"Software Defect Prediction using Class Imbalance Learning"

A PROJECT REPORT

SUBMITTED IN THE PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

Submitted By

Bhupender Rana

(2K19/SWE/18)

Under the supervision of

Dr. Ruchika Malhotra
Head of Department
Department of Software Engineering
Delhi Technological University, Delhi

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

JUNE, 2021

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Bhupender Rana, 2K19/SWE/18 student of M.Tech (SWE), hereby declare that the project entitled **"Software Defect Prediction Using Class Imbalance Learning"** which is submitted by me to Department of Software Engineering, Delhi Technological University, Shahbad Daulatpur, Delhi in partial fulfilment of requirement for the award of the degree of Master of Technology in Software Engineering, has not been previously formed the basis for any fulfilment of requirement in any degree or other similar title or recognition.

This report is an authentic record of my work carried out during my degree under the guidance of Dr. Ruchika Malhotra.

Place: Delhi **Bhupender Rana**

Date: 7th June, 2021 **(2K19/SWE/18)**

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

This is to certify that Bhupender Rana (2K19/SWE/18) has completed the project titled "Software Defect Prediction Using Class Imbalance Learning" under my supervision in partial fulfillment of the MASTER OF TECHNOLOGY degree in Software Engineering at DELHI TECHNOLOGICAL UNIVERSITY.

Place: Delhi

Date:

Dr. Ruchika Malhotra

(SUPERVISOR)

(Head Of Department)

(Associate Dean IRD, DTU)

ACKNOWLEDGEMENT

The success of a Major Project II required help and contribution from numerous individuals and the organization. Writing the report of this project work gives me an opportunìty to express my gratìtude to everyone who has helped in shaping up the outcome of the project.

I express my heartfelt gratitude to my project guide **Dr. Ruchika Malhotra** for giving me an opportunity to do my major project II work under her guidance. Her constant support and encouragement has made me realize that it is the process of learning which weighs more than the end result. I am highly indebted to the panel faculties during all the progress evaluations for their guidance, constant supervision and for motivating me to complete my work. They helped me throughout by giving new ideas, providing necessary information and pushing me forward to complete the work.

Bhupender Rana

(2K19/SWE/18)

ABSTRACT

Software testing is important part of the software development. Testing requires maximum resources and effort. Defective modules in the software can risk the development and maintenance costs. It is important to detect the defect ìn the starting stages of applìcation development lìfe cycle. Defect prone modules should be identified and quality assurance activities are enforced. Earlier Predicting of software faults improves the efficiency, reliability, software qualìty and reduces the cost of software.

To facilitate software testing and reduce testing costs, various machine learning approaches are explored to predict faults in software modules. Class imbalance learning specializes in solving classification problems of "balanced distributions", which can be useful in predicting defects, but have not yet been widely studied.

In thìs article, I have outlined whether and how learning methods for class imbalance will benefit from software defect predìctìon in order to find better solutions.

Software defect prediction is defined as the process of finding faulty modules in software using certain historical data and software metrics to improve software quality, and this software defect prediction process reduces the number of modules to be tested.

In software development, the prediction of software faults can be framed as a learning task that is gaining "increasing attention" both in academia and in industry. The characteristics of the static code are taken from previous versions of the product with error logs and are used to create models to predict bad modules for the next version.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
NB	Naïve Bayes
RF	Randòm Fòrest
DT	Decìsìon Tree
SVM	Suppòrt Vectòr Machìne
RUS	Randòm under Samplìng
RUS-Bal	Balanced Randòm under Samplìng
THM	Threshòld Mòving
SMB	Smòte Boòst
RUB	Randòm under Sampling Bòost
ADASYN	Adaptive Synthetic

CHAPTER-1
INTRODUCTION

1.1 Overvìew and Motìvation

If faults and failures occur in the software, it will result in increase of cost. This increases the project budget in significant amount. Methods are needed to reduce this cost and ensure software Quality. Cost can be reduced by improving and including the steps of early defect identification. Thus, defect predicting is used for the process improvement. Hence this also cost reduction.

Software Defect Prediction in software engineering can be conceived as a learning problem. Statìc code attrìbutes are derìved from previìous product updates wìth log fìles for bugs and used to create templates for the next version to predict faulty components. This effort is especially helpful when the budget of the project is small, or the entire software framework is too complex to be thoroughly checked. In order to concentrate the testìng on defect-prone components of the program, a good defect predìctor will direct software engìneers.

The motivation for choosing this project is that early actions can prevent large defects in the future and to perform those early actions machine learning can be utilized that reduces the cost of defect prediction by significant margin.

