_row

        # truncate sheet
        if truncate_sheet and sheet_name in writer.book.sheetnames:
            # index of[sheet_name] sheet
            idx = writer.book.sheetnames.index(sheet_name)
            # remove[sheet_name]
            writer.book.remove(writer.book.worksheets[idx])
```

```
            # create an empty sheet[sheet_name] using old index
            writer.book.create_sheet(sheet_name, idx)

        # copy existing sheets
        writer.sheets = {ws.title: ws for ws in writer.book.worksheets}
    except FileNotFoundError:
        # file does not exist yet, we will create it
        pass
    except EOFError:
        retry_count += 1
        if retry_count < 10:
            logging.info("File in use retrying {0}%".format(retry_count))
            time.sleep(5)
            append_df_to_excel(filename, df, sheet_name=sheet_name, startrow=startrow,
                               truncate_sheet=truncate_sheet, header=header,
                               retry_count=retry_count)

        pass

    if startrow is None:
        startrow = 0

    # write out the new sheet
    df.to_excel(writer, sheet_name, startrow=startrow, header=header, **to_excel_kwargs)

    # save the workbook
    writer.save()


if __name__ == "__main__":
    main()
```

II. FULL IMPLEMENTATION OF HCBST

```
import pandas as pd
import numpy as np
import sys
from random import seed
import random
from random import choice

from sklearn.metrics import euclidean_distances
from sklearn.neighbors import NearestNeighbors
from ugutils import *

sys.path.append("..")



class HCBST:
    def __init__(self, params=None):
        if params is None:
            params = {'ru': 1,'km': 3,'ro': 1,'kn': 3,'rmin': None,'pm': None}
        self.Maj = None
        self.Min = None
        self.data = None
        self.params = params
        self.ru = params['ru']
        self.ro = params['ro']
        self.km = params['km']
        self.rmin = params['rmin']
        self.pm = params['pm']
        self.beta = None
```

```
def get_params(self):
    return self.params


def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    self.data = self.Maj.append(self.Min)
    min_samples = self.oversample()
    if self.rmin is not None and self.rmin > self.Maj.shape[0]/self.Min.shape[0]:
        self.pm = None
        if self.ru > self.Maj.shape[0]/self.Min.shape[0]:
            self.ru = self.Maj.shape[0]/self.Min.shape[0]
        pass
    else:
        pass
    maj_samples = self.undersample()
    return min_samples, maj_samples


def oversample(self):
    _Min = self.Min[self.Min.columns[0:self.Min.shape[1]-1]].values
    _Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1]-1]].values
    y = self.Min.iloc[:, self.Min.shape[1] - 1]
    No = self.ro * (len(_Min)) - (len(_Min))
    No = int(np.round(No))
    print(No)
    if No == 0:
        return self.Min.values
    o = choice(self.Min.iloc[:, 0:self.Min.shape[1]-1].values)
    o = o.reshape(1, self.Min.iloc[:, 0:self.Min.shape[1]-1].shape[1])
    final_samples = self.Min.values.copy().tolist()
    m_i =[]
```

```
neighs =[]
neigh_dists =[]
for i in range(len(self.Min)):
    term = self.Min.iloc[i, 0:self.Min.shape[1]-1]
    term = pd.DataFrame(term)
    knn = iknn.KNN(o, 5)
   [pred, neigh, neigh_dist] = knn.fit(self.Min.iloc[:, 0:self.Min.shape[1]-1].values, y.values, term)
    m_i.append(np.mean(neigh_dist))
    neighs.append(neigh)
    neigh_dists.append(neigh_dist)
sigma = np.mean(m_i) + np.std(m_i)
# generating alphas
for m in range(No):
    sigma_neigh =[]
    max_search = 20
    search_count = 0
    s_i = 0
    x = 0
    y = 0
    z = 0
    while len(sigma_neigh) < 2 and search_count < max_search:
        a = round(random.uniform(0, 1), 1)
        x = random.uniform(0, a)
        y = random.uniform(0, 1 - a)
        z = 1 - (x + y)
        s_i = choice(self.Min.values)
        index_s_i = np.bincount(np.where(self.Min.values == s_i)[0]).argmax()
        neighd_s_i = neigh_dists[index_s_i]  # neigh_dists[index_s_i]
        neigh_s_i = neighs[index_s_i]
        sigma_neigh =[]
        for i in range(len(neighd_s_i)):
```

```
            if len(sigma_neigh) > 2:
                continue
            if neighd_s_i[i] < sigma:
                sigma_neigh.append(neigh_s_i[i])

        search_count += 1

    #print(sigma_neigh)
    if len(sigma_neigh) >= 2:
        s_i_new = x * s_i + y * self.Min.values[sigma_neigh[0]] + z * self.Min.values[sigma_neigh[1]]

        u = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[0]]])
        h = self.check_min_sample(self.Maj.values,[self.Min.values[sigma_neigh[1]]])
        g = (u+h)/2
        #print(g)
        f = self.check_min_sample(self.Maj.values,[s_i_new])
        #print(f)
        #print("----------")
        if f > g:
            final_samples.append(s_i_new)
            # print("added")
        else:
            # print("Not added")
            pass
    final_samples = np.array(final_samples)
    return np.concatenate((self.Min.values, final_samples), axis=0)

def get_average_distances(self, samples):
    return np.mean(euclidean_distances(samples))
def check_min_sample(self, samples, sample):
    nbrs = NearestNeighbors(n_neighbors=1, algorithm='ball_tree').fit(samples)
```

```
        distances, indices = nbrs.kneighbors(sample)
        return distances[0][0]
    def check_maj_sample(self, samples, sample):
        nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree').fit(samples)
        distances, indices = nbrs.kneighbors(sample)
        return indices[0]
    def stack_undersample(self):
        print("stack_triggered")
        _Min = self.Min[self.Min.columns[0:self.Min.shape[1] - 1]].values
        _Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1] - 1]].values
        _MajDf = _Maj
        _Maj_new = {}
       [_Min, _Maj] = self.remove_inconsistent(_Min, _Maj)
        keys = list(self.data.columns.values)
        temp = np.zeros((len(_Maj), _Maj.shape[1]+1))
        temp[:, :-1] = _Maj
        _MajDf = pd.DataFrame(data=temp, columns=keys)
        origin = np.zeros(shape=(1, _Maj.shape[1]))
        km = ik_means.K_Means(originPoint=origin[0], k=self.km)
        km.fit(_Maj)
        clusters = km.get_clusters()
        max_search = 25
        for i in range(self.km):
            _Maji = self.pm * len(clusters[i])
            sc = 1
            giveup = False
            while _Maji > 0:
                if len(clusters[i]) == 0:
                    break
                if sc >= max_search:
                    giveup = True
```

```
else:
    giveup = False
np.random.shuffle(clusters[i])
new_sample = clusters[i][0].copy()
new_sample.append(0)
indices = self.check_maj_sample(self.data.values,[new_sample])
overlap = False
for index in indices:
    if self.data.iloc[index]['class'] == 1:
        overlap = True
if overlap:
    sd = pd.DataFrame(data=[new_sample], columns=keys)
    sr = _MajDf[_MajDf.set_index(keys).index.isin(sd.set_index(keys).index)]
    if sr.shape[0] > 0:
        d_index = sr.index.values[0]
        _MajDf = _MajDf.drop(d_index)


        for j in range(len(clusters[i]) - 1):
            if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                np.delete(clusters[i], j + 1, 0)
        np.delete(clusters[i], 0, 0)
        _Maji -= 1
        sc = 1
    else:
        sc += 1
if not overlap and giveup:
    sd = pd.DataFrame(data=[new_sample], columns=keys)
    sr = _MajDf[_MajDf.set_index(keys).index.isin(sd.set_index(keys).index)]
    if sr.shape[0] > 0:
        d_index = sr.index.values[0]
        _MajDf = _MajDf.drop(d_index)
```

```
                for j in range(len(clusters[i]) - 1):
                    if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                        np.delete(clusters[i], j + 1, 0)
                np.delete(clusters[i], 0, 0)
                _Maji -= 1
                sc = 1
            else:
                sc += 1
        if not overlap:
            sc += 1
    if _MajDf.shape[0] == 0:
        raise ValueError("No Majority samples selected check your parameters")
    return _MajDf.values


def undersample(self):
    if self.pm is None and self.ru == self.Maj.shape[0]/self.Min.shape[0]:
        return self.Maj
    if self.pm is not None:
        return self.stack_undersample()
    _Min = self.Min[self.Min.columns[0:self.Min.shape[1]-1]].values
    _Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1]-1]].values
    _Maj_new = {}
    [_Min, _Maj] = self.remove_inconsistent(_Min, _Maj)
    samples = 0
    origin = np.zeros(shape=(1, _Maj.shape[1]))
    km = ik_means.K_Means(originPoint=origin[0], k=self.km)
    km.fit(_Maj)
    clusters = km.get_clusters()
    max_search = 50
    for i in range(self.km):
```

```
        _Maji = self.ru * (len(_Min) / len(_Maj)) * len(clusters[i])
        sc = 1
        giveup = False
        while _Maji > 0:
            if len(clusters[i]) == 0:
                break
            if sc >= max_search:
                #print("max search")
                giveup = True
            np.random.shuffle(clusters[i])
            new_sample = clusters[i][0].copy()
            new_sample.append(0)
            indices = self.check_maj_sample(self.data.values,[new_sample])
            overlap = False
            for index in indices:
                if self.data.iloc[index]['class'] == 1:
                    sc += 1
                    overlap = True
            if not overlap or giveup:
                _Maj_new[samples] = clusters[i][0]
                samples += 1
                for j in range(len(clusters[i]) - 1):
                    if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
                        np.delete(clusters[i], j + 1, 0)
                np.delete(clusters[i], 0, 0)
                _Maji -= 1
                giveup = False
                sc = 1
    _Maj_new = np.array(list(_Maj_new.values()))
    if _Maj_new.shape[0] == 0:
        raise ValueError("No Majority samples selected check your parameters")
```

```
    temp = np.zeros((len(_Maj_new), _Maj.shape[1]+1))
    temp[:, :-1] = _Maj_new
    _Maj_new = temp
    return temp


def remove_inconsistent(self, X, Y):
    for x in range(len(X) - 1):
        for y in range(len(Y) - 1):
            if self.euclidean_distance(X[x], Y[y], Y.shape[1] - 1) == 0:
                Y.drop([y])
    return X, Y


def euclidean_distance(self, data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])
    return np.sqrt(distance)


def total_runtime(self):
```

None

III. IMPLEMENTATION OF CUST

```
import pandas as pd
import numpy as np
import sys
from random import seed
import random
from random import choice
from ugutils import *

sys.path.append("..")



class CUST:
    def __init__(self, params=None):
        if params is None:
            params = {'r': 1,k_clusters': 3}
        self.Maj = None
        self.Min = None
        self.r = params['r']
        self.k_clusters = params['k_clusters']


    def resample(self, majority_samples, minority_samples):
        self.Maj = majority_samples
        self.Min = minority_samples
        min_samples = self.Min.values
        maj_samples = self.undersample()
        return min_samples, maj_samples


    def undersample(self):
```

```
_Min = self.Min[self.Min.columns[0:self.Min.shape[1] - 1]].values
_Maj = self.Maj[self.Maj.columns[0:self.Maj.shape[1] - 1]].values
# _Min = self.Min.iloc[:, 0:self.Min.shape[1]].values
# _Maj = self.Maj.iloc[:, 0:self.Maj.shape[1]].values

_Maj_new = {}

[_Min, _Maj] = self.remove_inconsistent(_Min, _Maj)



samples = 0

origin = np.zeros(shape=(1, _Maj.shape[1]))
#km = ik_means.K_Means(origin[0], k)
km = k_means.K_Means(self.k_clusters)
km.fit(_Maj)

clusters = km.get_clusters()

# print(km.get_clusters()[0])
for i in range(self.k_clusters):
    # Maji = r * (len(Min) / len(Maj)) * len(clusters[i])
    _Maji = self.r * (len(_Min) / len(_Maj)) * len(clusters[i])
    while _Maji > 0:
        if len(clusters[i]) == 0:
            break
        np.random.shuffle(clusters[i])
        _Maj_new[samples] = clusters[i][0]
        samples += 1
        for j in range(len(clusters[i]) - 1):
            if self.euclidean_distance(clusters[i][0], clusters[i][j + 1], _Maj.shape[1] - 1) == 0:
```

```
                np.delete(clusters[i], j + 1, 0)
            np.delete(clusters[i], 0, 0)
            _Maji -= 1
    _Maj_new = np.array(list(_Maj_new.values()))
    temp = np.zeros((len(_Maj_new), _Maj.shape[1] + 1))
    temp[:, :-1] = _Maj_new
    _Maj_new = temp
    return temp


def remove_inconsistent(self, X, Y):
    for x in range(len(X) - 1):
        for y in range(len(Y) - 1):
            if self.euclidean_distance(X[x], Y[y], Y.shape[1] - 1) == 0:
                Y.drop([y])
    return X, Y


def euclidean_distance(self, data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])
    return np.sqrt(distance)


def total_runtime(self):
    None
```

IV. IMPLEMENTATION OF CLUS

```
import pandas as pd
import os
import sys
import numpy as np
from sklearn.svm import LinearSVC

sys.path.append("..")
from ugutils import *

from imblearn.over_sampling import SMOTE
from collections import Counter


N = 1.5
k = 3


class CLUS:

    def __init__(self, params=None):
        if params is None:
            params = {'k_neigh': 3}
        self.N = 2 / 3
        self.Maj = None
        self.Min = None
        self.k_neigh = params['k_neigh']
        self.k_clusters = 3
```

```
def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    min_samples, maj_samples = self.do_clus()
    return min_samples, maj_samples


def do_clus(self):
    data = self.Maj.append([self.Min])
    d = data.shape[1]
    km = k_means.K_Means(self.k_clusters)
    km.fit(data[data.columns[0:d - 1]].values)


    clusters = km.get_clusters()
    c = pd.DataFrame(clusters[0][0:], columns=data.columns[:d - 1])
    c1 = data[data.set_index(data.columns[:d - 1].values.tolist()).index.isin(
        c.set_index(data.columns[:d - 1].values.tolist()).index)]
    # c1 = data.loc[(data.index.isin(c1.index))]
    c1Maj = c1.loc[c1['class'] == 0]
    c1Min = c1.loc[c1['class'] == 1]
    # print(c1Min)
    c = pd.DataFrame(clusters[1][0:], columns=data.columns[:d - 1])
    c2 = data[data.set_index(data.columns[:d - 1].values.tolist()).index.isin(
        c.set_index(data.columns[:d - 1].values.tolist()).index)]
    c2Maj = c2.loc[c2['class'] == 0]
    c2Min = c2.loc[c2['class'] == 1]
    # print(c2Min)
    c = pd.DataFrame(clusters[2][0:], columns=data.columns[:d - 1])
    c3 = data[data.set_index(data.columns[:d - 1].values.tolist()).index.isin(
        c.set_index(data.columns[:d - 1].values.tolist()).index)]
    c3Maj = c3.loc[c3['class'] == 0]
    c3Min = c3.loc[c3['class'] == 1]
```

```
##New Subsets
ss1 = c1Min.append([c2Maj])
ss2 = c2Min.append([c3Maj])
ss3 = c1Maj.append([c3Min])

c11 = pd.DataFrame([], columns=['class'])
c12, c13 = c11,c11

sm = SMOTE(random_state=0, sampling_strategy=self.N, k_neighbors=self.k_neigh)
if c1Min.shape[0] >= self.k_neigh+1 and c2Maj.shape[0] > 0 and self.N > c1Min.shape[0]/c2Maj.shape[0]:
    ss = ss1
    synth_x, synth_y = sm.fit_resample(ss.iloc[:, 0:ss.shape[1] - 1].values, ss.iloc[:, ss.shape[1] - 1].values)
    synth = pd.DataFrame(synth_x, columns=data.columns[:d - 1])
    synth['class'] = synth_y
    ss1 = synth
    #ss1 = synth.loc[synth['class'] == 1].append([c2Maj])
    print("ss1 done: {0}".format(ss1.shape))

if c2Min.shape[0] >= self.k_neigh+1 and c3Maj.shape[0] > 0 and self.N > c2Min.shape[0]/c3Maj.shape[0]:
    ss = ss2

    synth_x, synth_y = sm.fit_resample(ss.iloc[:, 0:ss.shape[1] - 1].values, ss.iloc[:, ss.shape[1] - 1].values)
    synth = pd.DataFrame(synth_x, columns=data.columns[:d - 1])
    synth['class'] = synth_y
    ss2 = synth
    #ss2 = synth.loc[synth['class'] == 1].append([c3Maj])
    print("ss2 done: {0}".format(ss2.shape))

if c3Min.shape[0] >= self.k_neigh+1 and c1Maj.shape[0] > 0 and self.N > c3Min.shape[0]/c1Maj.shape[0]:
    ss = ss3
```

```
    synth_x, synth_y = sm.fit_resample(ss.iloc[:, 0:ss.shape[1] - 1].values, ss.iloc[:, ss.shape[1] - 1].values)
    synth = pd.DataFrame(synth_x, columns=data.columns[:d - 1])
    synth['class'] = synth_y
    ss3 = synth
    #ss3 = synth.loc[synth['class'] == 1].append([c1Maj])
    print("ss3 done: {0}".format(ss3.shape))

from sklearn import svm
clf = LinearSVC(random_state=0, tol=1e-5)

scores =[]
sample_sets = {}

if (ss1.loc[ss1['class'] == 1].shape[0] <= self.k_neigh) or ss1.loc[ss1['class'] == 0].shape[0] == 0:
    score1 = 0
else:
    clf.fit(ss1[ss1.columns[0:d - 1]].values, ss1.iloc[:, d - 1].tolist())
    score1 = clf.score(ss1[ss1.columns[0:d - 1]].values, ss1.iloc[:, d - 1].tolist())
    score1 *= ss1.loc[ss1['class'] == 0].shape[0]*score1

scores.append(score1)
sample_sets[0] = ss1
if (ss2.loc[ss2['class'] == 1].shape[0] <= self.k_neigh) or ss2.loc[ss2['class'] == 0].shape[0] == 0:
    score2 = 0
else:
    clf.fit(ss2[ss2.columns[0:d - 1]].values, ss2.iloc[:, d - 1].tolist())
    score2 = clf.score(ss2[ss2.columns[0:d - 1]].values, ss2.iloc[:, d - 1].tolist())
    score2 *= ss2.loc[ss2['class'] == 0].shape[0] * score2

scores.append(score2)
sample_sets[1] = ss2
```

```
if (ss3.loc[ss3['class'] == 1].shape[0] <= self.k_neigh) or ss3.loc[ss3['class'] == 0].shape[0] == 0:
    score3 = 0
else:
    clf.fit(ss3[ss3.columns[0:d - 1]].values, ss3.iloc[:, d - 1].tolist())
    score3 = clf.score(ss3[ss3.columns[0:d - 1]].values, ss3.iloc[:, d - 1].tolist())
    score3 *= ss3.loc[ss3['class'] == 0].shape[0] * score3
scores.append(score3)
sample_sets[2] = ss3


sample_to_use = None


for score, sample in zip(scores, sample_sets):
    if score == max(scores):
        print("sample {0} won".format(sample))
        sample_to_use = sample_sets[sample]
        break


Min = sample_to_use.loc[sample_to_use['class'] == 1]  # Minority instances
Maj = sample_to_use.loc[sample_to_use['class'] == 0]  # Majority instances
print("final select {0}".format(sample_to_use.shape))
print("c1 {0}".format(Counter(c1['class'].values)))
print("ss1 {0}".format(Counter(ss1['class'].values)))
print("c2 {0}".format(Counter(c2['class'].values)))
print("ss2 {0}".format(Counter(ss2['class'].values)))
print("c3 {0}".format(Counter(c3['class'].values)))
print("ss3 {0}".format(Counter(ss3['class'].values)))
print("final select {0}".format(Counter(sample_to_use['class'].values)))
print(scores)
if Maj.shape[0] <5:
    return None, None
if Min.shape[0] <2:
```

```
        return None, None


    return Min, Maj
```

V. IMPLEMENTATION OF SBC

```
import math
import pandas as pd
import os
import sys
import numpy as np
sys.path.append("..")
from ugutils import *
from collections import Counter


class SBC:
  def __init__(self, params=None):
    if params is None:
      params = {'r': 1,k_clusters': 3}
    self.Maj = None
    self.Min = None
    self.imbalance_ratio = params['r']
    self.k_clusters = params['k_clusters']


  def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    min_samples, maj_samples = self.do_sbc()
```

```
    return min_samples, maj_samples


def do_sbc(self):
    data = self.Maj.append([self.Min])
    d = data.shape[1]
    km = k_means.K_Means(self.k_clusters)
    km.fit(data[data.columns[0:d - 1]].values)
    clusters = km.get_clusters()
    m = self.imbalance_ratio
    sum_ratios = 0
    cMins = {}
    cMaxs = {}
    for i in range(len(clusters)):
        c = pd.DataFrame(clusters[i][0:], columns=data.columns[:d - 1])
        c1 = data[data.set_index(data.columns[:d - 1].values.tolist()).index.isin(
            c.set_index(data.columns[:d - 1].values.tolist()).index)]
        c1Maj = c1.loc[c1['class'] == 0]
        c1Min = c1.loc[c1['class'] == 1]
        cMins[i] = c1Min
        cMaxs[i] = c1Maj
        SizeMaj = cMaxs[i].shape[0]
        SizeMin = cMins[i].shape[0]
        if SizeMin == 0:
            SizeMin = 1
        r = SizeMaj / SizeMin
        #r = c1Maj.shape[0] / c1Min.shape[0]
        sum_ratios += r

    final_samples = pd.DataFrame([], columns=data.columns[:d])
    for i in range(len(clusters)):
        SizeMaj = cMaxs[i].shape[0]
```

```
        SizeMin = cMins[i].shape[0]
        if SizeMin == 0:
            SizeMin = 1
        r = SizeMaj / SizeMin
        SampleSizeMaj = (m * self.Min.shape[0]) * (r / sum_ratios)
        if SampleSizeMaj == 0:
            SampleSizeMaj = 1
        if SampleSizeMaj > cMaxs[i].shape[0]:
            final_samples = final_samples.append(cMaxs[i])
        else:
            final_samples = final_samples.append(cMaxs[i].sample(n=math.ceil(SampleSizeMaj)))



        # print(math.ceil(SampleSizeMaj))
        # print("*{0},{1}*".format(SizeMaj,SizeMin))
        # randomly generate SampleSizeMaj instances from cMaxs[i]


    #print(final_samples.shape)
    #print(self.Min.shape)
    # merge with minority samples
    #final_samples = final_samples.append(self.Min)