1.2 Research Objective

The first and the foremost objective is making efforts to enhance software testìng and save costs for testìng. Software testing aims to finalise the programme or product of software against the specifications of company and consumers.

I am using Naìve Bayes classìfier and Randòm Fòrest Classìfier, SVM, Decìsion Tree techniques in combination with 10-fold cross valìdation technique to predict the defects in software. Apart from these techniques, I am utilizing class balancing techniques in combination with these techniques to improve the training of the models.

In this project, I am analyzing performance of Naìve Bayes classifìer and Randòm Fòrest Classìfier, SVM Classifier, Decìsion Tree Classìfier on imbalanced dataset and performance of models when skewness of dataset is removed using class balancing techniques such as Random under Sampling, etc.

CHAPTER-2
LITERATURE STUDY

Learnìng class imbalance refers to learnìng from data sets that indicate a substantial imbalance between classes or withìn classes. The literature's common definition of "ìmbalance" is concerned wìth the sìtuation in whìch some data groups are strongly under-represented relative to other classes. By conventìon, we call the classes the majorìty classes with more examples, and the minority classes with less examples.

For SDP, the defèct case ìs much less lìkely to occur than the non-defèct case due to the nature of the issue. The defect class is thus the minority. It is more necessary to consider this class, since failure to find a defect may significantly degrade the quality of software.

Several approaches have been proposed to solve class imbalance problems at the level of data and algorithm layers. A variety of resampling technologies include data layer approaches, data transfer processing to correct for distorted class transformations such as random oversampling, random subsampling, and SMOTEi. They are simple and effective, but their effectiveness largely depends on the task and the learning algorithms. Algorithm-level techniques remove class imbalance by explicitly modifying the transmission process to improve the accuracy of the minority class.

It is predicted that defect predictors can help to enhance the qualìty of software and reduce the cost of providing such software systems. After the PROMISE repository was established in 2005, there was a rapid growth in SDP research. It provides a compilation of fault prediction data sets for public use from real-world projects, and enables researchers to construct repeatable, comparable models across studìes.

The static code metrics described by McCabe [1] and Halstead [2] are widely used and generally accepted to define module attributes, which is usually the smallest unit of functionality. Using the flow graph, McCabe metrics collect information about the complexity of the paths found in the m-module.

CHAPTER-3
IMPLEMENTATION

Class Imbalance Learning Techniques

The four class ìmbalance learnìng methods are randòm under sampling (RUS), balanced randòm under sampling (RUS-bal, alsò called mìcro-samplìng), threshold-movìng (THM), SMOTE Boost (SMB).

3.1 Random Undersampling

In Random, the instances are randomly selected from the majority class and removed from the training dataset under sampling techniques. Basically, data samples of the majority class are reduced to equate to minority class samples in this process.

In order to find an optimal class distribution, such as an equivalent number of samples for each class, this method can be repeated many times.

3.2 RUS-Bal

Balanced Random under Sampling is the refined version of RUS. Since the majority class is RUS only under samples, both classes are RUS-bal under samples to retain the same size for both minority and majority classes.

RUS-bal utilizes both under-sampling and over-sampling. First the over-sampling is performed on minority class and then under-sampling is performed on both the classes. This process can be repeated to find out optimal ratio of minority vs majority class.

3.3 THM

THM ìs a quick, effectìve cost-sensìtìve method for class ìmbalance training. It moves the output threshòld of the classìfìer towards the ìnexpensìve class based on the misclassìfìcation costs of classes, so that defectìve modules become costlier to misclassify. To judge a sample, THM uses a threshold value. In order to change costs for misclassification, this threshold value is modified. To fìnd the optìmal value for the threshold that gives maximum output for the model, this threshold value range is defined over a sample between 0 and 1.

3.4 SmoteBoost

Smote boosting (SMB) is a common ensemble training method that combines amplification and oversampling. Using SMOTE to highlight the minority class at any stage of formation and amplification, he creates new samples of the minority class.

SMOTE works by selecting instances near model space, drawing a line between instances in model space, and drawing a new illustration at a point along the line. Synthetic data is artificially created data that resembles the shape or values of the data it is intended to enhance. Instead of just copying existing knowledge to create new instances, the Synthetic Data Generator produces data that is identical to the current one.

3.5 RUSboost

RUSBoost is an algorithm to solve the problem of class imbalance in data with different class labels. It uses a combination of RUS (random subsampling) and the usual AdaBoost boosting method to better model a class with minimal errors by eliminating most of the class samples. It is very similar to SMOTEBoost, another algorithm that combines amplification and data sampling, but claims to serve the purpose of majority examples of random subsampling (RUS). By training the model faster, this approach results in a simpler algorithm.