    Min = self.Min  # Minority instances
    Maj = final_samples  # Majority instances

    return Min, Maj
```

VI. EXTENSION OF ROS FROM SCIKIT-LEARN

```
import numpy as np
from imblearn.over_sampling import RandomOverSampler
import pandas as pd

class ROS:

    def __init__(self, params=None):
        if params is None:
            params = {'r': 1}
        self.Maj = None
        self.Min = None
        self.params = params
        self.r = params['r']

    def resample(self, majority_samples, minority_samples):
        self.Maj = majority_samples
        self.Min = minority_samples
        self.r = min(1, (self.Min.shape[0] / self.Maj.shape[0]) * (1 + self.params['r']))
        min_samples, maj_samples = self.do_sampling()
        return min_samples, maj_samples

    def do_sampling(self):
        data = self.Maj.append([self.Min])
        d = data.shape[1]
        res = RandomOverSampler(random_state=0, sampling_strategy=self.r)
        res_x, res_y = res.fit_resample(data.iloc[:, 0:data.shape[1] - 1].values, data.iloc[:, data.shape[1] - 1].values)
        res = pd.DataFrame(res_x, columns=data.columns[:d - 1])
        res['class'] = res_y
```

```
    Min = res.loc[res['class'] == 1]
    Maj = res.loc[res['class'] == 0]
    return Min, Maj
```

VII. EXTENSION OF RUS FROM SCIKIT-LEARN

```
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd

class RUS:

  def __init__(self, params=None):
    if params is None:
      params = {'r': 1}
    self.Maj = None
    self.Min = None
    self.params = params
    self.r = params['r']

  def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    self.r = min(1, (self.Min.shape[0] / self.Maj.shape[0]) * (1 / self.params['r']))
    min_samples, maj_samples = self.do_sampling()
    return min_samples, maj_samples

  def do_sampling(self):
    data = self.Maj.append([self.Min])
    d = data.shape[1]
```

```
    res = RandomUnderSampler(random_state=0, sampling_strategy=self.r)
    res_x, res_y = res.fit_resample(data.iloc[:, 0:data.shape[1] - 1].values, data.iloc[:, data.shape[1] - 1].values)
    res = pd.DataFrame(res_x, columns=data.columns[:d - 1])
    res['class'] = res_y
    Min = res.loc[res['class'] == 1]
    Maj = res.loc[res['class'] == 0]
    return Min, Maj
```

VIII. EXTENSION OF OSS FROM SCIKIT-LEARN

```
import numpy as np
from imblearn.under_sampling import OneSidedSelection
import pandas as pd

class OSS:

  def __init__(self, params=None):
    if params is None:
      params = {'k_neigh': 1}
    self.Maj = None
    self.Min = None
    self.k_neigh = params['k_neigh']

  def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    min_samples, maj_samples = self.do_sampling()
    return min_samples, maj_samples

  def do_sampling(self):
```

```
        data = self.Maj.append([self.Min])
        d = data.shape[1]
        res = OneSidedSelection(random_state=0, n_neighbors=self.k_neigh)
        res_x, res_y = res.fit_resample(data.iloc[:, 0:data.shape[1] - 1].values, data.iloc[:, data.shape[1] - 1].values)
        res = pd.DataFrame(res_x, columns=data.columns[:d - 1])
        res['class'] = res_y
        Min = res.loc[res['class'] == 1]
        Maj = res.loc[res['class'] == 0]
        return Min, Maj
```

IX. EXTENSION OF SMOTE FROM SCIKIT-LEARN

```
import numpy as np
from imblearn.over_sampling import SMOTE as smote
import pandas as pd


class SMOTE:


  and
   params takes an array of values
  and
   def __init__(self, params=None):
      if params is None:
          params = {'r': 1,k_neigh': 5}
      self.Maj = None
      self.Min = None
      self.params = params
      self.r = params['r']
      self.k_neigh = params['k_neigh']
```

```
def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    self.r = min(1, (self.Min.shape[0] / self.Maj.shape[0]) * (1 + self.params['r']))
    min_samples, maj_samples = self.do_smote()
    return min_samples, maj_samples


def do_smote(self):
    data = self.Maj.append([self.Min])
    d = data.shape[1]
    sm = smote(random_state=0, sampling_strategy=self.r, k_neighbors=self.k_neigh)
    synth_x, synth_y = sm.fit_resample(data.iloc[:, 0:data.shape[1] - 1].values, data.iloc[:, data.shape[1] - 1].values)
    synth = pd.DataFrame(synth_x, columns=data.columns[:d - 1])
    synth['class'] = synth_y
    Min = synth.loc[synth['class'] == 1]
    Maj = synth.loc[synth['class'] == 0]
    return Min, Maj
```

X. EXTENSION OF ADASYN FROM SCIKIT-LEARN

```
import numpy as np
from imblearn.over_sampling import ADASYN as Adasyn
import pandas as pd


class ADASYN:


    def __init__(self, params=None):
        if params is None:
```

```
        params = {'r': 1}
    self.Maj = None
    self.Min = None
    #self.imbalance_ratio = 1/params['r']
    self.params = params
    self.r = 1
    self.alpha = 0.1

def resample(self, majority_samples, minority_samples):
    self.Maj = majority_samples
    self.Min = minority_samples
    self.r = min(1, (self.Min.shape[0] / self.Maj.shape[0]) * (1 + self.params['r'] + self.alpha))
    print("ratio_ {0}---{1}".format(self.params['r'],self.r))
    print(self.Maj.shape)
    print(self.Min.shape)
    min_samples, maj_samples = self.do_sampling()
    return min_samples, maj_samples

def do_sampling(self):
    data = self.Maj.append([self.Min])
    d = data.shape[1]
    res = Adasyn(random_state=0, sampling_strategy=self.r)
    res_x, res_y = res.fit_resample(data.iloc[:, 0:data.shape[1] - 1].values, data.iloc[:, data.shape[1] - 1].values)
    res = pd.DataFrame(res_x, columns=data.columns[:d - 1])
    res['class'] = res_y
    Min = res.loc[res['class'] == 1]
    Maj = res.loc[res['class'] == 0]
    return Min, Maj
```

APPENDIX B

Table B.1 AUC Performance using KNN

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.7048	0.4346	0.6169	0.6500	0.4832	0.4874	0.6291	0.5899	0.6220	**0.7175**
KC3	0.5366	0.5091	0.5541	0.5666	0.4738	0.4931	0.5128	0.6078	**0.6213**	0.5516
MC2	0.5087	0.6445	0.6638	0.6445	0.6676	0.6171	**0.6689**	0.6551	0.6676	0.6175
MW1	0.6617	0.6185	0.7120	0.7289	0.4650	0.4645	0.6693	0.6816	**0.8002**	0.7001
PC1	0.5861	0.5052	0.4950	**0.7452**	0.4892	0.4757	0.5313	0.4861	0.4892	0.6342
PC2	0.5613	0.4823	**0.7701**	0.7660	0.5000	0.5000	0.5000	0.7031	0.7607	0.5090
abalone19	0.4945	0.5756	0.6851	**0.7709**	0.4998	0.4998	0.4950	0.4996	0.7320	0.4975
abalone9v18	**0.7454**	0.5731	0.7367	0.7339	0.7200	0.7200	0.7443	0.7247	0.7225	0.7368
ecoli4	0.7352	0.7459	0.8534	0.8552	0.7373	0.7373	0.7344	0.8575	**0.9128**	0.7334
glass2	0.6833	0.6080	0.6763	**0.7050**	0.6743	0.6570	0.7015	0.6728	0.6433	0.6985
yeast-2_vs_8	0.9947	0.7421	0.9860	**1.0000**	**1.0000**	0.9999	0.9979	**1.0000**	**1.0000**	0.9946
mean	0.6557	0.5854	0.7045	**0.7424**	0.6100	0.6047	0.6531	0.6798	0.7247	0.6719

Table B.2 AUC Performance using SVM

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.6600**	0.4738	0.5886	0.6367	0.5872	0.5875	0.6343	0.5747	0.5929	0.6254
KC3	0.5916	0.5431	0.5469	**0.5959**	0.5081	0.5166	0.5656	0.5491	0.5509	0.5894
MC2	0.5121	0.5833	0.5461	**0.6316**	0.5069	0.5105	0.5150	0.5274	0.5238	0.5189
MW1	0.7811	0.5998	0.7588	**0.8141**	0.7480	0.7364	0.7657	0.7457	0.7577	0.7774
PC1	0.6866	0.5124	0.6278	**0.7133**	0.5593	0.5592	0.7000	0.6087	0.6261	0.6864
PC2	0.5982	0.4256	0.6671	**0.6983**	0.5903	0.5710	0.5510	0.6393	0.6930	0.5845
abalone19	0.5000	0.5813	0.6673	**0.9181**	0.5000	0.5000	0.5000	0.5055	0.7360	0.5000
abalone9v18	0.8557	0.5914	0.8371	0.8375	0.5000	0.5000	0.8548	0.8146	0.8298	**0.8594**
ecoli4	0.7500	0.7552	0.9517	**0.9523**	0.7500	0.7500	0.7500	0.9220	0.9517	0.7500
glass2	0.5680	**0.5935**	0.5538	0.5538	0.5000	0.5008	0.5600	0.5548	0.5785	0.5573
Yeast-2_vs_8	0.9775	0.8001	**1.0000**	**1.0000**	**1.0000**	0.9975	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.6801	0.5872	0.7041	**0.7592**	0.6136	0.6118	0.6724	0.6765	0.7128	0.6771

Table B.3 AUC Performance using Decision Tree

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.7449**	0.4378	0.6611	0.6703	0.6078	0.6056	0.6765	0.6628	0.6534	0.7389
KC3	0.5988	0.5103	0.5759	0.6006	0.5663	0.5588	0.5425	0.5813	0.5878	**0.6166**
MC2	0.6870	0.6080	**0.7494**	0.7265	0.7313	0.6766	0.7079	0.7273	0.7339	0.7260
MW1	0.6989	0.6492	0.7475	0.7252	0.6472	0.6308	0.7276	**0.7824**	0.7380	0.6879
PC1	**0.7361**	0.5197	0.7103	0.7255	0.6174	0.6335	0.7233	0.7042	0.7149	0.7315
PC2	0.5881	0.3605	0.6458	**0.6742**	0.5515	0.5725	0.5579	0.6389	0.6615	0.5857
abalone19	0.5082	0.5833	0.7087	**0.7812**	0.4979	0.4994	0.5569	0.6236	0.7352	0.5048
abalone9v18	0.6912	0.6066	0.7071	0.7005	0.5710	0.5725	0.6763	0.6798	**0.7215**	0.6998
ecoli4	0.7481	0.7359	0.7458	0.7467	0.7486	0.7391	0.7475	0.7488	**0.7677**	0.7484
glass2	0.7070	0.6008	0.7293	0.7250	0.7035	0.6670	0.7260	0.7620	**0.7790**	0.7295
yeast-2_vs_8	0.9375	0.5763	0.9384	0.9543	0.8722	**0.9766**	0.8597	0.8968	0.8865	0.8799
mean	0.6951	0.5626	0.7199	**0.7300**	0.6468	0.6484	0.6820	0.7098	0.7254	0.6954

Table B.4 AUC Performance using Random Forest

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.7086**	0.4457	0.6580	0.6701	0.5193	0.5297	0.6502	0.6525	0.6453	0.6975
KC3	0.5713	0.5006	0.6197	0.6569	0.5025	0.5063	0.5175	**0.6716**	0.6138	0.5522
MC2	**0.7581**	0.6135	0.7181	0.7393	0.7078	0.7033	0.7451	0.7214	0.7161	0.7568
MW1	0.7776	0.6395	0.7666	0.7703	0.6291	0.6380	0.7502	0.7682	**0.8109**	0.7635
PC1	0.6536	0.5109	0.6879	**0.7493**	0.5255	0.5194	0.6341	0.6781	0.7144	0.6621
PC2	0.5091	0.3314	**0.6962**	0.6733	0.5000	0.4998	0.4999	0.6451	0.6471	0.5062
abalone19	0.5000	0.5831	0.6836	**0.8072**	0.5000	0.5000	0.5000	0.5200	0.7065	0.5000
abalone9v18	0.6075	0.5897	0.7072	0.7046	0.5071	0.5057	0.6130	0.6816	**0.7176**	0.6076
ecoli4	0.7466	0.7222	0.7841	0.7956	0.7275	0.7192	0.7517	0.7688	**0.7986**	0.7481
glass2	0.6470	0.6290	0.6825	0.6808	0.5025	0.5050	0.6035	0.6490	**0.6905**	0.6260
yeast-2_vs_8	0.7134	0.6738	0.8119	**0.8288**	0.6750	0.6975	0.7534	0.7475	0.7853	0.7851
mean	0.6539	0.5672	0.7105	**0.7342**	0.5724	0.5749	0.6381	0.6822	0.7133	0.6550

Table B.5 AUC Performance using Neural Networks

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.5588	0.4839	0.5095	0.5562	0.5113	0.5116	**0.5753**	0.5273	0.5062	0.5583
KC3	0.5506	0.5103	0.5372	0.5516	0.5181	0.5206	**0.5647**	0.5475	0.5328	0.5447
MC2	0.6005	0.5358	0.5988	**0.6453**	0.5825	0.5979	0.6076	0.5931	0.6086	0.6033
MW1	**0.6893**	0.5789	0.5684	0.6104	0.5133	0.4687	0.6426	0.5999	0.5743	0.6573
PC1	**0.6170**	0.5097	0.5265	0.6167	0.5292	0.5085	0.5998	0.5239	0.5417	0.6161
PC2	0.5333	0.4348	0.5439	0.5338	0.5139	0.5054	**0.5657**	0.5539	0.5177	0.5401
abalone19	0.5000	0.5807	0.6241	**0.8774**	0.5000	0.5000	0.5000	0.5000	0.6912	0.5000
abalone9v18	0.8450	0.5893	0.8256	0.8285	0.5000	0.5000	0.8385	0.7968	0.8183	**0.8464**
ecoli4	0.9248	0.7436	0.9464	0.9492	0.7350	0.7450	0.8063	0.9302	**0.9530**	0.7909
glass2	0.5578	0.5605	0.5333	0.5543	0.5000	0.5000	0.5028	0.5780	**0.5823**	0.5193
yeast-2_vs_8	0.9624	0.8506	**1.0000**	**1.0000**	0.7500	0.7500	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.6672	0.5798	0.6558	**0.7021**	0.5594	0.5553	0.6548	0.6501	0.6660	0.6524

Table B.6 AUC Performance using AdaBoost

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.7302	0.4367	0.6726	0.7033	0.5643	0.5765	0.6080	0.6866	0.6797	**0.7357**
KC3	0.5994	0.5084	0.5934	**0.6306**	0.5875	0.5494	0.6069	0.6063	0.5997	0.6003
MC2	0.6248	0.5908	0.6108	0.6339	0.5993	0.6015	0.5963	0.6329	0.6189	**0.6366**
MW1	0.6025	0.6484	0.7307	0.7240	0.5802	0.5880	0.6009	**0.7715**	0.7492	0.6460
PC1	0.6747	0.5200	0.7224	**0.7594**	0.6420	0.6438	0.6910	0.7088	0.7377	0.6794
PC2	0.5230	0.3639	0.7404	**0.7498**	0.4986	0.4962	0.5114	0.7001	0.7200	0.5314
abalone19	0.5653	0.5830	0.7423	**0.7950**	0.4995	0.4993	0.5125	0.6397	0.7347	0.5756
abalone9v18	0.6974	0.6128	0.7148	0.7171	0.6522	0.6494	**0.7205**	0.6798	0.7079	0.7096
ecoli4	0.7492	0.7341	0.7558	0.7577	0.7489	0.7438	0.7492	0.7488	**0.8039**	0.7483
glass2	0.8478	0.5985	0.7308	0.7625	0.7315	0.7183	0.7458	0.7398	0.7168	**0.8558**
yeast-2_vs_8	0.7486	0.5613	**0.8837**	0.8646	0.7596	0.8100	0.7604	0.8178	0.8229	0.7481
mean	0.6694	0.5598	0.7180	**0.7362**	0.6240	0.6251	0.6457	0.7029	0.7174	0.6788

Table B.7 AUC Performance using Naïve Bayes

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.6830**	0.4498	0.6468	0.6532	0.6003	0.6013	0.6021	0.6088	0.6515	0.6721
KC3	0.7381	0.5194	0.6894	0.6969	0.5631	0.6741	0.6456	0.6975	**0.7459**	0.7347
MC2	0.6975	0.6494	**0.7000**	0.6994	0.6988	0.6531	0.6988	0.6875	**0.7000**	0.6981
MW1	0.6872	0.6228	0.7015	**0.8404**	0.6963	0.6857	0.6874	0.6922	0.7239	0.6900
PC1	0.6366	0.5221	**0.6431**	0.6409	0.6416	0.6423	0.6426	0.6419	0.6416	0.6378
PC2	0.7083	0.3446	0.6777	0.6349	0.6369	0.5685	**0.7269**	0.6927	0.6051	0.7130
abalone19	0.5482	0.5750	0.5794	**0.7345**	0.5255	0.5325	0.5471	0.5536	0.7121	0.5413
abalone9v18	0.7661	0.5748	0.7674	0.7409	0.7692	0.7652	0.7680	**0.7699**	0.7524	0.7663
ecoli4	0.9088	0.8664	0.8909	0.9192	0.9131	0.8913	0.9127	0.9034	0.7892	**0.9345**
glass2	0.6382	0.6225	0.6507	**0.7163**	0.6395	0.6410	0.6360	0.6385	0.6288	0.6507
yeast-2_vs_8	0.6923	0.8577	**0.9941**	0.9846	0.6790	0.9659	0.6900	0.9746	0.9727	0.6986
mean	0.7004	0.6004	0.7219	**0.7510**	0.6694	0.6928	0.6870	0.7146	0.7203	0.7034

Table B.8 AUC Performance using QDA

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.5350	0.4675	0.5747	0.5795	0.5000	0.5000	0.5319	**0.6115**	0.5638	0.5322
KC3	0.5000	0.4844	0.5063	0.5150	0.5000	0.5000	0.5000	**0.5241**	0.5213	0.5000
MC2	0.5000	0.5029	0.5994	0.6139	0.5000	0.5290	0.5000	**0.6220**	0.6085	0.5000
MW1	0.5000	0.5000	0.5564	0.5518	0.5000	0.5000	0.5000	**0.5985**	0.5668	0.5000
PC1	0.6876	0.5206	0.6847	**0.6922**	0.6846	0.6835	0.6861	0.6839	0.6846	0.6802
PC2	0.5000	0.5100	0.5886	0.5960	0.5000	0.5000	0.5000	0.5165	**0.5977**	0.5000
abalone19	0.6171	0.4852	0.7337	0.7310	0.4929	0.4931	0.5961	0.7357	**0.7766**	0.6361
abalone9v18	**0.8625**	0.6214	0.8277	0.8379	0.7425	0.7421	0.8611	0.8269	0.8118	0.8561
ecoli4	0.6097	0.6002	0.6911	0.6955	0.6138	0.6161	0.6198	0.6650	**0.7061**	0.6411
glass2	0.8348	0.8298	0.8828	0.8923	0.8420	0.8270	0.8700	0.8565	**0.8973**	0.8285
yeast-2_vs_8	0.6564	0.6676	0.7995	**0.8106**	0.6952	0.7868	0.6948	0.7197	0.7517	0.7020
mean	0.6184	0.5627	0.6768	**0.6832**	0.5974	0.6071	0.6236	0.6691	0.6806	0.6251

Table B.9 G-Mean Performance using KNN

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.6964	0.1835	0.5889	0.6363	0.0097	0.0729	0.5993	0.5743	0.6019	**0.7100**
KC3	0.4757	0.0579	0.4852	0.5494	0.0000	0.1103	0.4305	0.5941	**0.5947**	0.4731
MC2	0.4921	0.5714	0.6246	0.6281	0.6552	0.6122	**0.6586**	0.6440	0.6552	0.6126
MW1	0.6133	0.5871	0.6709	0.6810	0.0000	0.0071	0.6001	0.6564	**0.7886**	0.6629
PC1	0.5435	0.1065	0.3752	**0.7251**	0.0000	0.0000	0.3793	0.3880	0.4131	0.6043
PC2	0.3284	0.1148	**0.7394**	0.7337	0.0000	0.0000	0.0000	0.6550	0.7326	0.1750
abalone19	0.0171	0.5232	0.6544	**0.7555**	0.0000	0.0000	0.0000	0.0631	0.7115	0.0057
abalone9v18	0.7015	0.4695	0.7167	**0.7230**	0.6545	0.6545	0.7019	0.6854	0.7138	0.6969
ecoli4	0.6965	0.7067	0.8382	0.8394	0.6981	0.6981	0.6960	0.8432	**0.9082**	0.6953
glass2	0.6530	0.4648	0.5993	0.6685	0.5506	0.5379	**0.6719**	0.5607	0.5861	0.6595
yeast-2_vs_8	0.9946	0.6855	0.9855	**1.0000**	**1.0000**	0.9999	0.9979	**1.0000**	**1.0000**	0.9945
mean	0.5647	0.4064	0.6617	**0.7218**	0.3244	0.3357	0.5214	0.6058	0.7005	0.5718

Table B.10 G-Mean Performance using SVM

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.5285**	0.0977	0.4040	0.5182	0.3996	0.4049	0.4467	0.3678	0.4276	0.4359
KC3	0.3810	0.2077	0.2177	**0.4050**	0.1404	0.1682	0.2656	0.2604	0.2465	0.3518
MC2	0.2726	0.4548	0.3070	**0.4921**	0.2019	0.2112	0.2542	0.2820	0.2812	0.2284
MW1	0.7318	0.5672	0.6916	**0.7894**	0.6767	0.6469	0.7147	0.6642	0.6937	0.7278
PC1	0.6021	0.1343	0.5232	**0.6431**	0.3371	0.3489	0.6220	0.4661	0.5245	0.6003
PC2	0.3773	0.1986	0.5493	**0.6263**	0.3458	0.2836	0.2239	0.4777	0.6048	0.3504
abalone19	0.0000	0.5274	0.6483	**0.9148**	0.0000	0.0000	0.0000	0.0400	0.7070	0.0000
abalone9v18	0.8491	0.4933	0.8326	0.8329	0.0000	0.0000	0.8459	0.8058	0.8259	**0.8524**
ecoli4	0.7075	0.7155	0.9494	**0.9500**	0.7071	0.7071	0.7071	0.9173	0.9493	0.7071
glass2	0.3032	**0.4449**	0.2640	0.2956	0.0000	0.0054	0.2560	0.2654	0.3844	0.2553
yeast-2_vs_8	0.9736	0.7722	**1.0000**	**1.0000**	**1.0000**	0.9971	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.5206	0.4194	0.5806	**0.6788**	0.3462	0.3430	0.4851	0.5043	0.6041	0.5009

Decision Tree G-Mean

Table B.11 G-Mean Performance using Decision Tree

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.7386**	0.1578	0.6454	0.6596	0.4548	0.4803	0.6629	0.6484	0.6356	0.7311
KC3	0.4129	0.0697	0.4994	**0.5044**	0.2650	0.2464	0.2252	0.4984	0.4833	0.4860
MC2	0.6775	0.4992	0.7132	**0.7144**	0.6968	0.6492	0.6670	0.6936	0.7014	0.7026
MW1	0.6280	0.6144	0.7178	0.7009	0.4695	0.4052	0.6469	**0.7624**	0.7185	0.5766
PC1	**0.7286**	0.1621	0.7020	0.7169	0.4821	0.5222	0.7183	0.6958	0.7061	0.7204
PC2	0.3137	0.1539	0.5733	**0.5933**	0.1480	0.2117	0.1694	0.5489	0.5925	0.3031
abalone19	0.0452	0.5282	0.6885	**0.7667**	0.0000	0.0057	0.2126	0.4695	0.7234	0.0341
abalone9v18	0.6391	0.5016	0.6860	0.6844	0.3119	0.3212	0.6101	0.6607	**0.7112**	0.6320
ecoli4	0.7058	0.6906	0.7076	0.7105	0.7061	0.6940	0.7048	0.7062	**0.7387**	0.7060
glass2	0.6656	0.4492	0.7097	0.6810	0.6133	0.5292	0.6639	0.6747	**0.7144**	0.6280
yeast-2_vs_8	0.9346	0.4214	0.9345	0.9482	0.8504	**0.9728**	0.8374	0.8847	0.8674	0.8686
mean	0.5899	0.3862	0.6888	**0.6982**	0.4544	0.4580	0.5562	0.6585	0.6902	0.5808

Table B.12 G-Mean Performance using Random Forest

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.3288	0.1268	0.1977	0.3587	0.1550	0.1909	**0.3749**	0.2123	0.2580	0.3392
KC3	0.2902	0.1693	0.2309	0.2918	0.1856	0.1889	**0.3319**	0.2448	0.2029	0.2515
MC2	0.3831	0.2178	0.3601	**0.5065**	0.3359	0.3424	0.3379	0.3215	0.3971	0.3592
MW1	**0.5534**	0.4253	0.3145	0.3816	0.1223	0.0866	0.4861	0.3361	0.3240	0.5009
PC1	0.4924	0.1417	0.2128	**0.4997**	0.2049	0.1241	0.4675	0.2040	0.2719	0.4698
PC2	0.2053	0.1953	0.2500	0.2765	0.0597	0.0851	0.2961	**0.3213**	0.2284	0.2914
abalone19	0.0000	0.5269	0.5448	**0.8689**	0.0000	0.0000	0.0000	0.0000	0.6448	0.0000
abalone9v18	0.8360	0.4850	0.8214	0.8228	0.0000	0.0000	0.8287	0.7893	0.8154	**0.8409**
ecoli4	0.9173	0.6812	0.9429	0.9461	0.6647	0.6930	0.7800	0.9261	**0.9504**	0.7618
glass2	**0.4721**	0.2966	0.3089	0.3258	0.0000	0.0000	0.3812	0.3685	0.3464	0.4420
yeast-2_vs_8	0.9584	0.8345	**1.0000**	**1.0000**	0.7071	0.7071	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.4943	0.3728	0.4713	**0.5708**	0.2214	0.2198	0.4804	0.4294	0.4945	0.4779

Table B.13 G-Mean Performance using Neural Networks

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.3288	0.1268	0.1977	0.3587	0.1550	0.1909	**0.3749**	0.2123	0.2580	0.3392
KC3	0.2902	0.1693	0.2309	0.2918	0.1856	0.1889	**0.3319**	0.2448	0.2029	0.2515
MC2	0.3831	0.2178	0.3601	**0.5065**	0.3359	0.3424	0.3379	0.3215	0.3971	0.3592
MW1	**0.5534**	0.4253	0.3145	0.3816	0.1223	0.0866	0.4861	0.3361	0.3240	0.5009
PC1	0.4924	0.1417	0.2128	**0.4997**	0.2049	0.1241	0.4675	0.2040	0.2719	0.4698
PC2	0.2053	0.1953	0.2500	0.2765	0.0597	0.0851	0.2961	**0.3213**	0.2284	0.2914
abalone19	0.0000	0.5269	0.5448	**0.8689**	0.0000	0.0000	0.0000	0.0000	0.6448	0.0000
abalone9v18	0.8360	0.4850	0.8214	0.8228	0.0000	0.0000	0.8287	0.7893	0.8154	**0.8409**
ecoli4	0.9173	0.6812	0.9429	0.9461	0.6647	0.6930	0.7800	0.9261	**0.9504**	0.7618
glass2	**0.4721**	0.2966	0.3089	0.3258	0.0000	0.0000	0.3812	0.3685	0.3464	0.4420
yeast-2_vs_8	0.9584	0.8345	**1.0000**	**1.0000**	0.7071	0.7071	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.4943	0.3728	0.4713	**0.5708**	0.2214	0.2198	0.4804	0.4294	0.4945	0.4779

Table B.14 G-Mean Performance using AdaBoost

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.7151	0.1578	0.6622	0.6881	0.3779	0.4536	0.5458	0.6783	0.6721	**0.7223**
KC3	0.4497	0.0773	0.5280	0.5316	0.3954	0.3283	0.4789	**0.5751**	0.5134	0.4775
MC2	0.5990	0.4711	0.5911	**0.6257**	0.5396	0.5875	0.5460	0.6081	0.5822	0.6038
MW1	0.3857	0.6135	0.7132	0.7023	0.2772	0.3002	0.3448	0.7337	**0.7363**	0.4459
PC1	0.6166	0.1616	0.7150	**0.7526**	0.5271	0.5363	0.6412	0.7000	0.7307	0.6201
PC2	0.0974	0.1237	0.7015	**0.7108**	0.0070	0.0000	0.0491	0.6290	0.6761	0.1249
abalone19	0.2413	0.5280	0.7238	**0.7830**	0.0000	0.0000	0.0631	0.4950	0.7225	0.2811
abalone9v18	0.6366	0.5080	0.6900	**0.7081**	0.5117	0.4869	0.6739	0.6414	0.6972	0.6525
ecoli4	0.7065	0.6887	0.7283	0.7353	0.7063	0.7026	0.7065	0.7295	**0.7862**	0.7059
glass2	0.8264	0.4400	0.7071	0.7106	0.6614	0.6461	0.6766	0.6785	0.6762	**0.8300**
yeast-2_vs_8	0.7089	0.4138	**0.8708**	0.8571	0.7196	0.7793	0.7214	0.8070	0.8092	0.7080
mean	0.5439	0.3803	0.6937	**0.7096**	0.4294	0.4383	0.4952	0.6614	0.6911	0.5611

Table B.15 G-Mean Performance using AdaBoost

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.6410**	0.2990	0.5798	0.5912	0.4983	0.4999	0.5013	0.5214	0.5815	0.6210
KC3	0.6938	0.1016	0.5702	0.5909	0.2253	0.5763	0.4539	0.6009	**0.7044**	0.6903
MC2	0.6308	0.5902	**0.6325**	0.6320	0.6316	0.6011	0.6316	0.6243	**0.6325**	0.6312
MW1	0.6611	0.5952	0.6719	**0.8293**	0.6680	0.6599	0.6613	0.6649	0.7007	0.6633
PC1	0.5597	0.2396	0.5612	**0.5635**	0.5627	0.5631	0.5633	0.5629	0.5627	0.5604
PC2	0.6767	0.2279	0.5717	0.5093	0.4408	0.2787	**0.6931**	0.6430	0.4268	0.6797
abalone19	0.4771	0.5239	0.5580	**0.7058**	0.2924	0.3210	0.4916	0.4737	0.6687	0.4551
abalone9v18	0.7660	0.4735	0.7672	0.7359	0.7687	0.7637	0.7678	**0.7696**	0.7523	0.7661
ecoli4	0.9020	0.8494	0.8784	0.9126	0.9058	0.8818	0.9057	0.8948	0.7568	**0.9295**
glass2	0.5251	0.4885	0.5483	**0.6651**	0.5276	0.5291	0.5210	0.5257	0.5072	0.5483
yeast-2_vs_8	0.6067	0.8404	**0.9940**	0.9838	0.5803	0.9540	0.6000	0.9645	0.9705	0.6155
mean	0.6491	0.4754	0.6666	**0.7018**	0.5547	0.6026	0.6173	0.6587	0.6604	0.6510

Table B.16 G-Mean Performance using QDA

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.1539	0.0716	0.3874	0.3907	0.0000	0.0000	0.1437	**0.5039**	0.4080	0.1391
KC3	0.0000	0.0495	0.2510	0.2499	0.0000	0.0000	0.0000	**0.3628**	0.3380	0.0000
MC2	0.0000	0.1901	0.3872	0.4020	0.0000	0.1297	0.0000	**0.5666**	0.3730	0.0000
MW1	0.0000	0.0000	0.2041	0.1813	0.0000	0.0000	0.0000	**0.3444**	0.3393	0.0000
PC1	0.6281	0.0786	0.6689	**0.6791**	0.6299	0.6278	0.6312	0.6407	0.6658	0.6144
PC2	0.0000	0.1024	0.3575	0.3730	0.0000	0.0000	0.0000	0.1111	**0.4295**	0.0000
abalone19	0.4376	0.0303	0.7284	0.7239	0.0000	0.0000	0.3891	0.7086	**0.7638**	0.4899
abalone9v18	**0.8547**	0.4878	0.8199	0.8307	0.7018	0.7015	0.8539	0.8146	0.8089	0.8495
ecoli4	0.3250	0.3151	0.5218	0.5268	0.3222	0.3416	0.3295	0.4828	**0.5621**	0.3973
glass2	0.7456	0.7259	0.8284	0.8431	0.7603	0.7437	0.8180	0.7958	**0.8496**	0.7397
yeast-2_vs_8	0.4810	0.4550	0.7103	**0.7263**	0.5552	0.6706	0.5735	0.5717	0.6242	0.5989
mean	0.3296	0.2278	0.5332	0.5388	0.2699	0.2923	0.3399	0.5366	**0.5602**	0.3481

Table B.17 MCC Performance using KNN

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.2780	-0.1762	0.1862	0.2241	-0.0618	-0.0403	0.2006	0.1434	0.1836	**0.3051**
KC3	0.0662	0.0287	0.0966	0.1103	-0.0938	-0.0318	0.0260	0.1871	**0.2154**	0.0886
MC2	0.0136	0.3953	**0.4522**	0.3045	0.3476	0.2334	0.3475	0.3200	0.3476	0.2349
MW1	0.2016	0.1756	0.2604	0.2887	-0.0702	-0.0673	0.2666	0.2103	**0.3637**	0.2392
PC1	0.1086	0.0248	-0.0198	**0.2803**	-0.0429	-0.0664	0.0545	-0.0372	-0.0251	0.1738
PC2	0.0548	-0.0169	**0.2106**	0.2028	0.0000	0.0000	0.0000	0.1501	0.1911	0.0074
abalone19	-0.0088	0.0356	0.0720	**0.1440**	-0.0006	-0.0006	-0.0083	-0.0013	0.0795	-0.0058
abalone9v18	**0.6529**	0.0673	0.5874	0.3631	0.6446	0.6446	0.6178	0.6466	0.6269	0.6285
ecoli4	0.4862	0.5285	**0.5939**	0.5419	0.5120	0.5120	0.4724	0.5236	0.5750	0.4673
glass2	0.3241	0.1658	0.3719	**0.3949**	0.3589	0.2654	0.3555	0.3512	0.2450	0.3565
yeast-2_vs_8	0.9062	0.2192	0.9061	**1.0000**	**1.0000**	0.9981	0.9623	**1.0000**	**1.0000**	0.9065
mean	0.2803	0.1316	0.3380	**0.3504**	0.2358	0.2225	0.2995	0.3176	0.3457	0.3093

Table B.18 MCC Performance using SVM

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.2111**	-0.0630	0.1421	0.1948	0.1300	0.1386	0.1759	0.1094	0.1322	0.1655
KC3	0.1744	0.1173	0.0849	**0.1933**	0.0020	0.0161	0.1121	0.0983	0.0994	0.1750
MC2	0.0236	**0.2910**	0.0968	0.2853	-0.0019	0.0124	0.0235	0.0561	0.0553	0.0333
MW1	0.5891	0.1350	**0.6149**	0.5078	0.5833	0.5286	0.5667	0.5561	0.5900	0.5785
PC1	0.2850	0.0383	0.1821	0.2866	0.1107	0.1162	**0.2949**	0.1568	0.1828	0.2718
PC2	0.0953	-0.0841	0.1546	0.1513	0.0893	0.0696	0.0428	0.1127	**0.1595**	0.0766
abalone19	0.0000	0.0376	0.1405	**0.2122**	0.0000	0.0000	0.0000	0.0042	0.0941	0.0000
abalone9v18	0.6951	0.0845	**0.6972**	0.5060	0.0000	0.0000	**0.6972**	**0.6972**	0.4518	**0.6972**
ecoli4	**0.6963**	0.5799	**0.6963**	**0.6963**	**0.6963**	**0.6963**	**0.6963**	**0.6963**	0.6857	**0.6963**
glass2	0.0990	**0.1409**	0.0800	0.0770	0.0000	0.0017	0.0820	0.0799	0.1194	0.0830
yeast-2_vs_8	0.9730	0.2948	**1.0000**	**1.0000**	**1.0000**	0.9970	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.3493	0.1429	0.3536	**0.3737**	0.2372	0.2342	0.3356	0.3243	0.3246	0.3434

Table B.19 MCC Performance using Decision Tree

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.3581**	-0.1850	0.2167	0.2359	0.2144	0.2176	0.2522	0.2230	0.2106	0.3562
KC3	0.3503	0.0398	0.2211	0.2940	0.2432	0.2110	0.1555	0.2223	0.2386	**0.4045**
MC2	0.3821	0.3234	**0.5751**	0.5033	0.5400	0.4049	0.4816	0.5283	0.5460	0.4886
MW1	0.3317	0.3256	0.3869	0.3093	0.2835	0.2543	0.4202	**0.4343**	0.3675	0.3346
PC1	0.3324	0.0712	0.3435	0.3414	0.3561	0.3513	0.3671	**0.3821**	0.3499	0.3646
PC2	0.1460	-0.1763	0.1437	0.1385	0.1269	0.1826	0.1495	**0.2122**	0.1367	0.1366
abalone19	0.0199	0.0386	0.0781	**0.1196**	-0.0051	-0.0026	0.0977	0.0721	0.0851	0.0025
abalone9v18	0.2992	0.0977	0.2391	0.2181	0.1935	0.1898	0.2849	0.2164	0.2252	**0.3220**
ecoli4	0.6702	0.5399	0.6385	0.6522	0.6758	0.5872	**0.6893**	0.6781	0.6234	0.6736
glass2	0.3361	0.1506	0.3180	0.3821	0.3827	0.2988	0.4031	0.4444	**0.4748**	0.4118
yeast-2_vs_8	0.7935	0.0796	0.8409	0.8738	0.8405	**0.9578**	0.7853	0.8329	0.8473	0.8360
mean	0.3654	0.1186	0.3638	0.3698	0.3501	0.3321	0.3715	0.3860	0.3732	**0.3937**

Table B.20 MCC Performance using Random Forest

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.3054	-0.1202	0.2542	0.2495	0.0724	0.1119	0.2782	0.2135	0.2170	**0.3309**
KC3	0.1636	0.0052	0.2299	0.3047	0.0092	0.0220	0.0337	**0.3135**	0.2186	0.1163
MC2	0.5647	0.3387	0.5022	0.5260	0.5338	0.4922	0.5626	0.5448	0.5375	**0.5852**
MW1	0.5569	0.2668	0.5200	0.4540	0.3505	0.3717	0.5463	0.5435	0.4936	**0.5579**
PC1	0.2250	0.0455	0.2718	**0.2904**	0.1124	0.0861	0.2828	0.2654	0.2595	0.2164
PC2	0.0193	-0.1945	**0.1477**	0.1190	0.0000	-0.0006	-0.0002	0.1223	0.1091	0.0129
abalone19	0.0000	0.0411	0.0655	**0.1760**	0.0000	0.0000	0.0000	0.0209	0.0701	0.0000
abalone9v18	0.1997	0.0812	0.2171	0.2081	0.0264	0.0211	0.1811	0.1777	0.2125	**0.2225**
ecoli4	0.6559	0.4948	0.6197	**0.6872**	0.6336	0.6032	**0.6872**	0.6453	0.5596	0.6714
glass2	**0.2536**	0.1730	0.2283	0.2220	0.0069	0.0138	0.1874	0.1897	0.2258	0.1982
yeast-2_vs_8	0.4926	0.1685	0.5729	0.6201	0.4538	0.4529	0.6242	0.5108	0.4777	**0.6751**
mean	0.3124	0.1182	0.3299	**0.3506**	0.1999	0.1977	0.3076	0.3225	0.3074	0.3261

Table B.21 MCC Performance using Neural Network

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.0910	-0.0254	0.0172	0.0856	0.0172	0.0184	**0.1163**	0.0453	0.0104	0.0923
KC3	0.0960	0.0198	0.0723	0.0989	0.0339	0.0468	**0.1336**	0.0891	0.0703	0.0893
MC2	0.2211	0.0856	0.2223	**0.3180**	0.1848	0.2176	0.2261	0.2078	0.2490	0.2292
MW1	**0.2532**	0.1094	0.0885	0.1321	0.0238	-0.0342	0.1788	0.1319	0.1042	0.1880
PC1	0.1578	0.0288	0.0373	**0.1775**	0.0480	0.0166	0.1284	0.0422	0.0608	0.1572
PC2	0.0235	-0.0564	0.0455	0.0318	0.0169	0.0042	**0.0627**	0.0487	0.0272	0.0294
abalone19	0.0000	0.0368	0.0849	**0.1621**	0.0000	0.0000	0.0000	0.0000	0.0804	0.0000
abalone9v18	0.7492	0.0831	0.6703	0.4959	0.0000	0.0000	0.7279	0.6263	0.4090	**0.7613**
ecoli4	**0.6963**	0.5614	**0.6963**	**0.6963**	0.6545	0.6824	**0.6963**	**0.6963**	0.6794	**0.6963**
glass2	0.0749	0.0914	0.0425	0.0683	0.0000	0.0000	0.0076	0.1014	**0.1167**	0.0327
yeast-2_vs_8	0.8310	0.3883	**1.0000**	**1.0000**	0.6997	0.6997	**1.0000**	**1.0000**	**1.0000**	**1.0000**
mean	0.2903	0.1203	0.2707	0.2970	0.1526	0.1501	**0.2980**	0.2717	0.2552	0.2978

Table B.22 MCC Performance using AdaBoost

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.3665	-0.1880	0.2464	0.3143	0.1474	0.1663	0.1821	0.2689	0.2450	**0.3841**
KC3	0.3017	0.0230	0.2373	**0.3124**	0.2769	0.1369	0.2907	0.2637	0.2602	0.2841
MC2	0.2644	**0.2911**	0.2286	0.2698	0.2135	0.2129	0.2074	0.2825	0.2540	0.2907
MW1	0.1927	0.3209	0.3352	0.2887	0.1816	0.2001	0.2120	**0.3883**	0.3257	0.2846
PC1	0.4047	0.0674	0.3842	0.3262	0.4130	0.3836	0.3990	0.3753	0.4130	**0.4347**
PC2	0.0338	-0.1828	0.1827	**0.1838**	-0.0059	-0.0096	0.0237	0.1584	0.1629	0.0475
abalone19	0.1032	0.0385	0.1015	**0.1642**	-0.0018	-0.0022	0.0202	0.0860	0.0837	0.1120
abalone9v18	**0.4690**	0.1035	0.3821	0.2872	0.4364	0.4068	0.4181	0.4361	0.3788	0.4584
ecoli4	**0.6849**	0.5464	0.6702	0.6781	0.6804	0.6274	**0.6849**	0.6736	0.4717	0.6713
glass2	0.5806	0.1491	0.5596	0.5598	0.5627	0.4823	0.5377	0.5681	0.4589	**0.5938**
yeast-2_vs_8	0.5806	0.0666	0.6086	0.6354	0.6568	**0.6895**	0.6474	0.6404	0.6501	0.5738
mean	0.3620	0.1123	0.3578	0.3654	0.3237	0.2995	0.3294	**0.3765**	0.3367	0.3759

Table B.23 MCC Performance using Naïve Bayes

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	**0.3521**	-0.1074	0.2710	0.2961	0.2229	0.2248	0.2487	0.2237	0.3075	0.3374
KC3	0.6233	0.0455	0.5177	0.5389	0.2054	0.5316	0.4154	0.5183	**0.6442**	0.6145
MC2	0.5305	0.3994	**0.5394**	0.5371	0.5349	0.3880	0.5349	0.4950	**0.5394**	0.5327
MW1	0.2832	0.1862	0.3284	**0.4665**	0.3119	0.2813	0.2851	0.2993	0.3259	0.2923
PC1	0.2800	0.0549	0.3235	0.3183	0.3060	0.3123	0.3135	0.3092	**0.3280**	0.2860
PC2	0.2620	-0.1658	0.2347	0.1210	0.1999	0.1014	**0.3177**	0.2397	0.1120	0.2961
abalone19	0.0199	0.0332	0.0268	**0.0820**	0.0105	0.0132	0.0190	0.0231	0.0718	0.0169
abalone9v18	0.2794	0.0687	0.2816	0.2502	0.2859	0.2838	0.2826	**0.2861**	0.2571	0.2797
ecoli4	0.5234	0.4762	0.5845	0.5855	0.5443	0.4786	0.5400	0.5269	0.4748	**0.6228**
glass2	0.1832	0.1692	0.1944	**0.2542**	0.1843	0.1860	0.1812	0.1835	0.1748	0.1944
yeast-2_vs_8	0.2211	0.3868	**0.9340**	0.8913	0.2145	0.9183	0.2214	0.9030	0.8377	0.2250
mean	0.3235	0.1406	0.3851	**0.3947**	0.2746	0.3381	0.3054	0.3643	0.3703	0.3361

Table B.24 MCC Performance using QDA

Dataset	ADASYN	CLUS	CUST	HCBST	NONE	OSS	ROS	RUS	SBC	SMOTE
CM1	0.1242	-0.0690	0.1156	0.1503	0.0000	0.0000	0.1133	**0.1575**	0.0938	0.1157
KC3	0.0000	-0.0285	0.0209	0.0354	0.0000	0.0000	0.0000	0.0454	**0.0457**	0.0000
MC2	0.0000	0.0057	0.2625	0.2807	0.0000	0.1059	0.0000	0.2680	**0.2990**	0.0000
MW1	0.0000	0.0000	0.1544	0.1401	0.0000	0.0000	0.0000	**0.2686**	0.1851	0.0000
PC1	0.3856	0.0390	0.3702	0.3532	0.3593	0.3633	**0.3861**	0.3844	0.3593	0.3772
PC2	0.0000	0.0093	0.0713	0.0722	0.0000	0.0000	0.0000	0.0091	**0.0822**	0.0000
abalone19	0.1026	-0.0108	0.1250	**0.1594**	-0.0100	-0.0097	0.0799	0.1277	0.0993	0.1091
abalone9v18	**0.6866**	0.1234	0.5560	0.5351	0.5526	0.5483	0.6559	0.5756	0.5117	0.6605
ecoli4	0.1912	0.1397	0.2590	0.2808	0.1769	0.1762	0.1834	0.2326	**0.3009**	0.2109
glass2	0.6064	0.4535	0.6146	**0.6239**	0.5809	0.5627	0.6178	0.5892	0.6127	0.5827
yeast-2_vs_8	0.1873	0.2170	0.4256	**0.4358**	0.2545	0.4347	0.2558	0.2859	0.3370	0.2612
mean	0.2076	0.0799	0.2705	**0.2788**	0.1740	0.1983	0.2084	0.2676	0.2661	0.2107

APPENDIX C

OTHER PERFORMANCE METRICS FOR NONE

I. OTHER PERFORMANCE METRICS FOR NONE USING KNN

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.000	0.000
glass2	0.899	0.400	0.949	0.674	0.895	0.551	0.359	18.970	0.800	1.200	1.030
ecoli4	0.947	0.500	0.975	0.737	0.946	0.698	0.512	31.190	1.000	1.000	0.810
CM1	0.845	0.005	0.961	0.483	0.784	0.010	-0.062	27.880	0.020	3.980	1.120
KC3	0.758	0.000	0.948	0.474	0.654	0.000	-0.094	15.160	0.000	4.000	0.840
MC2	0.692	0.564	0.771	0.668	0.690	0.655	0.348	6.170	2.820	2.180	1.830
MW1	0.856	0.000	0.930	0.465	0.844	0.000	-0.070	21.390	0.000	2.000	1.610
PC1	0.892	0.000	0.978	0.489	0.841	0.000	-0.043	60.660	0.000	6.000	1.340
PC2	0.973	0.000	1.000	0.500	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.970	0.440	1.000	0.720	0.964	0.654	0.645	70.000	1.760	2.240	0.000
abalone19	0.992	0.000	1.000	0.500	0.987	0.000	-0.001	414.860	0.000	3.000	0.140
Mean	0.893	0.264	0.956	0.610	0.869	0.324	0.236	71.298	0.764	2.509	0.793

II. OTHER PERFORMANCE METRICS FOR NONE USING SVM

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.000	0.000
glass2	0.852	0.070	0.930	0.500	0.784	0.000	0.000	18.600	0.140	1.860	1.400
ecoli4	0.971	0.500	1.000	0.750	0.967	0.707	0.696	32.000	1.000	1.000	0.000
CM1	0.776	0.338	0.837	0.587	0.799	0.400	0.130	24.270	1.350	2.650	4.730
KC3	0.721	0.153	0.864	0.508	0.655	0.140	0.002	13.820	0.610	3.390	2.180
MC2	0.564	0.260	0.754	0.507	0.465	0.202	-0.002	6.030	1.300	3.700	1.970
MW1	0.914	0.550	0.946	0.748	0.928	0.677	0.583	21.760	1.100	0.900	1.240
PC1	0.857	0.198	0.920	0.559	0.854	0.337	0.111	57.060	1.190	4.810	4.940
PC2	0.898	0.265	0.916	0.590	0.939	0.346	0.089	65.010	0.530	1.470	5.990
abalone9v18	0.946	0.000	1.000	0.500	0.905	0.000	0.000	70.000	0.000	4.000	0.000
abalone19	0.993	0.000	1.000	0.500	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.863	0.303	0.924	0.614	0.844	0.346	0.237	70.050	0.838	2.435	2.041

III. OTHER PERFORMANCE METRICS FOR NONE USING DECISION TREE

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.989	0.745	0.999	0.872	0.988	0.850	0.841	46.970	1.490	0.580	0.030
glass2	0.895	0.470	0.940	0.704	0.897	0.613	0.383	18.800	0.940	1.110	1.270
ecoli4	0.968	0.500	0.997	0.749	0.965	0.706	0.676	31.910	1.000	1.000	0.150
CM1	0.830	0.315	0.902	0.608	0.830	0.455	0.214	26.160	1.260	2.850	3.040
KC3	0.826	0.133	1.000	0.566	0.748	0.265	0.243	16.000	0.530	3.530	0.000
MC2	0.776	0.540	0.929	0.731	0.777	0.697	0.540	7.430	2.700	2.370	0.710
MW1	0.897	0.350	0.945	0.647	0.895	0.470	0.283	21.730	0.700	1.390	1.300
PC1	0.917	0.253	0.982	0.617	0.899	0.482	0.356	60.860	1.520	4.540	1.210
PC2	0.974	0.105	0.998	0.552	0.957	0.148	0.127	70.860	0.210	1.830	0.180
abalone9v18	0.940	0.158	0.985	0.571	0.921	0.312	0.194	68.930	0.630	3.410	1.190
abalone19	0.989	0.000	0.996	0.498	0.986	0.000	-0.005	413.230	0.000	3.000	1.950

IV. OTHER PERFORMANCE METRICS FOR NONE USING RANDOM FOREST

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.973	0.350	1.000	0.675	0.960	0.458	0.454	47.000	0.700	1.470	0.000
glass2	0.910	0.005	1.000	0.503	0.843	0.007	0.007	20.000	0.010	2.000	0.010
ecoli4	0.968	0.455	1.000	0.728	0.961	0.643	0.634	32.000	0.910	1.150	0.000
CM1	0.883	0.040	1.000	0.519	0.806	0.080	0.072	28.990	0.160	3.900	0.070
KC3	0.801	0.005	1.000	0.503	0.670	0.010	0.009	16.000	0.020	4.000	0.040
MC2	0.772	0.432	0.988	0.708	0.764	0.637	0.534	7.900	2.160	2.940	0.230
MW1	0.939	0.260	1.000	0.629	0.908	0.367	0.351	22.990	0.520	1.600	0.060
PC1	0.916	0.052	1.000	0.526	0.862	0.119	0.112	61.980	0.310	5.840	0.050
PC2	0.973	0.000	1.000	0.500	0.951	0.000	0.000	71.000	0.000	2.000	0.030
abalone9v18	0.946	0.015	1.000	0.507	0.907	0.030	0.026	69.980	0.060	3.990	0.060
abalone19	0.993	0.000	1.000	0.500	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.916	0.147	0.999	0.572	0.875	0.214	0.200	72.076	0.441	2.899	0.050

V. OTHER PERFORMANCE METRICS FOR NONE USING NEURAL NET

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.980	0.500	1.000	0.750	0.977	0.707	0.700	47.000	1.000	1.010	0.000
glass2	0.909	0.000	1.000	0.500	0.842	0.000	0.000	20.000	0.000	2.000	0.000
ecoli4	0.969	0.470	1.000	0.735	0.963	0.665	0.655	32.000	0.940	1.090	0.000
CM1	0.767	0.255	0.854	0.511	0.740	0.155	0.017	24.770	1.020	3.450	6.930
KC3	0.727	0.323	0.871	0.518	0.644	0.186	0.034	13.930	1.290	3.380	4.580
MC2	0.567	0.718	0.520	0.583	0.468	0.336	0.185	4.160	3.590	1.930	4.590
MW1	0.795	0.210	0.851	0.513	0.789	0.122	0.024	19.580	0.420	1.710	5.410
PC1	0.801	0.260	0.863	0.529	0.798	0.205	0.048	53.510	1.560	5.020	12.500
PC2	0.962	0.055	0.988	0.514	0.949	0.060	0.017	70.150	0.110	1.940	2.620
abalone9v18	0.946	0.000	1.000	0.500	0.905	0.000	0.000	70.000	0.000	4.000	0.000
abalone19	0.993	0.000	1.000	0.500	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.856	0.254	0.904	0.559	0.824	0.221	0.153	70.009	0.903	2.594	3.330

VI. OTHER PERFORMANCE METRICS FOR NONE USING ADABOOST

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.975	0.525	0.994	0.760	0.974	0.720	0.657	46.730	1.050	0.950	0.270
glass2	0.933	0.485	0.978	0.732	0.928	0.661	0.563	19.560	0.970	1.030	0.440
ecoli4	0.969	0.500	0.998	0.749	0.965	0.706	0.680	31.930	1.000	1.000	0.070
CM1	0.835	0.208	0.921	0.564	0.821	0.378	0.147	26.710	0.830	3.170	2.300
KC3	0.817	0.208	0.970	0.588	0.766	0.395	0.277	15.520	0.830	3.210	0.530
MC2	0.646	0.396	0.803	0.599	0.624	0.540	0.213	6.420	1.980	3.050	1.640
MW1	0.900	0.200	0.960	0.580	0.882	0.277	0.182	22.090	0.400	1.600	0.920
PC1	0.924	0.300	0.984	0.642	0.908	0.527	0.413	61.010	1.800	4.200	1.000
PC2	0.965	0.005	0.992	0.499	0.950	0.007	-0.006	70.440	0.010	1.990	0.560
abalone9v18	0.955	0.313	0.992	0.652	0.944	0.512	0.436	69.440	1.250	2.750	0.560
abalone19	0.992	0.000	0.999	0.499	0.987	0.000	-0.002	414.570	0.000	3.000	0.430
Mean	0.901	0.285	0.963	0.624	0.886	0.429	0.324	71.311	0.920	2.359	0.793

VII. OTHER PERFORMANCE METRICS FOR NONE USING NAÏVE BAYES

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.384	1.000	0.358	0.679	0.675	0.580	0.215	16.830	2.000	0.000	30.170
glass2	0.345	1.000	0.279	0.639	0.611	0.528	0.184	5.580	2.000	0.000	14.420
ecoli4	0.876	0.955	0.871	0.913	0.934	0.906	0.544	27.880	1.910	0.090	4.120
CM1	0.851	0.270	0.931	0.600	0.837	0.498	0.223	26.990	1.080	2.920	2.010
KC3	0.824	0.128	0.999	0.563	0.736	0.225	0.205	15.980	0.510	3.490	0.020
MC2	0.768	0.400	0.998	0.699	0.767	0.632	0.535	7.980	2.000	3.000	0.020
MW1	0.861	0.500	0.893	0.696	0.891	0.668	0.312	20.530	1.000	1.000	2.470
PC1	0.895	0.333	0.950	0.642	0.890	0.563	0.306	58.890	2.000	4.000	3.110
PC2	0.922	0.335	0.939	0.637	0.948	0.441	0.200	66.660	0.670	1.330	4.340
abalone9v18	0.789	0.748	0.791	0.769	0.897	0.769	0.286	55.360	2.990	1.010	14.640
abalone19	0.863	0.183	0.868	0.526	0.958	0.292	0.010	360.120	0.550	2.450	54.880
Mean	0.762	0.532	0.807	0.669	0.831	0.555	0.275	60.255	1.519	1.754	11.836

VIII. OTHER PERFORMANCE METRICS FOR NONE USING NAÏVE BAYES

Dataset	Accuracy	Sensitivity	Specificity	AUC	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.709	0.680	0.710	0.695	0.818	0.555	0.255	33.390	1.360	0.640	13.610
glass2	0.852	0.830	0.854	0.842	0.854	0.760	0.581	17.080	1.660	0.340	2.920
ecoli4	0.864	0.330	0.898	0.614	0.884	0.322	0.177	28.720	0.660	1.340	3.280
CM1	0.879	0.000	1.000	0.500	0.791	0.000	0.000	29.000	0.000	4.000	0.000
KC3	0.800	0.000	1.000	0.500	0.667	0.000	0.000	16.000	0.000	4.000	0.000
MC2	0.615	0.000	1.000	0.500	0.410	0.000	0.000	8.000	0.000	5.000	0.000
MW1	0.920	0.000	1.000	0.500	0.860	0.000	0.000	23.000	0.000	2.000	0.000
PC1	0.893	0.432	0.938	0.685	0.896	0.630	0.359	58.130	2.590	3.410	3.870
PC2	0.973	0.000	1.000	0.500	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.959	0.500	0.985	0.743	0.955	0.702	0.553	68.950	2.000	2.000	1.050
abalone19	0.979	0.000	0.986	0.493	0.984	0.000	-0.010	409.110	0.000	3.000	5.890
Mean	0.858	0.252	0.943	0.597	0.825	0.270	0.174	69.307	0.752	2.521	2.784

APPENDIX D

OTHER PERFORMANCE METRICS FOR HCBST

I. OTHER PERFORMANCE METRICS FOR HCBST USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.050	12.010
glass2	0.889	0.705	0.640	0.936	0.898	0.669	0.395	18.720	1.280	1.200	8.580
ecoli4	0.950	0.855	0.820	0.978	0.949	0.839	0.542	31.300	1.640	1.000	3.810
CM1	0.805	0.650	0.793	0.900	0.816	0.636	0.224	26.090	3.170	3.520	16.100
KC3	0.711	0.567	0.700	0.871	0.714	0.549	0.110	13.940	2.800	3.880	9.070
MC2	0.670	0.645	0.808	0.755	0.669	0.628	0.304	6.040	4.040	2.650	6.960
MW1	0.791	0.729	0.830	0.812	0.869	0.681	0.289	18.670	1.660	1.660	13.970
PC1	0.719	0.745	0.938	0.778	0.808	0.725	0.280	48.230	5.630	5.320	42.120
PC2	0.869	0.766	0.970	0.884	0.935	0.734	0.203	62.750	1.940	1.280	39.480
abalone9v18	0.927	0.734	0.913	0.955	0.933	0.723	0.363	66.820	3.400	3.130	31.170
abalone19	0.904	0.771	0.980	0.907	0.971	0.756	0.144	376.200	2.940	1.370	201.190
Mean	0.840	0.742	0.854	0.889	0.869	0.722	0.350	65.069	2.773	2.278	34.951

II. OTHER PERFORMANCE METRICS FOR HCBST USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.160	3.910
glass2	0.829	0.554	0.590	0.901	0.816	0.296	0.077	18.010	1.180	1.830	9.750
ecoli4	0.971	0.952	0.980	1.000	0.967	0.950	0.696	32.000	1.960	1.000	3.810
CM1	0.746	0.635	0.773	0.806	0.772	0.518	0.195	23.360	3.090	2.730	15.300
KC3	0.724	0.594	0.783	0.864	0.709	0.405	0.193	13.830	2.320	3.370	6.970
MC2	0.665	0.632	0.705	0.794	0.627	0.492	0.285	6.350	3.160	3.980	4.930
MW1	0.862	0.814	0.830	0.892	0.912	0.789	0.508	20.510	1.660	0.950	8.480
PC1	0.757	0.713	0.858	0.789	0.824	0.643	0.287	48.940	5.150	3.590	30.080
PC2	0.828	0.698	0.840	0.841	0.914	0.626	0.151	59.720	1.680	1.310	33.400
abalone9v18	0.942	0.837	1.000	0.970	0.945	0.833	0.506	67.870	4.000	2.210	45.820
abalone19	0.878	0.918	1.000	0.882	0.966	0.915	0.212	365.860	3.000	2.000	357.750
Mean	0.836	0.759	0.851	0.885	0.859	0.679	0.374	63.950	2.655	2.103	47.291

III. OTHER PERFORMANCE METRICS FOR HCBST USING DECISION TREES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.991	0.954	0.995	0.999	0.990	0.948	0.874	46.940	1.990	0.640	14.590
glass2	0.880	0.725	0.745	0.916	0.893	0.681	0.382	18.310	1.490	1.430	8.570
ecoli4	0.964	0.747	0.590	0.993	0.962	0.711	0.652	31.790	1.180	1.000	3.810
CM1	0.822	0.670	0.723	0.897	0.825	0.660	0.236	26.000	2.890	2.910	11.630
KC3	0.822	0.601	0.974	0.994	0.770	0.504	0.294	15.900	1.540	3.860	4.600
MC2	0.765	0.726	0.911	0.961	0.761	0.714	0.503	7.690	3.750	3.420	3.200
MW1	0.893	0.725	0.897	0.947	0.888	0.701	0.309	21.770	1.440	1.450	7.310
PC1	0.911	0.725	0.936	0.970	0.896	0.717	0.341	60.150	4.640	4.310	22.480
PC2	0.928	0.674	0.625	0.949	0.947	0.593	0.138	67.400	1.250	1.740	19.640
abalone9v18	0.911	0.700	0.928	0.959	0.906	0.684	0.218	67.160	2.950	3.730	24.560
abalone19	0.920	0.781	0.913	0.923	0.974	0.767	0.120	383.220	2.740	1.790	154.560
Mean	0.892	0.730	0.840	0.955	0.892	0.698	0.370	67.848	2.351	2.389	24.995

IV. OTHER PERFORMANCE METRICS FOR HCBST USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.979	0.829	0.780	1.000	0.972	0.801	0.620	47.000	1.560	1.310	12.750
glass2	0.910	0.681	0.790	1.000	0.854	0.651	0.222	19.990	1.580	1.990	8.570
ecoli4	0.969	0.796	0.705	1.000	0.966	0.775	0.687	32.000	1.410	1.090	3.640
CM1	0.881	0.670	0.680	0.998	0.844	0.659	0.249	28.930	2.720	3.850	10.660
KC3	0.816	0.657	0.989	0.997	0.778	0.596	0.305	15.950	1.940	3.910	3.630
MC2	0.775	0.739	0.955	0.979	0.773	0.715	0.526	7.830	4.590	4.600	4.540
MW1	0.913	0.770	0.936	0.975	0.915	0.757	0.454	22.420	1.780	1.920	9.840
PC1	0.871	0.749	0.962	0.928	0.872	0.732	0.290	57.560	5.770	4.380	29.600
PC2	0.929	0.673	0.720	0.955	0.942	0.651	0.119	67.790	1.440	1.990	27.300
abalone9v18	0.915	0.705	0.926	0.966	0.902	0.697	0.208	67.590	3.510	3.960	37.420
abalone19	0.944	0.807	0.993	0.947	0.980	0.798	0.176	393.030	2.980	1.450	206.940
Mean	0.900	0.734	0.858	0.977	0.891	0.712	0.351	69.099	2.662	2.768	32.263

V. OTHER PERFORMANCE METRICS FOR HCBST USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	1.760	11.130
glass2	0.909	0.554	0.650	1.000	0.842	0.326	0.068	20.000	1.300	2.000	10.830
ecoli4	0.971	0.949	0.975	1.000	0.967	0.946	0.696	32.000	1.950	2.000	5.790
CM1	0.709	0.556	0.668	0.769	0.704	0.359	0.086	22.310	2.670	2.910	17.520
KC3	0.671	0.552	0.690	0.779	0.610	0.266	0.099	12.460	2.760	3.040	9.970
MC2	0.636	0.645	0.892	0.761	0.591	0.506	0.318	6.090	4.460	3.380	5.500
MW1	0.752	0.610	0.940	0.788	0.819	0.382	0.132	18.130	1.880	1.350	19.170
PC1	0.603	0.617	0.885	0.607	0.714	0.500	0.178	37.630	5.310	2.870	49.010
PC2	0.695	0.534	0.765	0.707	0.813	0.277	0.032	50.170	1.530	1.480	53.770
abalone9v18	0.941	0.828	1.000	0.981	0.944	0.823	0.496	68.680	4.000	3.820	65.850
abalone19	0.991	0.877	1.000	0.998	0.987	0.869	0.162	414.330	3.000	2.960	409.900
Mean	0.807	0.702	0.860	0.854	0.817	0.568	0.297	66.255	2.805	2.506	59.858

VI. OTHER PERFORMANCE METRICS FOR HCBST USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.972	0.865	0.915	0.990	0.972	0.857	0.635	46.540	1.830	0.940	12.970
glass2	0.930	0.763	0.805	0.973	0.927	0.711	0.560	19.450	1.610	0.990	7.810
ecoli4	0.968	0.758	0.640	0.998	0.965	0.735	0.678	31.920	1.280	1.010	4.550
CM1	0.845	0.703	0.703	0.923	0.845	0.688	0.314	26.760	2.810	3.060	10.450
KC3	0.819	0.631	0.941	0.961	0.781	0.532	0.312	15.370	1.780	3.210	4.690
MC2	0.665	0.634	0.813	0.866	0.649	0.626	0.270	6.930	2.970	3.360	3.430
MW1	0.889	0.724	0.908	0.955	0.873	0.702	0.289	21.970	1.510	1.750	7.060
PC1	0.908	0.759	0.916	0.967	0.894	0.753	0.326	59.960	4.980	4.260	21.180
PC2	0.934	0.750	0.765	0.959	0.944	0.711	0.184	68.070	1.530	1.910	18.850
abalone9v18	0.944	0.717	0.955	0.991	0.930	0.708	0.287	69.360	2.740	3.500	19.060
abalone19	0.964	0.790	0.926	0.968	0.983	0.782	0.164	401.900	2.650	1.970	134.320
Mean	0.894	0.736	0.844	0.959	0.888	0.709	0.365	69.839	2.335	2.360	22.215

VII. OTHER PERFORMANCE METRICS FOR HCBST USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.970	0.985	1.000	0.969	0.987	0.984	0.891	45.550	2.000	0.000	28.660
glass2	0.856	0.716	1.000	0.940	0.830	0.665	0.254	18.790	2.000	1.950	14.950
ecoli4	0.937	0.919	0.955	0.973	0.947	0.913	0.586	31.120	1.910	1.700	4.700
CM1	0.850	0.653	0.860	0.930	0.838	0.591	0.296	26.980	3.440	2.920	25.850
KC3	0.874	0.697	0.500	0.998	0.850	0.591	0.539	15.970	2.000	3.340	13.690
MC2	0.768	0.699	0.530	0.999	0.768	0.632	0.537	7.990	2.650	3.720	5.990
MW1	0.840	0.840	0.885	0.870	0.899	0.829	0.466	20.000	1.770	1.390	16.040
PC1	0.903	0.641	0.872	0.961	0.893	0.563	0.318	59.580	5.230	4.320	56.160
PC2	0.655	0.635	0.805	0.663	0.785	0.509	0.121	47.070	1.610	1.240	50.780
abalone9v18	0.798	0.741	0.920	0.833	0.881	0.736	0.250	58.300	3.680	3.760	45.310
abalone19	0.757	0.735	0.997	0.758	0.931	0.706	0.082	314.510	2.990	1.520	237.260
Mean	0.837	0.751	0.848	0.899	0.874	0.702	0.395	58.715	2.662	2.351	45.399

VIII. OTHER PERFORMANCE METRICS FOR HCBST USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.812	0.811	0.970	0.820	0.885	0.726	0.436	38.560	1.940	0.950	23.530
glass2	0.888	0.892	0.950	0.930	0.875	0.843	0.624	18.600	1.900	1.860	4.120
ecoli4	0.913	0.695	0.655	0.963	0.896	0.527	0.281	30.820	1.310	1.780	10.640
CM1	0.883	0.580	0.835	1.000	0.817	0.391	0.150	29.000	3.340	4.000	21.850
KC3	0.800	0.515	1.000	1.000	0.669	0.250	0.035	16.000	3.720	4.000	14.890
MC2	0.672	0.614	1.000	1.000	0.595	0.402	0.281	8.000	4.990	5.000	7.930
MW1	0.927	0.552	1.000	1.000	0.879	0.181	0.140	23.000	1.780	2.000	19.740
PC1	0.915	0.692	0.998	1.000	0.898	0.679	0.353	62.000	4.300	6.000	20.600
PC2	0.973	0.596	0.510	1.000	0.951	0.373	0.072	71.000	1.020	2.000	26.330
abalone9v18	0.940	0.838	0.949	0.966	0.950	0.831	0.535	67.610	3.140	3.280	11.330
abalone19	0.976	0.731	0.955	0.981	0.986	0.724	0.159	406.960	2.420	2.170	148.090
Mean	0.882	0.683	0.893	0.969	0.855	0.539	0.279	70.141	2.715	3.004	28.095

APPENDIX E

OTHER PERFORMANCE METRICS FOR ADASYN

I. OTHER PERFORMANCE METRICS FOR ADASYN USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.990	0.995	1.000	0.989	0.992	0.995	0.906	46.500	2.000	0.000	5.510
glass2	0.881	0.683	0.500	0.927	0.887	0.653	0.324	18.530	1.000	1.140	2.970
ecoli4	0.943	0.735	0.500	0.970	0.943	0.697	0.486	31.050	1.000	1.000	1.200
CM1	0.815	0.705	0.730	0.907	0.816	0.696	0.278	26.290	2.920	3.470	9.290
KC3	0.726	0.537	0.340	0.901	0.690	0.476	0.066	14.410	1.360	3.890	4.270
MC2	0.492	0.509	0.580	0.438	0.507	0.492	0.014	3.500	2.900	2.130	5.100
MW1	0.853	0.662	0.570	0.920	0.864	0.613	0.202	21.170	1.140	1.900	5.760
PC1	0.878	0.586	0.427	0.957	0.848	0.543	0.109	59.350	2.560	5.760	15.780
PC2	0.969	0.561	0.250	0.997	0.950	0.328	0.055	70.760	0.500	2.000	9.050
abalone9v18	0.969	0.745	0.508	0.996	0.965	0.701	0.653	69.710	2.030	2.050	3.500
abalone19	0.982	0.495	0.010	0.989	0.985	0.017	-0.009	410.460	0.030	3.000	13.970
Mean	0.864	0.656	0.492	0.908	0.859	0.565	0.280	70.157	1.585	2.395	6.945

II. OTHER PERFORMANCE METRICS FOR ADASYN USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.998	0.978	0.955	1.000	0.998	0.974	0.973	47.000	1.910	0.260	2.910
glass2	0.799	0.568	0.655	0.864	0.757	0.303	0.099	17.270	1.310	1.700	10.380
ecoli4	0.971	0.750	0.520	1.000	0.967	0.708	0.696	32.000	1.040	1.000	1.000
CM1	0.774	0.660	0.643	0.845	0.778	0.529	0.211	24.500	2.570	2.950	9.350
KC3	0.711	0.592	0.460	0.839	0.679	0.381	0.174	13.420	1.840	3.200	4.430
MC2	0.562	0.512	0.358	0.758	0.473	0.273	0.024	6.060	1.790	3.750	2.670
MW1	0.910	0.781	0.720	0.937	0.930	0.732	0.589	21.560	1.440	0.860	3.720
PC1	0.833	0.687	0.580	0.888	0.853	0.602	0.285	55.040	3.480	4.420	12.820
PC2	0.892	0.598	0.290	0.914	0.938	0.377	0.095	64.860	0.580	1.710	8.860
abalone9v18	0.973	0.856	0.750	1.000	0.970	0.849	0.695	70.000	3.000	2.090	6.870
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	1.920
Mean	0.856	0.680	0.539	0.913	0.848	0.521	0.349	69.701	1.724	2.267	5.903

III. OTHER PERFORMANCE METRICS FOR ADASYN USING DECISION TREE

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.985	0.938	0.990	0.998	0.984	0.935	0.793	46.920	1.980	0.800	5.980
glass2	0.886	0.707	0.600	0.935	0.886	0.666	0.336	18.690	1.200	1.240	4.760
ecoli4	0.967	0.748	0.500	0.996	0.964	0.706	0.670	31.880	1.000	1.000	0.620
CM1	0.811	0.745	0.725	0.859	0.844	0.739	0.358	24.900	2.900	2.130	7.370
KC3	0.833	0.599	0.238	0.990	0.786	0.413	0.350	15.840	0.950	3.410	0.710
MC2	0.702	0.687	0.624	0.750	0.705	0.677	0.382	6.000	3.120	1.980	2.320
MW1	0.891	0.699	0.530	0.939	0.899	0.628	0.332	21.590	1.060	1.330	3.080
PC1	0.903	0.736	0.658	0.962	0.893	0.729	0.332	59.650	3.950	4.250	11.540
PC2	0.961	0.588	0.230	0.984	0.955	0.314	0.146	69.880	0.460	1.760	4.210
abalone9v18	0.935	0.691	0.493	0.972	0.929	0.639	0.299	68.060	1.970	2.850	7.710
abalone19	0.990	0.508	0.027	0.997	0.987	0.045	0.020	413.680	0.080	3.000	15.550
Mean	0.897	0.695	0.510	0.944	0.894	0.590	0.365	70.645	1.697	2.159	5.805

IV. OTHER PERFORMANCE METRICS FOR ADASYN USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.974	0.713	0.480	1.000	0.962	0.612	0.493	47.000	0.960	1.490	2.500
glass2	0.909	0.647	0.570	1.000	0.867	0.603	0.254	19.990	1.140	1.990	5.860
ecoli4	0.968	0.747	0.510	0.999	0.963	0.705	0.656	31.980	1.020	1.130	0.900
CM1	0.882	0.709	0.648	0.996	0.847	0.702	0.305	28.890	2.590	3.820	6.690
KC3	0.800	0.571	0.208	0.999	0.735	0.333	0.164	15.980	0.830	3.970	1.110
MC2	0.792	0.758	0.616	0.921	0.792	0.737	0.565	7.370	3.080	2.310	0.860
MW1	0.942	0.778	0.620	0.995	0.935	0.747	0.557	22.880	1.240	1.390	1.710
PC1	0.916	0.654	0.520	0.998	0.879	0.633	0.225	61.880	3.120	5.600	13.190
PC2	0.972	0.509	0.030	0.999	0.951	0.042	0.019	70.960	0.060	2.000	0.840
abalone9v18	0.945	0.608	0.333	0.997	0.920	0.532	0.200	69.800	1.330	3.860	8.220
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.150
Mean	0.918	0.654	0.412	0.991	0.894	0.513	0.312	71.975	1.397	2.778	3.821

V. OTHER PERFORMANCE METRICS FOR ADASYN USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.987	0.962	0.950	1.000	0.987	0.958	0.831	47.000	1.900	2.000	1.180
glass2	0.909	0.558	0.730	1.000	0.842	0.472	0.075	20.000	1.460	2.000	12.290
ecoli4	0.971	0.925	0.905	1.000	0.967	0.917	0.696	32.000	1.810	1.180	1.770
CM1	0.661	0.559	0.600	0.707	0.677	0.329	0.091	20.500	2.400	2.680	13.990
KC3	0.648	0.551	0.503	0.749	0.594	0.290	0.096	11.980	2.010	3.030	6.850
MC2	0.581	0.601	0.796	0.515	0.507	0.383	0.221	4.120	3.980	1.570	5.020
MW1	0.728	0.689	0.690	0.772	0.805	0.553	0.253	17.750	1.380	1.550	7.830
PC1	0.742	0.617	0.647	0.787	0.779	0.492	0.158	48.790	3.880	4.410	25.580
PC2	0.934	0.533	0.235	0.958	0.942	0.205	0.024	68.020	0.470	1.850	11.960
abalone9v18	0.974	0.845	0.750	1.000	0.974	0.836	0.749	70.000	3.000	4.000	7.870
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.830	0.667	0.619	0.862	0.824	0.494	0.290	68.651	2.026	2.479	8.576

VI. OTHER PERFORMANCE METRICS FOR ADASYN USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.968	0.749	0.545	0.987	0.967	0.709	0.581	46.400	1.090	0.980	3.070
glass2	0.929	0.848	0.790	0.974	0.927	0.826	0.581	19.470	1.580	1.030	1.890
ecoli4	0.969	0.749	0.500	0.998	0.966	0.707	0.685	31.950	1.000	1.030	0.600
CM1	0.841	0.730	0.625	0.888	0.859	0.715	0.367	25.760	2.500	2.380	5.070
KC3	0.815	0.599	0.258	0.959	0.776	0.450	0.302	15.340	1.030	3.100	1.180
MC2	0.658	0.625	0.482	0.768	0.650	0.599	0.264	6.140	2.410	2.720	1.970
MW1	0.894	0.603	0.305	0.956	0.882	0.386	0.193	21.980	0.610	1.700	2.300
PC1	0.920	0.675	0.427	0.977	0.907	0.617	0.405	60.550	2.560	4.010	4.790
PC2	0.962	0.523	0.070	0.988	0.950	0.097	0.034	70.150	0.140	1.970	1.750
abalone9v18	0.953	0.697	0.435	0.985	0.947	0.637	0.469	68.980	1.740	2.500	3.400
abalone19	0.990	0.565	0.140	0.997	0.987	0.241	0.103	413.940	0.420	3.000	4.610
Mean	0.900	0.669	0.416	0.952	0.892	0.544	0.362	70.969	1.371	2.220	2.785

VII. OTHER PERFORMANCE METRICS FOR ADASYN USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.410	0.692	1.000	0.385	0.704	0.607	0.221	18.080	2.000	0.000	30.020
glass2	0.342	0.638	1.000	0.277	0.609	0.525	0.183	5.530	2.000	0.000	14.740
ecoli4	0.876	0.909	0.950	0.872	0.933	0.902	0.523	27.900	1.900	0.120	5.570
CM1	0.852	0.683	0.473	0.926	0.858	0.641	0.352	26.850	1.890	2.900	3.090
KC3	0.888	0.738	0.490	0.996	0.880	0.694	0.623	15.940	1.960	3.100	0.220
MC2	0.766	0.698	0.400	0.995	0.765	0.631	0.530	7.960	2.000	3.000	0.050
MW1	0.844	0.687	0.500	0.874	0.884	0.661	0.283	20.110	1.000	1.000	3.210
PC1	0.886	0.637	0.333	0.940	0.885	0.560	0.280	58.270	2.000	4.000	4.300
PC2	0.900	0.708	0.505	0.912	0.946	0.677	0.262	64.720	1.010	1.000	13.040
abalone9v18	0.781	0.766	0.750	0.782	0.894	0.766	0.279	54.760	3.000	1.000	24.970
abalone19	0.786	0.548	0.327	0.790	0.938	0.477	0.020	327.710	0.980	2.100	146.330
Mean	0.757	0.700	0.612	0.795	0.845	0.649	0.323	57.075	1.795	1.656	22.322

VIII. OTHER PERFORMANCE METRICS FOR ADASYN USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.811	0.656	0.580	0.829	0.885	0.481	0.187	38.970	1.160	1.290	12.890
glass2	0.863	0.835	0.800	0.870	0.860	0.746	0.606	17.390	1.600	0.530	3.120
ecoli4	0.896	0.610	0.315	0.935	0.897	0.325	0.191	29.920	0.630	1.640	3.260
CM1	0.886	0.535	0.078	0.999	0.817	0.154	0.124	28.970	0.310	3.750	0.220
KC3	0.800	0.500	0.000	1.000	0.667	0.000	0.000	16.000	0.000	4.000	0.000
MC2	0.615	0.500	0.000	1.000	0.410	0.000	0.000	8.000	0.000	5.000	0.000
MW1	0.920	0.500	0.000	1.000	0.860	0.000	0.000	23.000	0.000	2.000	0.000
PC1	0.907	0.688	0.425	0.960	0.902	0.628	0.386	59.530	2.550	3.960	3.090
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.965	0.863	0.750	0.982	0.967	0.855	0.687	68.730	3.000	1.680	4.800
abalone19	0.983	0.617	0.273	0.990	0.985	0.438	0.103	411.030	0.820	3.000	16.280
Mean	0.874	0.618	0.293	0.960	0.836	0.330	0.208	70.231	0.915	2.623	3.969

APPENDIX F

OTHER PERFORMANCE METRICS FOR ROS

I. OTHER PERFORMANCE METRICS FOR ROS USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.996	0.998	1.000	0.996	0.997	0.998	0.962	46.800	2.000	0.000	0.410
glass2	0.890	0.702	0.500	0.935	0.892	0.672	0.356	18.700	1.000	1.130	1.940
ecoli4	0.941	0.734	0.500	0.969	0.942	0.696	0.472	31.000	1.000	1.000	1.170
CM1	0.803	0.629	0.455	0.874	0.818	0.599	0.201	25.350	1.820	2.840	5.710
KC3	0.713	0.513	0.268	0.854	0.682	0.431	0.026	13.660	1.070	3.410	4.580
MC2	0.691	0.669	0.594	0.764	0.690	0.659	0.347	6.110	2.970	2.130	2.180
MW1	0.854	0.669	0.450	0.906	0.884	0.600	0.267	20.830	0.900	1.840	3.090
PC1	0.873	0.531	0.178	0.956	0.842	0.379	0.054	59.300	1.070	5.920	7.170
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.910
abalone9v18	0.967	0.744	0.498	0.996	0.962	0.702	0.618	69.730	1.990	2.200	1.090
abalone19	0.983	0.495	0.000	0.990	0.985	0.000	-0.008	410.870	0.000	3.000	9.170
Mean	0.880	0.653	0.404	0.931	0.877	0.521	0.300	70.305	1.256	2.315	3.402

II. OTHER PERFORMANCE METRICS FOR ROS USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.020	0.050
glass2	0.827	0.560	0.520	0.900	0.759	0.256	0.082	18.000	1.040	1.800	8.000
ecoli4	0.971	0.750	0.500	1.000	0.967	0.707	0.696	32.000	1.000	1.000	1.000
CM1	0.781	0.634	0.550	0.857	0.775	0.447	0.176	24.840	2.200	3.060	8.160
KC3	0.720	0.566	0.345	0.855	0.675	0.266	0.112	13.680	1.380	3.280	3.620
MC2	0.565	0.515	0.300	0.746	0.479	0.254	0.024	5.970	1.500	3.670	2.160
MW1	0.909	0.766	0.690	0.940	0.926	0.715	0.567	21.630	1.380	0.900	3.650
PC1	0.846	0.700	0.605	0.904	0.855	0.622	0.295	56.020	3.630	4.510	12.710
PC2	0.909	0.551	0.180	0.932	0.938	0.224	0.043	66.170	0.360	1.850	11.760
abalone9v18	0.973	0.855	0.750	1.000	0.970	0.846	0.697	70.000	3.000	3.130	6.860
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	1.670
Mean	0.863	0.672	0.495	0.921	0.848	0.485	0.336	70.028	1.590	2.384	5.422

III. OTHER PERFORMANCE METRICS FOR ROS USING DECISION TREE

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.984	0.860	0.760	0.997	0.983	0.837	0.785	46.850	1.520	0.630	3.130
glass2	0.893	0.726	0.550	0.939	0.899	0.664	0.403	18.770	1.100	1.130	4.650
ecoli4	0.970	0.748	0.500	1.000	0.967	0.705	0.689	32.000	1.000	1.030	0.260
CM1	0.789	0.677	0.618	0.847	0.824	0.663	0.252	24.570	2.470	2.720	7.710
KC3	0.808	0.543	0.130	0.985	0.722	0.225	0.155	15.760	0.520	3.600	1.390
MC2	0.748	0.708	0.562	0.884	0.746	0.667	0.482	7.070	2.810	2.340	1.690
MW1	0.905	0.728	0.530	0.952	0.911	0.647	0.420	21.900	1.060	1.440	2.950
PC1	0.910	0.723	0.718	0.966	0.900	0.718	0.367	59.890	4.310	3.990	16.850
PC2	0.972	0.558	0.120	0.996	0.958	0.169	0.150	70.700	0.240	1.930	2.780
abalone9v18	0.943	0.676	0.440	0.986	0.927	0.610	0.285	69.010	1.760	3.250	6.120
abalone19	0.988	0.557	0.123	0.996	0.987	0.213	0.098	413.170	0.370	2.990	8.090
Mean	0.901	0.682	0.459	0.959	0.893	0.556	0.371	70.881	1.560	2.277	5.056

IV. OTHER PERFORMANCE METRICS FOR ROS USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.977	0.753	0.510	1.000	0.972	0.654	0.624	47.000	1.020	1.420	0.300
glass2	0.914	0.604	0.460	0.999	0.870	0.533	0.187	19.970	0.920	1.940	5.060
ecoli4	0.969	0.752	0.520	0.999	0.966	0.712	0.687	31.980	1.040	1.080	0.620
CM1	0.885	0.650	0.500	0.995	0.849	0.629	0.278	28.860	2.000	3.780	6.010
KC3	0.797	0.518	0.103	0.994	0.694	0.179	0.034	15.910	0.410	3.970	1.190
MC2	0.789	0.745	0.554	0.980	0.789	0.712	0.563	7.840	2.770	2.710	0.600
MW1	0.945	0.750	0.550	0.997	0.933	0.718	0.546	22.920	1.100	1.340	1.140
PC1	0.918	0.634	0.447	0.998	0.888	0.598	0.283	61.900	2.680	5.650	11.060
PC2	0.972	0.500	0.005	1.000	0.951	0.007	0.000	70.990	0.010	2.000	0.590
abalone9v18	0.945	0.613	0.308	0.998	0.920	0.522	0.181	69.850	1.230	3.920	5.710
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.030
Mean	0.919	0.638	0.360	0.996	0.893	0.478	0.308	72.020	1.198	2.801	2.937

V. OTHER PERFORMANCE METRICS FOR ROS USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	2.000	0.000
glass2	0.909	0.503	0.505	1.000	0.842	0.381	0.008	20.000	1.010	2.000	10.190
ecoli4	0.971	0.806	0.655	1.000	0.967	0.780	0.696	32.000	1.310	1.000	1.360
CM1	0.661	0.575	0.658	0.709	0.664	0.375	0.116	20.570	2.630	2.760	14.700
KC3	0.624	0.565	0.555	0.687	0.588	0.332	0.134	10.990	2.220	2.510	8.000
MC2	0.548	0.608	0.868	0.400	0.452	0.338	0.226	3.200	4.340	1.270	5.690
MW1	0.777	0.643	0.640	0.823	0.800	0.486	0.179	18.940	1.280	1.510	8.510
PC1	0.759	0.600	0.578	0.809	0.780	0.467	0.128	50.160	3.470	4.540	23.480
PC2	0.936	0.566	0.280	0.961	0.940	0.296	0.063	68.200	0.560	1.870	10.550
abalone9v18	0.975	0.838	0.750	1.000	0.973	0.829	0.728	70.000	3.000	4.000	7.800
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.832	0.655	0.590	0.854	0.818	0.480	0.298	68.733	1.984	2.405	8.207

VI. OTHER PERFORMANCE METRICS FOR ROS USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.974	0.760	0.530	0.993	0.973	0.721	0.647	46.690	1.060	0.970	1.020
glass2	0.929	0.746	0.525	0.975	0.924	0.677	0.538	19.500	1.050	1.060	0.700
ecoli4	0.969	0.749	0.500	0.999	0.966	0.707	0.685	31.960	1.000	1.010	0.090
CM1	0.825	0.608	0.375	0.906	0.824	0.546	0.182	26.270	1.500	3.030	4.610
KC3	0.816	0.607	0.283	0.962	0.773	0.479	0.291	15.390	1.130	3.080	1.150
MC2	0.639	0.596	0.412	0.791	0.621	0.546	0.207	6.330	2.060	3.080	1.790
MW1	0.903	0.601	0.250	0.966	0.885	0.345	0.212	22.210	0.500	1.690	1.500
PC1	0.920	0.691	0.450	0.982	0.905	0.641	0.399	60.870	2.700	4.290	5.200
PC2	0.965	0.511	0.035	0.992	0.950	0.049	0.024	70.420	0.070	1.980	1.140
abalone9v18	0.954	0.721	0.488	0.991	0.942	0.674	0.418	69.400	1.950	2.840	3.250
abalone19	0.991	0.513	0.037	0.998	0.987	0.063	0.020	414.360	0.110	3.000	4.840
Mean	0.899	0.646	0.353	0.960	0.886	0.495	0.329	71.218	1.194	2.366	2.299

VII. OTHER PERFORMANCE METRICS FOR ROS USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.405	0.690	1.000	0.380	0.696	0.600	0.221	17.860	2.000	0.000	32.350
glass2	0.338	0.636	1.000	0.272	0.604	0.521	0.181	5.440	2.000	0.000	14.930
ecoli4	0.875	0.913	0.955	0.870	0.934	0.906	0.540	27.850	1.910	0.090	7.400
CM1	0.861	0.602	0.278	0.944	0.844	0.501	0.249	27.380	1.110	2.960	2.190
KC3	0.856	0.646	0.295	0.999	0.808	0.454	0.415	15.990	1.180	3.370	0.070
MC2	0.768	0.699	0.400	0.998	0.767	0.632	0.535	7.980	2.000	3.000	0.020
MW1	0.845	0.687	0.500	0.875	0.884	0.661	0.285	20.120	1.000	1.000	3.720
PC1	0.897	0.643	0.333	0.952	0.891	0.563	0.314	59.020	2.000	4.000	4.040
PC2	0.924	0.727	0.535	0.935	0.954	0.693	0.318	66.410	1.070	1.090	6.540
abalone9v18	0.784	0.768	0.750	0.786	0.896	0.768	0.283	55.020	3.000	1.000	22.370
abalone19	0.835	0.547	0.333	0.840	0.951	0.492	0.019	348.500	1.000	2.440	143.910
Mean	0.763	0.687	0.580	0.805	0.839	0.617	0.305	59.234	1.661	1.723	21.595

VIII. OTHER PERFORMANCE METRICS FOR ROS USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.782	0.695	0.655	0.792	0.870	0.574	0.256	37.220	1.310	0.910	12.890
glass2	0.880	0.870	0.885	0.888	0.883	0.818	0.618	17.760	1.770	0.460	3.040
ecoli4	0.893	0.620	0.320	0.933	0.900	0.330	0.183	29.840	0.640	1.570	3.580
CM1	0.882	0.532	0.073	0.995	0.815	0.144	0.113	28.860	0.290	3.770	0.260
KC3	0.800	0.500	0.000	1.000	0.667	0.000	0.000	16.000	0.000	4.000	0.000
MC2	0.615	0.500	0.000	1.000	0.410	0.000	0.000	8.000	0.000	5.000	0.000
MW1	0.920	0.500	0.000	1.000	0.860	0.000	0.000	23.000	0.000	2.000	0.000
PC1	0.907	0.686	0.430	0.956	0.902	0.631	0.386	59.270	2.580	3.590	3.580
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.963	0.861	0.750	0.985	0.964	0.854	0.656	68.950	3.000	1.970	3.700
abalone19	0.978	0.596	0.233	0.985	0.984	0.389	0.080	408.820	0.700	3.000	17.080
Mean	0.872	0.624	0.304	0.958	0.837	0.340	0.208	69.884	0.935	2.570	4.012

APPENDIX G

OTHER PERFORMANCE METRICS FOR RUS

I. OTHER PERFORMANCE METRICS FOR RUS USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.000	12.670
glass2	0.896	0.673	0.585	0.946	0.893	0.561	0.351	18.910	1.170	1.200	10.090
ecoli4	0.947	0.858	0.835	0.975	0.946	0.843	0.524	31.190	1.670	1.000	3.840
CM1	0.851	0.590	0.553	0.951	0.820	0.574	0.143	27.590	2.210	3.520	10.810
KC3	0.761	0.608	0.558	0.943	0.730	0.594	0.187	15.090	2.230	3.990	5.480
MC2	0.676	0.655	0.586	0.746	0.676	0.644	0.320	5.970	2.930	2.180	3.150
MW1	0.854	0.682	0.755	0.929	0.854	0.656	0.210	21.360	1.510	2.000	9.010
PC1	0.886	0.486	0.265	0.972	0.840	0.388	-0.037	60.280	1.590	6.000	20.290
PC2	0.973	0.703	0.655	1.000	0.951	0.655	0.150	71.000	1.310	2.000	17.670
abalone9v18	0.970	0.725	0.635	1.000	0.965	0.685	0.647	70.000	2.540	2.240	19.330
abalone19	0.992	0.500	0.040	0.999	0.987	0.063	-0.001	414.690	0.120	3.000	34.880
Mean	0.891	0.680	0.588	0.951	0.878	0.606	0.318	71.189	1.753	2.466	13.384

II. OTHER PERFORMANCE METRICS FOR RUS USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.020	1.010
glass2	0.861	0.555	0.615	0.941	0.795	0.265	0.080	18.820	1.230	1.880	10.110
ecoli4	0.971	0.922	0.980	1.000	0.967	0.917	0.696	32.000	1.960	1.000	4.350
CM1	0.764	0.575	0.365	0.824	0.783	0.368	0.109	23.910	1.460	2.830	7.740
KC3	0.723	0.549	0.425	0.863	0.658	0.260	0.098	13.800	1.700	3.340	5.230
MC2	0.562	0.527	0.376	0.728	0.488	0.282	0.056	5.820	1.880	3.630	2.570
MW1	0.918	0.746	0.570	0.953	0.930	0.664	0.556	21.910	1.140	0.970	4.520
PC1	0.838	0.609	0.458	0.895	0.850	0.466	0.157	55.510	2.750	4.520	15.200
PC2	0.895	0.639	0.500	0.913	0.938	0.478	0.113	64.830	1.000	1.500	15.720
abalone9v18	0.973	0.815	0.750	1.000	0.970	0.806	0.697	70.000	3.000	4.000	13.570
abalone19	0.993	0.506	0.023	1.000	0.987	0.040	0.004	415.000	0.070	3.000	5.100
Mean	0.863	0.677	0.551	0.920	0.851	0.504	0.324	69.873	1.654	2.426	7.738

III. OTHER PERFORMANCE METRICS FOR RUS USING DECISION TREES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.989	0.897	0.960	1.000	0.987	0.885	0.833	46.990	1.920	0.550	15.300
glass2	0.890	0.762	0.710	0.937	0.904	0.675	0.444	18.730	1.420	1.160	8.430
ecoli4	0.968	0.749	0.560	0.998	0.965	0.706	0.678	31.920	1.120	1.000	4.490
CM1	0.808	0.663	0.705	0.874	0.825	0.648	0.223	25.350	2.820	2.700	11.050
KC3	0.818	0.581	0.373	0.989	0.744	0.498	0.222	15.830	1.490	3.460	3.700
MC2	0.770	0.727	0.612	0.913	0.771	0.694	0.528	7.300	3.060	2.290	3.170
MW1	0.901	0.782	0.775	0.949	0.910	0.762	0.434	21.820	1.550	1.290	7.460
PC1	0.919	0.704	0.745	0.980	0.903	0.696	0.382	60.760	4.470	4.300	20.870
PC2	0.972	0.639	0.480	0.996	0.959	0.549	0.212	70.730	0.960	1.750	14.360
abalone9v18	0.937	0.680	0.635	0.981	0.922	0.661	0.216	68.640	2.540	3.310	20.470
abalone19	0.988	0.624	0.343	0.995	0.986	0.470	0.072	412.950	1.030	3.000	39.880
Mean	0.905	0.710	0.627	0.965	0.898	0.658	0.386	71.002	2.035	2.255	13.562

IV. OTHER PERFORMANCE METRICS FOR RUS USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.975	0.748	0.775	1.000	0.964	0.724	0.511	47.000	1.550	1.330	13.160
glass2	0.909	0.649	0.745	1.000	0.860	0.615	0.190	19.990	1.490	2.000	8.940
ecoli4	0.968	0.769	0.655	1.000	0.962	0.746	0.645	31.990	1.310	1.100	3.760
CM1	0.883	0.653	0.625	0.998	0.829	0.642	0.213	28.930	2.500	3.850	9.570
KC3	0.798	0.672	0.528	0.996	0.774	0.614	0.314	15.940	2.110	3.980	3.350
MC2	0.778	0.721	0.630	0.969	0.776	0.681	0.545	7.750	3.150	2.630	2.150
MW1	0.944	0.768	0.755	0.997	0.934	0.752	0.543	22.920	1.510	1.540	5.030
PC1	0.914	0.678	0.688	0.999	0.882	0.671	0.265	61.910	4.130	5.820	20.590
PC2	0.972	0.645	0.470	1.000	0.951	0.568	0.122	70.970	0.940	2.000	12.760
abalone9v18	0.944	0.682	0.663	0.998	0.911	0.674	0.178	69.840	2.650	3.960	20.950
abalone19	0.993	0.520	0.060	1.000	0.987	0.102	0.021	415.000	0.180	3.000	8.320
Mean	0.916	0.682	0.599	0.996	0.894	0.617	0.322	72.022	1.956	2.837	9.871

V. OTHER PERFORMANCE METRICS FOR RUS USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	2.000	0.080
glass2	0.909	0.578	0.750	1.000	0.842	0.368	0.101	20.000	1.500	2.000	11.880
ecoli4	0.971	0.930	0.980	1.000	0.967	0.926	0.696	32.000	1.960	2.000	3.830
CM1	0.708	0.527	0.760	0.774	0.708	0.212	0.045	22.450	3.040	3.080	21.000
KC3	0.675	0.548	0.740	0.778	0.608	0.245	0.089	12.450	2.960	2.940	10.320
MC2	0.536	0.593	0.840	0.416	0.439	0.322	0.208	3.330	4.200	1.370	5.490
MW1	0.770	0.600	0.825	0.824	0.798	0.336	0.132	18.960	1.650	1.720	14.710
PC1	0.789	0.524	0.725	0.846	0.792	0.204	0.042	52.460	4.350	4.790	43.580
PC2	0.935	0.554	0.680	0.959	0.941	0.321	0.049	68.090	1.360	1.850	40.620
abalone9v18	0.967	0.797	0.750	1.000	0.963	0.789	0.626	70.000	3.000	4.000	20.460
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	1.040
Mean	0.841	0.650	0.732	0.873	0.822	0.429	0.272	69.249	2.365	2.614	15.728

VI. OTHER PERFORMANCE METRICS FOR RUS USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.973	0.818	0.880	0.992	0.972	0.807	0.640	46.630	1.760	0.940	13.400
glass2	0.935	0.740	0.725	0.979	0.929	0.679	0.568	19.580	1.450	1.070	7.810
ecoli4	0.968	0.749	0.675	0.997	0.965	0.730	0.674	31.900	1.350	1.010	5.680
CM1	0.826	0.687	0.678	0.907	0.821	0.678	0.269	26.290	2.710	3.040	10.510
KC3	0.807	0.606	0.490	0.955	0.764	0.575	0.264	15.280	1.960	3.130	4.590
MC2	0.668	0.633	0.540	0.784	0.659	0.608	0.282	6.270	2.700	2.590	3.190
MW1	0.878	0.772	0.765	0.936	0.893	0.734	0.388	21.530	1.530	1.590	7.440
PC1	0.920	0.709	0.758	0.982	0.902	0.700	0.375	60.890	4.550	4.310	21.130
PC2	0.964	0.700	0.600	0.991	0.950	0.629	0.158	70.350	1.200	1.990	14.180
abalone9v18	0.954	0.680	0.590	0.990	0.944	0.641	0.436	69.280	2.360	2.830	20.730
abalone19	0.991	0.640	0.360	0.999	0.987	0.495	0.086	414.440	1.080	3.000	33.450
Mean	0.899	0.703	0.642	0.956	0.890	0.661	0.376	71.131	2.059	2.318	12.919

VII. OTHER PERFORMANCE METRICS FOR RUS USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.951	0.975	1.000	0.949	0.967	0.964	0.903	44.610	2.000	0.010	30.700
glass2	0.343	0.638	1.000	0.277	0.609	0.526	0.183	5.540	2.000	0.020	14.810
ecoli4	0.891	0.903	0.940	0.919	0.932	0.895	0.527	29.420	1.880	1.140	4.260
CM1	0.850	0.609	0.318	0.930	0.837	0.521	0.224	26.960	1.270	2.930	2.900
KC3	0.861	0.698	0.425	0.999	0.842	0.601	0.518	15.980	1.700	3.540	0.750
MC2	0.754	0.688	0.400	0.975	0.750	0.624	0.495	7.800	2.000	3.000	0.370
MW1	0.854	0.692	0.510	0.884	0.888	0.665	0.299	20.340	1.020	1.000	3.640
PC1	0.896	0.642	0.333	0.953	0.890	0.563	0.309	59.080	2.000	4.220	3.260
PC2	0.918	0.693	0.525	0.933	0.948	0.643	0.240	66.220	1.050	1.240	10.500
abalone9v18	0.788	0.770	0.750	0.790	0.897	0.770	0.286	55.280	3.000	1.000	25.370
abalone19	0.848	0.554	0.337	0.853	0.955	0.474	0.023	353.840	1.010	2.290	159.550
Mean	0.814	0.715	0.594	0.860	0.865	0.659	0.364	62.279	1.721	1.854	23.283

VIII. OTHER PERFORMANCE METRICS FOR RUS USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.714	0.720	0.905	0.716	0.825	0.572	0.286	33.650	1.810	0.670	21.880
glass2	0.850	0.857	0.930	0.854	0.855	0.796	0.589	17.070	1.860	0.360	4.440
ecoli4	0.843	0.665	0.570	0.872	0.881	0.483	0.233	27.900	1.140	1.250	8.370
CM1	0.879	0.612	0.728	1.000	0.796	0.504	0.158	29.000	2.910	4.000	14.630
KC3	0.800	0.524	0.795	1.000	0.667	0.363	0.045	16.000	3.180	4.000	11.950
MC2	0.630	0.622	0.624	1.000	0.624	0.567	0.268	8.000	3.120	4.960	3.040
MW1	0.933	0.598	0.745	1.000	0.897	0.344	0.269	23.000	1.490	2.000	16.420
PC1	0.902	0.684	0.642	0.949	0.901	0.641	0.384	58.820	3.850	3.750	20.690
PC2	0.973	0.517	0.095	1.000	0.951	0.111	0.009	71.000	0.190	2.000	4.400
abalone9v18	0.958	0.827	0.743	0.984	0.957	0.815	0.576	68.860	2.970	2.000	12.530
abalone19	0.978	0.736	0.580	0.985	0.984	0.709	0.128	408.790	1.740	3.000	45.060
Mean	0.860	0.669	0.669	0.942	0.849	0.537	0.268	69.281	2.205	2.545	14.855

APPENDIX H

OTHER PERFORMANCE METRICS FOR OSS

I. OTHER PERFORMANCE METRICS FOR OSS USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	0.998	46.990	2.000	0.000	0.010
glass2	0.867	0.657	0.400	0.914	0.876	0.538	0.265	18.280	0.800	1.200	1.720
ecoli4	0.947	0.737	0.500	0.975	0.946	0.698	0.512	31.190	1.000	1.000	0.810
CM1	0.828	0.487	0.038	0.937	0.786	0.073	-0.040	27.180	0.150	3.850	1.820
KC3	0.755	0.493	0.058	0.929	0.675	0.110	-0.032	14.860	0.230	3.770	1.140
MC2	0.626	0.617	0.578	0.656	0.633	0.612	0.233	5.250	2.890	2.110	2.750
MW1	0.850	0.464	0.005	0.924	0.843	0.007	-0.067	21.250	0.010	1.990	1.750
PC1	0.868	0.476	0.000	0.951	0.835	0.000	-0.066	58.990	0.000	6.000	3.010
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.970	0.720	0.440	1.000	0.964	0.654	0.645	70.000	1.760	2.240	0.000
abalone19	0.992	0.500	0.000	1.000	0.987	0.000	-0.001	414.860	0.000	3.000	0.140
Mean	0.880	0.605	0.274	0.935	0.863	0.336	0.222	70.895	0.804	2.469	1.195

II. OTHER PERFORMANCE METRICS FOR OSS USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	0.998	0.995	1.000	1.000	0.997	0.997	47.000	1.990	0.010	0.000
glass2	0.763	0.501	0.180	0.822	0.697	0.005	0.002	16.430	0.360	1.640	3.570
ecoli4	0.971	0.750	0.500	1.000	0.967	0.707	0.696	32.000	1.000	1.000	0.000
CM1	0.764	0.588	0.355	0.820	0.786	0.405	0.139	23.780	1.420	2.580	5.220
KC3	0.715	0.517	0.185	0.848	0.659	0.168	0.016	13.570	0.740	3.260	2.430
MC2	0.567	0.511	0.266	0.755	0.468	0.211	0.012	6.040	1.330	3.670	1.960
MW1	0.897	0.736	0.545	0.928	0.915	0.647	0.529	21.340	1.090	0.910	1.660
PC1	0.843	0.559	0.215	0.903	0.849	0.349	0.116	56.010	1.290	4.710	5.990
PC2	0.907	0.571	0.215	0.927	0.940	0.284	0.070	65.810	0.430	1.570	5.190
abalone9v18	0.946	0.500	0.000	1.000	0.905	0.000	0.000	70.000	0.000	4.000	0.000
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.851	0.612	0.314	0.909	0.834	0.343	0.234	69.725	0.877	2.395	2.365

III. OTHER PERFORMANCE METRICS FOR OSS USING DECISION TREE

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.997	0.977	0.955	0.998	0.997	0.973	0.958	46.920	1.910	0.090	0.080
glass2	0.869	0.667	0.420	0.914	0.880	0.529	0.299	18.280	0.840	1.160	1.720
ecoli4	0.954	0.739	0.495	0.983	0.954	0.694	0.587	31.460	0.990	1.010	0.540
CM1	0.826	0.606	0.315	0.896	0.830	0.480	0.218	25.990	1.260	2.740	3.010
KC3	0.819	0.559	0.125	0.993	0.739	0.246	0.211	15.880	0.500	3.500	0.120
MC2	0.712	0.677	0.522	0.831	0.713	0.649	0.405	6.650	2.610	2.390	1.350
MW1	0.896	0.631	0.315	0.947	0.891	0.405	0.254	21.770	0.630	1.370	1.230
PC1	0.912	0.634	0.295	0.972	0.898	0.522	0.351	60.270	1.770	4.230	1.730
PC2	0.967	0.573	0.155	0.990	0.958	0.212	0.183	70.290	0.310	1.690	0.710
abalone9v18	0.938	0.572	0.163	0.982	0.921	0.321	0.190	68.770	0.650	3.350	1.230
abalone19	0.988	0.499	0.003	0.995	0.986	0.006	-0.003	413.110	0.010	2.990	1.890
Mean	0.898	0.648	0.342	0.955	0.888	0.458	0.332	70.854	1.044	2.229	1.237

IV. OTHER PERFORMANCE METRICS FOR OSS USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.975	0.698	0.395	1.000	0.960	0.455	0.453	47.000	0.790	1.210	0.000
glass2	0.910	0.505	0.010	1.000	0.844	0.014	0.014	20.000	0.020	1.980	0.000
ecoli4	0.966	0.719	0.440	0.998	0.958	0.622	0.603	31.950	0.880	1.120	0.050
CM1	0.884	0.530	0.063	0.997	0.814	0.125	0.112	28.910	0.250	3.750	0.090
KC3	0.798	0.506	0.020	0.993	0.677	0.040	0.022	15.880	0.080	3.920	0.120
MC2	0.756	0.703	0.474	0.933	0.754	0.660	0.492	7.460	2.370	2.630	0.540
MW1	0.939	0.638	0.280	0.996	0.911	0.395	0.372	22.910	0.560	1.440	0.090
PC1	0.914	0.519	0.040	0.999	0.858	0.095	0.086	61.920	0.240	5.760	0.080
PC2	0.972	0.500	0.000	1.000	0.951	0.000	-0.001	70.970	0.000	2.000	0.030
abalone9v18	0.946	0.506	0.013	0.999	0.906	0.025	0.021	69.920	0.050	3.950	0.080
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.914	0.575	0.158	0.992	0.875	0.221	0.198	71.993	0.476	2.796	0.098

V. OTHER PERFORMANCE METRICS FOR OSS USING NEURAL NETWORKS

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.980	0.750	0.500	1.000	0.977	0.707	0.700	47.000	1.000	1.000	0.000
glass2	0.909	0.500	0.000	1.000	0.842	0.000	0.000	20.000	0.000	2.000	0.000
ecoli4	0.970	0.745	0.490	1.000	0.966	0.693	0.682	32.000	0.980	1.020	0.000
CM1	0.668	0.512	0.305	0.718	0.673	0.191	0.018	20.830	1.220	2.780	8.170
KC3	0.595	0.521	0.398	0.644	0.544	0.189	0.047	10.300	1.590	2.410	5.700
MC2	0.551	0.598	0.802	0.394	0.461	0.342	0.218	3.150	4.010	0.990	4.850
MW1	0.678	0.469	0.220	0.717	0.705	0.087	-0.034	16.500	0.440	1.560	6.500
PC1	0.824	0.509	0.125	0.892	0.809	0.124	0.017	55.310	0.750	5.250	6.690
PC2	0.912	0.505	0.075	0.936	0.934	0.085	0.004	66.440	0.150	1.850	4.560
abalone9v18	0.946	0.500	0.000	1.000	0.905	0.000	0.000	70.000	0.000	4.000	0.000
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.820	0.555	0.265	0.846	0.800	0.220	0.150	68.775	0.922	2.351	3.315

VI. OTHER PERFORMANCE METRICS FOR OSS USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.975	0.810	0.630	0.990	0.976	0.779	0.690	46.530	1.260	0.740	0.470
glass2	0.917	0.718	0.475	0.962	0.915	0.646	0.482	19.230	0.950	1.050	0.770
ecoli4	0.959	0.744	0.500	0.988	0.959	0.703	0.627	31.600	1.000	1.000	0.400
CM1	0.818	0.577	0.258	0.896	0.821	0.454	0.166	25.970	1.030	2.970	3.030
KC3	0.770	0.549	0.183	0.916	0.724	0.328	0.137	14.660	0.730	3.270	1.340
MC2	0.625	0.602	0.498	0.705	0.625	0.587	0.213	5.640	2.490	2.510	2.360
MW1	0.897	0.588	0.220	0.956	0.884	0.300	0.200	21.990	0.440	1.560	1.010
PC1	0.917	0.644	0.312	0.976	0.903	0.536	0.384	60.510	1.870	4.130	1.490
PC2	0.965	0.496	0.000	0.992	0.950	0.000	-0.010	70.460	0.000	2.000	0.540
abalone9v18	0.954	0.649	0.308	0.991	0.942	0.487	0.407	69.390	1.230	2.770	0.610
abalone19	0.992	0.499	0.000	0.999	0.987	0.000	-0.002	414.460	0.000	3.000	0.540
Mean	0.890	0.625	0.307	0.943	0.880	0.438	0.299	70.949	1.000	2.273	1.142

VII. OTHER PERFORMANCE METRICS FOR OSS USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.934	0.966	1.000	0.932	0.963	0.954	0.918	43.790	2.000	0.000	3.210
glass2	0.347	0.641	1.000	0.282	0.612	0.529	0.186	5.640	2.000	0.000	14.360
ecoli4	0.839	0.891	0.950	0.833	0.920	0.882	0.479	26.640	1.900	0.100	5.360
CM1	0.850	0.601	0.273	0.930	0.838	0.500	0.225	26.970	1.090	2.910	2.030
KC3	0.869	0.674	0.350	0.998	0.847	0.576	0.532	15.970	1.400	2.600	0.030
MC2	0.712	0.653	0.400	0.906	0.704	0.601	0.388	7.250	2.000	3.000	0.750
MW1	0.842	0.686	0.500	0.871	0.883	0.660	0.281	20.040	1.000	1.000	2.960
PC1	0.897	0.642	0.333	0.951	0.891	0.563	0.312	58.980	2.000	4.000	3.020
PC2	0.903	0.568	0.215	0.922	0.935	0.279	0.101	65.460	0.430	1.570	5.540
abalone9v18	0.792	0.765	0.735	0.795	0.898	0.764	0.284	55.680	2.940	1.060	14.320
abalone19	0.857	0.533	0.203	0.862	0.956	0.321	0.013	357.630	0.610	2.390	57.370
Mean	0.804	0.693	0.542	0.844	0.859	0.603	0.338	62.186	1.579	1.694	9.905

VIII. OTHER PERFORMANCE METRICS FOR OSS USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.664	0.787	0.920	0.654	0.763	0.671	0.435	30.720	1.840	0.160	16.280
glass2	0.849	0.827	0.800	0.854	0.851	0.744	0.563	17.080	1.600	0.400	2.920
ecoli4	0.842	0.616	0.360	0.872	0.880	0.342	0.176	27.910	0.720	1.280	4.090
CM1	0.879	0.500	0.000	1.000	0.791	0.000	0.000	29.000	0.000	4.000	0.000
KC3	0.800	0.500	0.000	1.000	0.667	0.000	0.000	16.000	0.000	4.000	0.000
MC2	0.638	0.529	0.058	1.000	0.481	0.130	0.106	8.000	0.290	4.710	0.000
MW1	0.920	0.500	0.000	1.000	0.860	0.000	0.000	23.000	0.000	2.000	0.000
PC1	0.895	0.683	0.427	0.940	0.897	0.628	0.363	58.300	2.560	3.440	3.700
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.958	0.742	0.500	0.984	0.955	0.702	0.548	68.900	2.000	2.000	1.100
abalone19	0.979	0.493	0.000	0.986	0.984	0.000	-0.010	409.250	0.000	3.000	5.750
Mean	0.854	0.607	0.279	0.936	0.825	0.292	0.198	69.015	0.819	2.454	3.076

APPENDIX I

OTHER PERFORMANCE METRICS FOR SMOTE

I. OTHER PERFORMANCE METRICS FOR SMOTE USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.990	0.995	1.000	0.989	0.992	0.995	0.907	46.490	2.000	0.000	4.300
glass2	0.895	0.699	0.500	0.943	0.894	0.660	0.357	18.860	1.000	1.170	3.060
ecoli4	0.939	0.733	0.500	0.967	0.941	0.695	0.467	30.940	1.000	1.000	1.190
CM1	0.831	0.718	0.735	0.929	0.844	0.710	0.305	26.940	2.940	3.510	8.700
KC3	0.715	0.552	0.363	0.883	0.697	0.473	0.089	14.120	1.450	3.830	4.240
MC2	0.626	0.617	0.580	0.655	0.633	0.613	0.235	5.240	2.900	2.150	2.880
MW1	0.826	0.700	0.695	0.896	0.852	0.663	0.239	20.600	1.390	1.990	7.050
PC1	0.883	0.634	0.503	0.963	0.851	0.604	0.174	59.680	3.020	5.890	14.560
PC2	0.967	0.509	0.135	0.994	0.950	0.175	0.007	70.600	0.270	2.000	8.310
abalone9v18	0.967	0.737	0.503	0.996	0.963	0.697	0.629	69.750	2.010	2.160	2.650
abalone19	0.988	0.497	0.003	0.995	0.986	0.006	-0.006	412.900	0.010	3.000	13.600
Mean	0.875	0.672	0.502	0.928	0.873	0.572	0.309	70.556	1.635	2.427	6.413

II. OTHER PERFORMANCE METRICS FOR SMOTE USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.010	0.160
glass2	0.754	0.557	0.545	0.811	0.729	0.255	0.083	16.210	1.090	1.620	8.610
ecoli4	0.971	0.750	0.500	1.000	0.967	0.707	0.696	32.000	1.000	1.000	1.000
CM1	0.758	0.625	0.523	0.818	0.782	0.436	0.165	23.730	2.090	2.700	8.220
KC3	0.720	0.589	0.443	0.848	0.678	0.352	0.175	13.570	1.770	3.290	4.220
MC2	0.573	0.519	0.284	0.768	0.469	0.228	0.033	6.140	1.420	3.810	1.970
MW1	0.915	0.777	0.670	0.947	0.929	0.728	0.579	21.780	1.340	0.900	2.930
PC1	0.844	0.686	0.602	0.899	0.858	0.600	0.272	55.730	3.610	4.310	14.190
PC2	0.894	0.584	0.275	0.913	0.937	0.350	0.077	64.840	0.550	1.680	10.560
abalone9v18	0.973	0.859	0.750	1.000	0.970	0.852	0.697	70.000	3.000	3.190	6.440
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	1.520
Mean	0.854	0.677	0.508	0.909	0.846	0.501	0.343	69.636	1.625	2.319	5.438

III. OTHER PERFORMANCE METRICS FOR SMOTE USING DECISION TREES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.988	0.880	0.845	0.999	0.987	0.869	0.836	46.940	1.690	0.850	5.380
glass2	0.882	0.730	0.545	0.928	0.898	0.628	0.412	18.560	1.090	1.220	5.510
ecoli4	0.968	0.748	0.500	0.997	0.965	0.706	0.674	31.900	1.000	1.000	0.470
CM1	0.812	0.739	0.715	0.858	0.844	0.731	0.356	24.880	2.860	2.230	7.510
KC3	0.838	0.617	0.273	0.988	0.805	0.486	0.404	15.810	1.090	3.180	0.690
MC2	0.755	0.726	0.620	0.850	0.754	0.703	0.489	6.800	3.100	2.000	1.520
MW1	0.900	0.688	0.465	0.946	0.901	0.577	0.335	21.760	0.930	1.300	2.630
PC1	0.908	0.732	0.635	0.963	0.899	0.720	0.365	59.700	3.810	4.020	10.660
PC2	0.965	0.586	0.225	0.990	0.956	0.303	0.137	70.320	0.450	1.850	3.800
abalone9v18	0.941	0.700	0.475	0.983	0.930	0.632	0.322	68.800	1.900	3.150	7.080
abalone19	0.990	0.505	0.020	0.997	0.987	0.034	0.003	413.700	0.060	3.000	15.450
Mean	0.904	0.695	0.483	0.954	0.902	0.581	0.394	70.834	1.635	2.164	5.518

IV. OTHER PERFORMANCE METRICS FOR SMOTE USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.978	0.785	0.575	1.000	0.975	0.735	0.675	47.000	1.150	1.350	0.750
glass2	0.909	0.626	0.495	0.999	0.863	0.533	0.198	19.980	0.990	1.990	5.930
ecoli4	0.968	0.748	0.505	1.000	0.964	0.706	0.671	31.990	1.010	1.090	0.650
CM1	0.884	0.698	0.605	0.995	0.855	0.684	0.331	28.860	2.420	3.690	6.420
KC3	0.795	0.552	0.178	0.994	0.721	0.293	0.116	15.900	0.710	3.990	1.310
MC2	0.801	0.757	0.568	0.958	0.800	0.727	0.585	7.660	2.840	2.320	0.550
MW1	0.943	0.763	0.610	0.993	0.936	0.738	0.558	22.840	1.220	1.340	1.910
PC1	0.915	0.662	0.530	0.999	0.877	0.640	0.216	61.910	3.180	5.660	12.760
PC2	0.973	0.506	0.025	1.000	0.952	0.035	0.013	71.000	0.050	2.000	0.900
abalone9v18	0.946	0.608	0.320	0.999	0.923	0.524	0.222	69.920	1.280	3.920	8.180
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.220
Mean	0.919	0.655	0.401	0.994	0.896	0.511	0.326	72.005	1.350	2.759	3.598

V. OTHER PERFORMANCE METRICS FOR SMOTE USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	2.000	0.000
glass2	0.909	0.519	0.645	1.000	0.842	0.442	0.033	20.000	1.290	2.000	12.130
ecoli4	0.971	0.791	0.620	1.000	0.967	0.762	0.696	32.000	1.240	1.000	1.220
CM1	0.673	0.558	0.613	0.727	0.693	0.339	0.092	21.080	2.450	2.860	14.470
KC3	0.629	0.545	0.533	0.707	0.579	0.252	0.089	11.310	2.130	2.730	7.540
MC2	0.564	0.603	0.816	0.433	0.480	0.359	0.229	3.460	4.080	1.320	5.210
MW1	0.781	0.657	0.680	0.834	0.798	0.501	0.188	19.180	1.360	1.650	9.340
PC1	0.782	0.616	0.635	0.838	0.790	0.470	0.157	51.940	3.810	4.750	24.970
PC2	0.949	0.540	0.305	0.975	0.946	0.291	0.029	69.210	0.610	1.910	15.960
abalone9v18	0.976	0.846	0.750	1.000	0.976	0.841	0.761	70.000	3.000	4.000	7.610
abalone19	0.993	0.500	0.000	1.000	0.987	0.000	0.000	415.000	0.000	3.000	0.000
Mean	0.839	0.652	0.600	0.865	0.823	0.478	0.298	69.107	1.997	2.475	8.950

VI. OTHER PERFORMANCE METRICS FOR SMOTE USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.967	0.748	0.530	0.986	0.967	0.708	0.574	46.350	1.060	0.980	2.640
glass2	0.931	0.856	0.790	0.976	0.929	0.830	0.594	19.520	1.580	1.040	1.780
ecoli4	0.967	0.748	0.500	0.997	0.964	0.706	0.671	31.900	1.000	1.020	0.370
CM1	0.834	0.736	0.638	0.903	0.859	0.722	0.384	26.200	2.550	2.680	4.960
KC3	0.811	0.600	0.275	0.954	0.771	0.477	0.284	15.260	1.100	3.100	1.190
MC2	0.672	0.637	0.482	0.791	0.662	0.604	0.291	6.330	2.410	2.650	1.730
MW1	0.894	0.646	0.355	0.954	0.893	0.446	0.285	21.950	0.710	1.620	1.890
PC1	0.925	0.679	0.432	0.982	0.911	0.620	0.435	60.910	2.590	4.070	4.520
PC2	0.963	0.531	0.090	0.990	0.950	0.125	0.047	70.260	0.180	1.950	1.930
abalone9v18	0.954	0.710	0.460	0.990	0.946	0.653	0.458	69.310	1.840	2.740	2.850
abalone19	0.991	0.576	0.163	0.998	0.987	0.281	0.112	414.340	0.490	3.000	5.060
Mean	0.901	0.679	0.429	0.957	0.895	0.561	0.376	71.121	1.410	2.259	2.629

VII. OTHER PERFORMANCE METRICS FOR SMOTE USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.422	0.699	1.000	0.397	0.713	0.615	0.225	18.670	2.000	0.000	29.300
glass2	0.365	0.651	1.000	0.302	0.633	0.548	0.194	6.030	2.000	0.000	14.650
ecoli4	0.920	0.935	0.955	0.918	0.951	0.930	0.623	29.380	1.910	0.130	3.740
CM1	0.855	0.672	0.433	0.925	0.858	0.621	0.337	26.830	1.730	2.900	2.670
KC3	0.886	0.735	0.488	0.996	0.877	0.690	0.614	15.930	1.950	3.080	0.290
MC2	0.767	0.698	0.400	0.996	0.766	0.631	0.533	7.970	2.000	3.000	0.040
MW1	0.850	0.690	0.505	0.880	0.886	0.663	0.292	20.240	1.010	1.000	3.440
PC1	0.889	0.638	0.333	0.942	0.886	0.560	0.286	58.420	2.000	4.000	4.370
PC2	0.910	0.713	0.505	0.921	0.949	0.680	0.296	65.390	1.010	1.110	12.800
abalone9v18	0.781	0.766	0.750	0.783	0.894	0.766	0.280	54.780	3.000	1.000	24.040
abalone19	0.815	0.541	0.317	0.819	0.946	0.455	0.017	339.840	0.950	2.310	150.430

VIII. OTHER PERFORMANCE METRICS FOR SMOTE USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.801	0.702	0.655	0.814	0.877	0.599	0.261	38.260	1.310	1.020	11.800
glass2	0.883	0.829	0.810	0.910	0.870	0.740	0.583	18.200	1.620	0.770	3.060
ecoli4	0.889	0.641	0.385	0.930	0.897	0.397	0.211	29.770	0.770	1.540	3.580
CM1	0.885	0.532	0.070	1.000	0.815	0.139	0.116	28.990	0.280	3.820	0.170
KC3	0.800	0.500	0.000	1.000	0.667	0.000	0.000	16.000	0.000	4.000	0.000
MC2	0.615	0.500	0.000	1.000	0.410	0.000	0.000	8.000	0.000	5.000	0.000
MW1	0.920	0.500	0.000	1.000	0.860	0.000	0.000	23.000	0.000	2.000	0.000
PC1	0.905	0.680	0.408	0.960	0.901	0.614	0.377	59.500	2.450	3.980	2.970
PC2	0.973	0.500	0.000	1.000	0.951	0.000	0.000	71.000	0.000	2.000	0.000
abalone9v18	0.963	0.856	0.750	0.982	0.965	0.849	0.660	68.710	3.000	1.890	4.690
abalone19	0.983	0.636	0.317	0.990	0.985	0.490	0.109	410.890	0.950	3.000	18.470
Mean	0.874	0.625	0.309	0.962	0.836	0.348	0.211	70.211	0.944	2.638	4.067

APPENDIX J

OTHER PERFORMANCE METRICS FOR SBC

I. OTHER PERFORMANCE METRICS FOR SBC USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.000	16.230
glass2	0.842	0.643	0.925	0.887	0.868	0.586	0.245	17.730	1.850	1.200	14.440
ecoli4	0.943	0.913	0.995	0.959	0.951	0.908	0.575	30.680	1.990	0.840	6.210
CM1	0.845	0.622	0.610	0.961	0.803	0.602	0.184	27.880	2.440	3.980	15.030
KC3	0.758	0.621	0.948	0.948	0.738	0.595	0.215	15.160	2.060	4.000	6.790
MC2	0.692	0.668	0.771	0.771	0.690	0.655	0.348	6.170	3.320	2.180	3.860
MW1	0.856	0.800	0.930	0.930	0.879	0.789	0.364	21.390	1.680	2.000	10.900
PC1	0.892	0.489	0.963	0.978	0.841	0.413	-0.025	60.660	1.850	6.000	23.070
PC2	0.972	0.761	0.855	1.000	0.951	0.733	0.191	70.990	1.710	2.000	30.260
abalone9v18	0.968	0.723	0.748	0.998	0.963	0.714	0.627	69.830	2.990	2.210	26.570
abalone19	0.914	0.732	0.990	0.920	0.970	0.712	0.079	381.880	2.970	3.000	223.570
Mean	0.880	0.725	0.885	0.941	0.878	0.701	0.346	68.125	2.260	2.492	34.266

II. OTHER PERFORMANCE METRICS FOR SBC USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.010	2.770
glass2	0.791	0.579	0.875	0.856	0.733	0.384	0.119	17.110	1.750	1.700	14.360
ecoli4	0.966	0.952	1.000	0.992	0.965	0.949	0.686	31.750	2.000	1.000	5.800
CM1	0.766	0.593	0.573	0.828	0.778	0.428	0.132	24.000	2.290	2.730	12.190
KC3	0.724	0.551	0.871	0.869	0.670	0.246	0.099	13.900	1.600	3.460	5.620
MC2	0.575	0.524	0.773	0.799	0.485	0.281	0.055	6.390	1.940	4.000	2.890
MW1	0.920	0.758	0.950	0.946	0.933	0.694	0.590	21.750	1.260	0.920	4.870
PC1	0.842	0.626	0.887	0.900	0.851	0.525	0.183	55.790	3.000	4.560	16.530
PC2	0.896	0.693	0.650	0.915	0.936	0.605	0.159	64.980	1.300	1.600	19.590
abalone9v18	0.947	0.830	0.750	1.000	0.936	0.826	0.452	70.000	3.000	3.950	24.240
abalone19	0.990	0.736	0.987	0.997	0.987	0.707	0.094	413.960	2.960	3.000	213.580
Mean	0.856	0.713	0.847	0.918	0.843	0.604	0.325	69.694	2.100	2.448	29.313

III. OTHER PERFORMANCE METRICS FOR SBC USING DECISION TREE

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.989	0.887	0.980	0.998	0.988	0.867	0.847	46.910	1.960	0.900	16.160
glass2	0.897	0.779	0.895	0.923	0.910	0.714	0.475	18.460	1.790	0.850	11.190
ecoli4	0.951	0.768	0.670	0.975	0.957	0.739	0.623	31.210	1.340	0.990	5.550
CM1	0.828	0.653	0.703	0.900	0.829	0.636	0.211	26.100	2.810	2.770	13.920
KC3	0.826	0.588	1.000	1.000	0.751	0.483	0.239	16.000	1.430	3.490	5.680
MC2	0.780	0.734	0.928	0.934	0.780	0.701	0.546	7.470	3.250	2.380	3.150
MW1	0.899	0.738	0.947	0.945	0.899	0.718	0.367	21.740	1.490	1.390	8.220
PC1	0.916	0.715	0.963	0.981	0.898	0.706	0.350	60.800	4.580	4.490	21.660
PC2	0.964	0.662	0.570	0.989	0.953	0.592	0.137	70.220	1.140	1.830	18.700
abalone9v18	0.922	0.722	0.788	0.963	0.917	0.711	0.225	67.410	3.150	3.190	24.110
abalone19	0.903	0.735	0.890	0.908	0.968	0.723	0.085	376.870	2.670	2.390	205.190
Mean	0.898	0.725	0.848	0.956	0.895	0.690	0.373	67.563	2.328	2.243	30.321

IV. OTHER PERFORMANCE METRICS FOR SBC USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.973	0.785	0.785	1.000	0.958	0.747	0.478	46.990	1.570	1.310	14.620
glass2	0.906	0.691	0.940	0.995	0.848	0.639	0.226	19.900	1.880	1.960	12.120
ecoli4	0.942	0.799	0.750	0.967	0.949	0.780	0.560	30.940	1.500	1.000	4.980
CM1	0.883	0.645	0.708	0.999	0.813	0.631	0.217	28.970	2.830	3.840	13.940
KC3	0.801	0.614	1.000	0.998	0.751	0.562	0.219	15.970	2.190	4.000	5.410
MC2	0.776	0.716	0.981	0.986	0.770	0.700	0.537	7.890	3.170	3.020	2.380
MW1	0.940	0.811	0.999	0.998	0.925	0.797	0.494	22.960	1.540	1.580	6.210
PC1	0.914	0.714	0.999	0.999	0.881	0.704	0.259	61.930	4.700	5.780	22.070
PC2	0.972	0.647	0.575	1.000	0.951	0.586	0.109	70.980	1.150	2.000	20.070
abalone9v18	0.940	0.718	0.838	0.991	0.908	0.710	0.212	69.390	3.350	3.840	29.950
abalone19	0.981	0.706	0.937	0.988	0.985	0.675	0.070	410.210	2.810	2.960	232.690
Mean	0.912	0.713	0.865	0.993	0.885	0.685	0.307	71.466	2.426	2.845	33.131

V. OTHER PERFORMANCE METRICS FOR SBC USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	1.000	0.640
glass2	0.909	0.582	0.805	1.000	0.842	0.346	0.117	20.000	1.610	2.000	13.100
ecoli4	0.965	0.953	1.000	1.000	0.965	0.950	0.679	32.000	2.000	1.990	5.290
CM1	0.718	0.506	0.553	0.789	0.719	0.258	0.010	22.890	2.210	3.180	16.880
KC3	0.685	0.533	0.802	0.799	0.606	0.203	0.070	12.790	2.710	3.220	10.200
MC2	0.586	0.609	0.814	0.518	0.519	0.397	0.249	4.140	4.070	1.790	5.410
MW1	0.781	0.574	0.837	0.750	0.798	0.324	0.104	17.240	1.560	1.720	14.750
PC1	0.785	0.542	0.802	0.841	0.775	0.272	0.061	52.150	4.350	4.740	43.590
PC2	0.935	0.518	0.450	0.960	0.942	0.228	0.027	68.160	0.900	1.880	32.270
abalone9v18	0.945	0.818	0.750	0.999	0.928	0.815	0.409	69.960	3.000	4.000	34.500
abalone19	0.993	0.691	0.990	1.000	0.987	0.645	0.080	414.980	2.770	3.000	224.450
Mean	0.846	0.666	0.800	0.878	0.826	0.494	0.255	69.210	2.471	2.593	36.462

VI. OTHER PERFORMANCE METRICS FOR SBC USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.973	0.823	0.855	0.990	0.973	0.809	0.650	46.520	1.710	0.950	15.090
glass2	0.906	0.717	0.885	0.949	0.909	0.676	0.459	18.970	1.770	1.030	11.730
ecoli4	0.922	0.804	0.760	0.947	0.937	0.786	0.472	30.300	1.520	0.950	6.080
CM1	0.834	0.680	0.735	0.921	0.820	0.672	0.245	26.700	2.940	3.170	13.110
KC3	0.814	0.600	0.969	0.967	0.761	0.513	0.260	15.470	1.730	3.220	6.060
MC2	0.659	0.619	0.793	0.799	0.643	0.582	0.254	6.390	2.600	3.110	3.300
MW1	0.900	0.749	0.946	0.960	0.882	0.736	0.326	22.090	1.590	1.600	8.230
PC1	0.924	0.738	0.960	0.984	0.908	0.731	0.413	61.010	4.940	4.200	22.590
PC2	0.953	0.720	0.710	0.979	0.947	0.676	0.163	69.540	1.420	1.980	19.170
abalone9v18	0.944	0.708	0.733	0.979	0.938	0.697	0.379	68.540	2.930	2.660	22.620
abalone19	0.928	0.735	0.933	0.933	0.974	0.722	0.084	387.150	2.800	2.410	207.220
Mean	0.887	0.717	0.843	0.946	0.881	0.691	0.337	68.425	2.359	2.298	30.473

VII. OTHER PERFORMANCE METRICS FOR SBC USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.948	0.973	1.000	0.945	0.978	0.970	0.838	44.430	2.000	0.000	36.070
glass2	0.325	0.629	1.000	0.258	0.589	0.507	0.175	5.150	2.000	0.000	14.990
ecoli4	0.921	0.789	0.665	0.949	0.932	0.757	0.475	30.360	1.330	1.210	3.190
CM1	0.851	0.652	0.963	0.931	0.837	0.582	0.308	26.990	3.850	2.920	27.310
KC3	0.892	0.746	0.999	0.999	0.887	0.704	0.644	15.980	2.180	3.920	5.840
MC2	0.769	0.700	0.998	1.000	0.769	0.632	0.539	8.000	2.000	3.000	0.960
MW1	0.861	0.724	0.886	0.893	0.892	0.701	0.326	20.530	1.160	1.000	3.550
PC1	0.907	0.642	0.964	0.970	0.895	0.563	0.328	60.120	2.000	4.880	4.630
PC2	0.885	0.605	0.690	0.908	0.918	0.427	0.112	64.440	1.380	1.860	39.450
abalone9v18	0.754	0.752	0.928	0.755	0.885	0.752	0.257	52.830	3.710	1.000	39.020
abalone19	0.537	0.712	0.963	0.538	0.842	0.669	0.072	223.380	2.890	1.990	224.390
Mean	0.786	0.720	0.914	0.831	0.857	0.660	0.370	50.201	2.227	1.980	36.309

VIII. OTHER PERFORMANCE METRICS FOR SBC USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.704	0.752	0.930	0.714	0.810	0.624	0.337	33.560	1.860	1.070	21.210
glass2	0.848	0.897	0.975	0.847	0.858	0.850	0.613	16.940	1.950	0.280	4.320
ecoli4	0.782	0.706	0.665	0.795	0.854	0.562	0.301	25.430	1.330	0.900	10.590
CM1	0.880	0.564	0.968	1.000	0.808	0.408	0.094	29.000	3.870	4.000	27.450
KC3	0.800	0.521	1.000	1.000	0.667	0.338	0.046	16.000	2.950	4.000	11.120
MC2	0.695	0.609	1.000	1.000	0.613	0.373	0.299	8.000	3.970	5.000	6.390
MW1	0.930	0.567	1.000	1.000	0.886	0.339	0.185	23.000	1.030	2.000	10.240
PC1	0.893	0.685	0.925	0.940	0.896	0.666	0.359	58.260	3.870	3.710	17.960
PC2	0.973	0.598	0.460	1.000	0.951	0.430	0.082	71.000	0.920	2.000	19.280
abalone9v18	0.950	0.812	0.760	0.973	0.950	0.809	0.512	68.140	3.040	1.870	18.990
abalone19	0.866	0.777	0.983	0.869	0.960	0.764	0.099	360.750	2.950	1.680	217.280
Mean	0.847	0.681	0.879	0.922	0.841	0.560	0.266	64.553	2.522	2.410	33.166

APPENDIX K

OTHER PERFORMANCE METRICS FOR CLUS

I. OTHER PERFORMANCE METRICS FOR CLUS USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.5053061	0.7421277	1	0.484255	0.775407	0.685517	0.219152	22.76	2	0	24.61
glass2	0.3281818	0.608	0.95	0.266	0.547712	0.464781	0.165799	5.32	1.9	0.1	14.68
ecoli4	0.8835294	0.7459375	0.63	0.901875	0.927082	0.706651	0.528454	28.86	1.26	0.82	4.61
CM1	0.2581818	0.4346121	0.6675	0.201724	0.249471	0.183484	-0.1762	5.85	2.67	1.33	23.15
KC3	0.705	0.5090625	0.205	0.835625	0.583123	0.05789	0.028654	13.37	0.82	3.27	3.15
MC2	0.7	0.6445	0.564	0.885	0.692778	0.571395	0.3953	7.08	2.82	2.98	5.33
MW1	0.718	0.6184783	0.5	0.736957	0.788541	0.587055	0.175552	16.95	1	1	6.05
PC1	0.2376471	0.5051613	0.83	0.180323	0.209685	0.106462	0.024847	11.18	4.98	1.02	50.84
PC2	0.2056164	0.4822887	0.775	0.189577	0.24589	0.11485	-0.01689	13.46	1.55	0.45	57.57
abalone9v18	0.6943243	0.5731071	0.5625	0.720714	0.839573	0.469474	0.067295	50.45	2.25	3.07	29.14
abalone19	0.7949043	0.5756305	0.35333333	0.79812	0.936705	0.523196	0.03562	331.22	1.06	1.95	83.86

II. OTHER PERFORMANCE METRICS FOR CLUS USING SVM

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.6532653	0.8001064	0.96	0.640213	0.853769	0.772189	0.294778	30.09	1.92	0.1	19.34
glass2	0.3018182	0.5935	0.95	0.238	0.524152	0.444901	0.140856	4.76	1.9	0.15	15.26
ecoli4	0.8832353	0.7551563	0.63	0.900313	0.931317	0.715482	0.579929	28.81	1.26	0.78	4.19
CM1	0.2645455	0.4737931	0.75	0.198276	0.221827	0.097719	-0.06296	5.75	3	1.03	23.27
KC3	0.719	0.543125	0.25	0.83625	0.635317	0.207678	0.117335	13.38	1	3.01	3.3
MC2	0.6546154	0.58325	0.54	0.8925	0.620753	0.454834	0.291004	7.14	2.7	3.63	5.76
MW1	0.6968	0.5997826	0.49	0.717826	0.78785	0.567227	0.134964	16.51	0.98	1.09	6.68
PC1	0.2361765	0.5124194	0.855	0.178387	0.203123	0.134297	0.038343	11.06	5.13	1	51.47
PC2	0.2134247	0.4255634	0.65	0.201127	0.320174	0.198552	-0.08409	14.28	1.3	0.7	56.75
abalone9v18	0.6927027	0.5913929	0.5625	0.719	0.840268	0.493283	0.08453	50.33	2.25	3.07	26.58
abalone19	0.7993301	0.581253	0.36	0.802506	0.938506	0.527386	0.037645	333.04	1.08	1.92	82.05

III. OTHER PERFORMANCE METRICS FOR CLUS USING DECISION TREES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.3479592	0.5763298	0.84	0.32766	0.585031	0.421417	0.079598	15.4	1.68	0.35	32.75
glass2	0.3272727	0.60075	0.935	0.2665	0.55148	0.449206	0.150562	5.33	1.87	0.14	14.69
ecoli4	0.8891176	0.7359375	0.605	0.917188	0.916579	0.690588	0.539883	29.35	1.21	1.12	4.26
CM1	0.2542424	0.4377586	0.68	0.196207	0.230523	0.157754	-0.18501	5.69	2.72	1.3	23.33
KC3	0.704	0.5103125	0.21	0.8375	0.575957	0.069745	0.039808	13.4	0.84	3.32	3.03
MC2	0.6707692	0.608	0.566	0.88	0.646329	0.499209	0.323377	7.04	2.83	3.32	5.31
MW1	0.762	0.6492391	0.52	0.783478	0.820619	0.614399	0.325575	18.02	1.04	0.97	5.17
PC1	0.2407353	0.5196505	0.85833333	0.180968	0.217512	0.16214	0.071165	11.22	5.15	0.98	50.78
PC2	0.2143836	0.3604577	0.515	0.205915	0.305146	0.153902	-0.17634	14.62	1.03	0.97	56.45
abalone9v18	0.7182432	0.6066429	0.58	0.745857	0.849257	0.501615	0.097717	52.21	2.32	3.06	25.67
abalone19	0.8034211	0.5833133	0.36	0.806627	0.93951	0.528236	0.038594	334.75	1.08	1.92	80.51
Mean	0.539	0.563	0.606	0.559	0.603	0.386	0.119	46.094	1.979	1.586	27.450

IV. OTHER PERFORMANCE METRICS FOR CLUS USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.6359184	0.6737766	0.75	0.632553	0.826918	0.621762	0.168537	29.73	1.5	0.57	20.36
glass2	0.3881818	0.629	0.925	0.336	0.62674	0.505377	0.173043	6.72	1.85	0.18	13.34
ecoli4	0.8976471	0.7221875	0.545	0.930625	0.920916	0.621564	0.494809	29.78	1.09	1.26	3.22
CM1	0.2824242	0.4457328	0.6625	0.230345	0.332456	0.256357	-0.12018	6.68	2.65	1.36	22.36
KC3	0.7035	0.500625	0.18	0.83875	0.575398	0.036286	0.00516	13.42	0.72	3.35	3.02
MC2	0.6784615	0.6135	0.556	0.895	0.652701	0.500012	0.338678	7.16	2.78	3.34	5.12
MW1	0.7628	0.6394565	0.495	0.786522	0.821326	0.6069	0.266798	18.09	0.99	1.02	4.97
PC1	0.2454412	0.5109409	0.83333333	0.188548	0.234167	0.140867	0.045522	11.69	5	1.02	50.35
PC2	0.2476712	0.3314085	0.42	0.242817	0.405941	0.170729	-0.19451	17.24	0.84	1.16	53.87
abalone9v18	0.7256757	0.5896786	0.5375	0.754286	0.851598	0.484754	0.081181	52.8	2.15	3.1	25.07
abalone19	0.8211722	0.5830803	0.34666667	0.824699	0.945245	0.527661	0.041077	342.25	1.04	2	74.91
Mean	0.581	0.567	0.568	0.605	0.654	0.407	0.118	48.687	1.874	1.669	25.145

V. OTHER PERFORMANCE METRICS FOR CLUS USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.7455102	0.8505851	0.965	0.73617	0.8955	0.834521	0.388324	34.6	1.93	0.07	14.52
glass2	0.5213636	0.5605	0.675	0.514	0.638605	0.296607	0.091432	10.28	1.35	0.81	11.08
ecoli4	0.8967647	0.7435938	0.6	0.917188	0.934246	0.681197	0.561425	29.35	1.2	0.86	4.01
CM1	0.6193939	0.4838793	0.33	0.662759	0.604709	0.126843	-0.02535	19.22	1.32	2.78	10.81
KC3	0.4175	0.5103125	0.7675	0.356875	0.330238	0.169277	0.019798	5.71	3.07	1.36	11.95
MC2	0.46	0.53575	0.864	0.48625	0.376042	0.217814	0.085569	3.89	4.32	3.05	6.34
MW1	0.5852	0.578913	0.6	0.586522	0.639187	0.42529	0.109392	13.49	1.2	0.86	10.17
PC1	0.4861765	0.5097312	0.56333333	0.481129	0.491684	0.14175	0.028809	29.83	3.38	2.77	34.69
PC2	0.5905479	0.4347887	0.295	0.599577	0.74431	0.195335	-0.05643	42.57	0.59	1.46	30.24
abalone9v18	0.6918919	0.5892857	0.55	0.718714	0.839435	0.484982	0.083057	50.31	2.2	3.11	26
abalone19	0.7982057	0.5806867	0.36	0.801373	0.93834	0.526853	0.036843	332.57	1.08	1.92	84.68
Mean	0.619	0.580	0.597	0.624	0.676	0.373	0.120	51.984	1.967	1.732	22.226

VI. OTHER PERFORMANCE METRICS FOR CLUS USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.3559184	0.5613298	0.79	0.33766	0.605151	0.413797	0.066555	15.87	1.58	0.43	31.64
glass2	0.3277273	0.5985	0.93	0.268	0.547848	0.440002	0.149074	5.36	1.86	0.15	14.66
ecoli4	0.8814706	0.7340625	0.605	0.905	0.921286	0.688697	0.546377	28.96	1.21	0.99	4.38
CM1	0.2542424	0.436681	0.6775	0.196207	0.230523	0.157754	-0.18796	5.69	2.71	1.3	23.32
KC3	0.706	0.5084375	0.2125	0.839375	0.580981	0.077289	0.022981	13.43	0.85	3.31	3.13
MC2	0.6569231	0.59075	0.56	0.8775	0.626585	0.471084	0.291067	7.02	2.8	3.48	5.31
MW1	0.7604	0.6483696	0.515	0.781739	0.819139	0.613526	0.320865	17.98	1.03	0.97	5.04
PC1	0.24	0.52	0.86	0.180323	0.216298	0.161641	0.067394	11.18	5.16	0.96	50.84
PC2	0.2121918	0.3639085	0.525	0.203803	0.299209	0.123664	-0.18282	14.47	1.05	0.98	56.6
abalone9v18	0.7204054	0.6128214	0.5825	0.748143	0.850145	0.508006	0.103525	52.37	2.33	3.06	24.98
abalone19	0.802823	0.583012	0.36	0.806024	0.939194	0.527993	0.038523	334.5	1.08	1.92	81.02
Mean	0.538	0.560	0.602	0.559	0.603	0.380	0.112	46.075	1.969	1.595	27.356

VII. OTHER PERFORMANCE METRICS FOR CLUS USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.7269388	0.8576596	1	0.715319	0.890124	0.840353	0.386786	33.62	2	0	13.9
glass2	0.3681818	0.6225	0.935	0.312	0.605828	0.488514	0.169226	6.24	1.87	0.14	13.8
ecoli4	0.8776471	0.8664063	0.93	0.903438	0.905716	0.849413	0.476231	28.91	1.86	1.07	6.31
CM1	0.2906061	0.4498276	0.66	0.239655	0.378217	0.298991	-0.10742	6.95	2.64	1.36	22.08
KC3	0.7065	0.519375	0.23	0.83125	0.589338	0.101572	0.045457	13.3	0.92	3.17	3.12
MC2	0.7023077	0.649375	0.56	0.87875	0.700042	0.590162	0.399394	7.03	2.8	2.9	5.18
MW1	0.726	0.6228261	0.5	0.745652	0.804009	0.595161	0.186244	17.15	1	1	5.88
PC1	0.2467647	0.5221237	0.85666667	0.189355	0.265616	0.239567	0.054939	11.74	5.14	0.96	50.37
PC2	0.240274	0.3446127	0.455	0.234225	0.412518	0.227863	-0.16583	16.63	0.91	1.09	54.38
abalone9v18	0.6994595	0.5747857	0.565	0.726143	0.841133	0.473537	0.068659	50.83	2.26	3.07	29.08
abalone19	0.7869378	0.575012	0.36	0.790024	0.935632	0.523912	0.033232	327.86	1.08	1.92	91.5
Mean	0.579	0.600	0.641	0.597	0.666	0.475	0.141	47.296	2.044	1.516	26.873

VIII. OTHER PERFORMANCE METRICS FOR CLUS USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.422449	0.6676064	0.935	0.401277	0.545732	0.45503	0.216972	18.86	1.87	0.16	28.19
glass2	0.7927273	0.82975	0.875	0.7845	0.831939	0.725905	0.453453	15.69	1.75	0.26	4.34
ecoli4	0.8432353	0.6001563	0.375	0.88375	0.849348	0.315059	0.13968	28.28	0.75	1.61	5.59
CM1	0.2818182	0.4674569	0.7125	0.222414	0.238545	0.07162	-0.06899	6.45	2.85	1.15	22.55
KC3	0.727	0.484375	0.09	0.88875	0.627394	0.049494	-0.02845	14.22	0.36	3.68	2
MC2	0.6138462	0.502875	0.15	0.98375	0.458039	0.190055	0.005717	7.87	0.75	4.89	1.32
MW1	0.92	0.5	0	1	0.860163	0	0	23	0	2	0
PC1	0.2277941	0.5206183	0.87833333	0.168548	0.159624	0.078635	0.039041	10.45	5.27	0.96	51.9
PC2	0.9164384	0.51	0.08	0.94	0.938233	0.102365	0.009335	66.74	0.16	1.84	4.26
abalone9v18	0.8744595	0.6214286	0.555	0.911571	0.890786	0.487838	0.123393	63.81	2.22	3.1	21.85
abalone19	0.946244	0.4851687	0.02	0.952964	0.976267	0.030336	-0.0108	395.48	0.06	2.95	20.61
Mean	0.688	0.563	0.425	0.740	0.671	0.228	0.080	59.168	1.458	2.055	14.783

APPENDIX L

OTHER PERFORMANCE METRICS FOR CUST

I. OTHER PERFORMANCE METRICS FOR CUST USING KNN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.993	0.986	1.000	1.000	0.993	0.985	0.906	46.980	2.000	0.300	12.32
glass2	0.902	0.676	0.600	0.953	0.897	0.599	0.372	19.050	1.200	1.200	8.75
ecoli4	0.957	0.853	0.815	0.986	0.956	0.838	0.594	31.550	1.630	1.000	3.85
CM1	0.849	0.617	0.956	0.966	0.810	0.589	0.186	28.000	2.310	3.990	10.85
KC3	0.795	0.554	0.974	0.994	0.709	0.485	0.097	15.910	1.670	4.000	5.44
MC2	0.736	0.664	0.973	0.978	0.720	0.625	0.452	7.820	2.960	3.250	3.38
MW1	0.913	0.712	0.946	0.992	0.869	0.671	0.260	22.820	1.540	2.000	9.47
PC1	0.903	0.495	0.959	0.990	0.844	0.375	-0.020	61.380	1.560	6.000	20.82
PC2	0.972	0.770	0.805	0.999	0.951	0.739	0.211	70.940	1.610	2.000	26.79
abalone9v18	0.966	0.737	0.998	1.000	0.959	0.717	0.587	70.000	2.520	3.570	16.3
abalone19	0.979	0.685	0.947	0.986	0.984	0.654	0.072	409.140	2.030	3.000	136.69
Mean	0.906	0.704	0.907	0.986	0.881	0.662	0.338	71.235	1.912	2.755	23.151

II. OTHER PERFORMANCE METRICS FOR CUST USING SVN

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	0.200	0.54
glass2	0.844	0.554	0.595	0.920	0.775	0.264	0.080	18.400	1.190	1.840	9.75
ecoli4	0.971	0.952	0.980	1.000	0.967	0.949	0.696	32.000	1.960	1.000	3.67
CM1	0.818	0.589	0.865	0.893	0.808	0.404	0.142	25.890	2.150	2.910	10.45
KC3	0.759	0.547	0.914	0.936	0.664	0.218	0.085	14.980	1.510	3.800	4.72
MC2	0.585	0.546	0.846	0.910	0.511	0.307	0.097	7.280	1.880	4.880	2.27
MW1	0.936	0.759	0.927	0.976	0.936	0.692	0.615	22.450	1.390	1.040	6.74
PC1	0.867	0.628	0.879	0.935	0.856	0.523	0.182	57.940	3.120	4.960	16.93
PC2	0.894	0.667	0.620	0.914	0.936	0.549	0.155	64.920	1.240	1.630	20.77
abalone9v18	0.973	0.837	1.000	1.000	0.970	0.833	0.697	70.000	3.000	4.000	10.59
abalone19	0.993	0.667	0.997	1.000	0.987	0.648	0.141	415.000	1.830	3.000	114.28
Mean	0.876	0.704	0.875	0.953	0.856	0.581	0.354	70.533	1.934	2.660	18.246

III. OTHER PERFORMANCE METRICS FOR CUST USING DECISION TREE

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.987	0.938	1.000	0.995	0.988	0.934	0.841	46.770	2.000	0.650	14.95
glass2	0.892	0.729	0.780	0.950	0.877	0.710	0.318	19.000	1.560	1.370	8.06
ecoli4	0.963	0.746	0.565	0.992	0.961	0.708	0.639	31.730	1.130	1.000	3.71
CM1	0.845	0.661	0.932	0.948	0.827	0.645	0.217	27.500	2.820	3.670	11.41
KC3	0.817	0.576	0.998	1.000	0.749	0.499	0.221	16.000	1.490	3.920	4.5
MC2	0.795	0.749	0.988	0.990	0.795	0.713	0.575	7.920	3.280	3.410	3.26
MW1	0.915	0.748	0.950	0.957	0.910	0.718	0.387	22.010	1.470	1.180	7.02
PC1	0.911	0.710	0.955	0.975	0.895	0.702	0.343	60.470	4.400	4.510	21.11
PC2	0.960	0.646	0.545	0.983	0.955	0.573	0.144	69.790	1.090	1.700	17.99
abalone9v18	0.947	0.707	0.957	0.994	0.920	0.686	0.239	69.610	2.570	3.560	18.42
abalone19	0.969	0.709	0.931	0.975	0.983	0.688	0.078	404.690	2.130	2.580	137.7
Mean	0.909	0.720	0.873	0.978	0.896	0.689	0.364	70.499	2.176	2.505	22.557

IV. OTHER PERFORMANCE METRICS FOR CUST USING RANDOM FOREST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.976	0.812	0.765	1.000	0.965	0.776	0.573	47.000	1.530	1.170	12.48
glass2	0.909	0.683	0.750	1.000	0.862	0.653	0.228	20.000	1.500	2.000	8.16
ecoli4	0.967	0.784	0.675	1.000	0.959	0.761	0.620	32.000	1.350	1.110	3.42
CM1	0.879	0.658	0.998	1.000	0.843	0.637	0.254	28.990	2.450	3.990	10.23
KC3	0.800	0.620	0.999	1.000	0.750	0.565	0.230	16.000	1.880	4.000	3.75
MC2	0.764	0.718	0.999	1.000	0.759	0.697	0.502	8.000	3.330	4.480	1.96
MW1	0.939	0.767	0.998	1.000	0.931	0.740	0.520	23.000	1.420	1.970	4.79
PC1	0.915	0.688	0.996	1.000	0.884	0.679	0.272	61.990	4.180	5.900	19.89
PC2	0.971	0.696	0.655	0.998	0.951	0.658	0.148	70.840	1.310	1.990	18.64
abalone9v18	0.946	0.707	0.991	1.000	0.914	0.696	0.217	70.000	2.610	4.000	19.76
abalone19	0.993	0.684	0.995	1.000	0.987	0.666	0.065	414.920	2.150	3.000	157.68
Mean	0.914	0.711	0.893	1.000	0.891	0.684	0.330	72.067	2.155	3.055	23.705

V. OTHER PERFORMANCE METRICS FOR CUST USING NEURAL NETWORK

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	1.000	1.000	1.000	1.000	1.000	1.000	1.000	47.000	2.000	2.000	0
glass2	0.909	0.533	0.495	1.000	0.842	0.309	0.043	20.000	0.990	2.000	8.57
ecoli4	0.971	0.946	0.960	1.000	0.967	0.943	0.696	32.000	1.920	2.000	3.22
CM1	0.852	0.509	0.814	0.963	0.780	0.198	0.017	27.930	2.650	3.830	19.92
KC3	0.786	0.537	0.889	0.974	0.668	0.231	0.072	15.590	2.570	3.860	9.42
MC2	0.662	0.599	0.891	0.985	0.547	0.360	0.222	7.880	4.010	4.360	5.17
MW1	0.912	0.568	0.846	0.989	0.859	0.315	0.088	22.750	1.510	1.960	14.22
PC1	0.873	0.527	0.860	0.949	0.840	0.213	0.037	58.810	4.390	5.470	43.65
PC2	0.910	0.544	0.510	0.933	0.927	0.250	0.046	66.240	1.020	1.840	31.14
abalone9v18	0.970	0.826	1.000	1.000	0.967	0.821	0.670	70.000	3.000	4.000	16.63
abalone19	0.993	0.624	1.000	1.000	0.987	0.545	0.085	415.000	1.000	3.000	153.95
Mean	0.894	0.656	0.842	0.981	0.853	0.471	0.271	71.200	2.278	3.120	27.808

VI. OTHER PERFORMANCE METRICS FOR CUST USING ADABOOST

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.969	0.884	0.920	0.987	0.969	0.871	0.609	46.400	1.840	0.900	12.94
glass2	0.936	0.731	0.785	0.982	0.928	0.707	0.560	19.630	1.570	1.040	7.85
ecoli4	0.967	0.756	0.650	0.996	0.964	0.728	0.670	31.880	1.300	1.000	4.95
CM1	0.835	0.673	0.908	0.932	0.820	0.662	0.246	27.030	2.740	3.490	10.87
KC3	0.814	0.593	0.971	0.989	0.753	0.528	0.237	15.820	1.670	3.530	4.81
MC2	0.645	0.611	0.820	0.843	0.632	0.591	0.229	6.740	2.760	3.700	3.24
MW1	0.928	0.731	0.933	0.983	0.905	0.713	0.335	22.620	1.580	1.490	7.56
PC1	0.922	0.722	0.957	0.986	0.902	0.715	0.384	61.110	4.550	4.430	21.2
PC2	0.945	0.740	0.735	0.971	0.946	0.702	0.183	68.940	1.470	1.970	18.05
abalone9v18	0.955	0.715	0.967	0.997	0.939	0.690	0.382	69.810	2.600	3.170	19
abalone19	0.979	0.742	0.948	0.985	0.985	0.724	0.102	408.880	2.290	2.850	138.99
Mean	0.899	0.718	0.872	0.968	0.886	0.694	0.358	70.805	2.215	2.506	22.678

VII. OTHER PERFORMANCE METRICS FOR CUST USING NAÏVE BAYES

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.989	0.994	1.000	0.988	0.994	0.994	0.934	46.450	2.000	0.000	28.18
glass2	0.365	0.651	1.000	0.302	0.633	0.548	0.194	6.030	2.000	0.040	14.98
ecoli4	0.908	0.891	0.880	0.941	0.943	0.878	0.585	30.100	1.760	1.230	3.19
CM1	0.867	0.647	0.938	0.950	0.848	0.580	0.271	27.540	1.970	2.930	5.99
KC3	0.871	0.689	0.999	1.000	0.843	0.570	0.518	16.000	1.550	3.660	0.56
MC2	0.769	0.700	0.999	1.000	0.769	0.632	0.539	8.000	2.000	3.000	0.3
MW1	0.871	0.702	0.876	0.903	0.895	0.672	0.328	20.770	1.000	1.000	3.31
PC1	0.905	0.643	0.955	0.963	0.894	0.561	0.323	59.730	1.980	4.250	2.77
PC2	0.902	0.678	0.660	0.915	0.943	0.572	0.235	64.990	1.320	1.320	31.46
abalone9v18	0.838	0.767	0.785	0.866	0.898	0.767	0.282	60.640	3.000	2.610	24.3
abalone19	0.700	0.579	0.647	0.703	0.911	0.558	0.027	291.760	1.940	2.000	202.47
Mean	0.817	0.722	0.885	0.866	0.870	0.667	0.385	57.455	1.865	2.004	28.865

VIII. OTHER PERFORMANCE METRICS FOR CUST USING QDA

Dataset	Accuracy	AUC	Sensitivity	Specificity	F-measure	G-Mean	MCC	TN	TP	FN	FP
yeast-2_vs_8	0.722	0.800	0.980	0.725	0.820	0.710	0.426	34.090	1.960	0.840	23.66
glass2	0.851	0.883	0.930	0.857	0.859	0.828	0.615	17.130	1.860	0.510	4.09
ecoli4	0.806	0.691	0.635	0.831	0.846	0.522	0.259	26.600	1.270	1.190	9.36
CM1	0.879	0.575	1.000	1.000	0.805	0.387	0.116	29.000	3.260	4.000	19.93
KC3	0.800	0.506	1.000	1.000	0.668	0.251	0.021	16.000	3.680	4.000	14.9
MC2	0.685	0.599	1.000	1.000	0.589	0.387	0.263	8.000	5.000	5.000	8
MW1	0.927	0.556	1.000	1.000	0.881	0.204	0.154	23.000	1.710	2.000	18.84
PC1	0.896	0.685	0.941	0.938	0.898	0.669	0.370	58.140	4.110	3.680	19.78
PC2	0.973	0.589	0.530	1.000	0.951	0.357	0.071	71.000	1.060	2.000	27.24
abalone9v18	0.959	0.828	0.975	0.986	0.956	0.820	0.556	68.990	2.990	2.000	11.57
abalone19	0.962	0.734	0.931	0.968	0.981	0.728	0.125	401.710	2.380	2.690	151.06
Mean	0.860	0.677	0.902	0.937	0.841	0.533	0.270	68.515	2.662	2.537	28.039