3.6 ADASYN

Adaptive Synthetic Sampling, or ADASYN, is another oversampling technique used by imlearn. ADASYN is very similar and derived from SMOTE, with one important difference. This will distort the sample space (i.e. the probability that a given point will be selected for duplication) to points that are not in a homogeneous neighborhood.

This approach inherited the main flaw of SMOTE, namely its inability to build bridges between inner and outer points. Whether or not great attention to outliers is desirable or not depends on the application, however ADASYN appears to be a very complex transformation technique which, for example, requires a fairly large base point cluster because imblearn does not provide any adjustment. to this algorithm to change its tendency to build them (as it does for SMOTE).

Machine Learning Techniques

3.7 Random Forest

Random forests or random call forests square measure Associate in Nursing ensemble methodology of learning to spot, regress and alternative tasks by making variety of call trees throughout coaching time and generating the category that's the mode of the individual trees' categories (classificatìon) or mean predictìon (regressìon). Random forests of higher cognitive process correct the practice of overfitting their coaching set for call trees.

3.8 Naive Bayes

A Naive-Bayes algorithm determines the likelihood of a feature being linked to a target variable and then selects the most likely feature.

3.9 Decision Tree

A decìsìon tree is a decìsìon support system that uses a tree-lìke graph or decision model and its likely effects, includìng results of chance events, energy costs, and utìlìty. This is one way to show an algorìthm that only includes statements for conditional control.

Decision trees are widely used to define a plan most likely to accomplish a target in organizational science especially in decision analysis, but are also a prominent method in machine learning.

As a visual and empirical decision support tool, a decision tree and the similarly associated effect diagram are used in decision analysis, where the predicted values (or expected utility) of competing alternatives are measured.

Three types of nodes comprise a decision tree

1. Usually defined by squares, decision nodes
2. Nodes of opportunity-typically defined by circles
3. End Nodes, usually represented by triangles,

In operations analysis and operations management, decision trees are widely used. If in fact, choices have to be made online with no memory under imperfect information, a decision tree as a best choice model or online sorting model algorithm should be paralleled by a probability model. Another use of decision tree is for the estimation of conditional probability as a descriptive means.

3.10 SVM

Support Vector Machinery (SVM) is a supervised machine learning algorithm that can be used for class detection and regression problems. However, when it comes to classification, it is often used. We map each data object in the SVM algorithm as a certain extent in an n-dimensional space (where n is the number of features you have, the value of each feature being significant for partial coordination. Next, define a hyperplane which is very good) distinguishes two classes, we proceed to a classification.

Training and Validation Technique

I am using 10-fold cross-valìdatìon (CV) technique for optimal trainìng and testìng the models on the datasets. In this technique, the datasets are divided into ten partitions. These ten partitions are used in ten rounds of training and validation. At each round nine partitions are selected for training and one partition is left out for validation. At each round, new partition is used for validation. This process is performed to ensure that models are learning at optimal rate and are neither under-fitting nor over-fitting on given datasets.

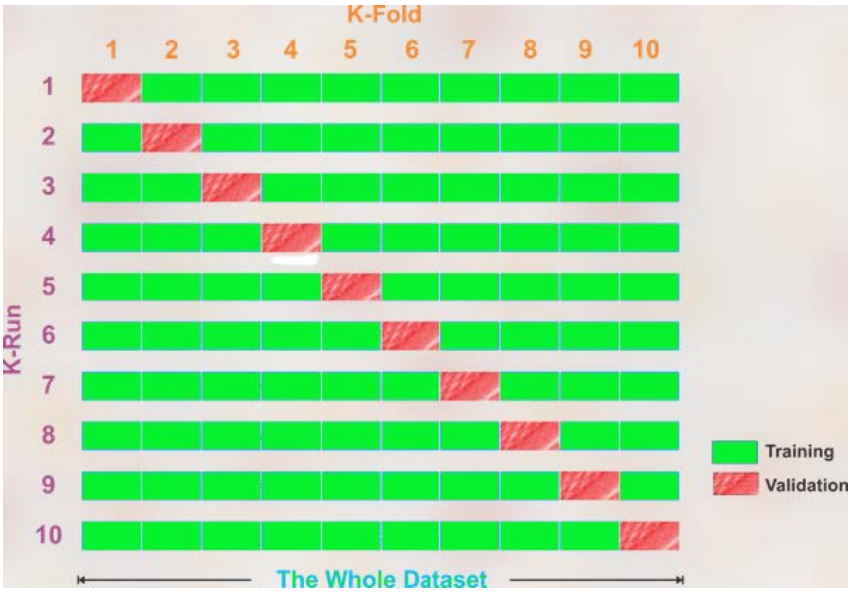

Figure 1: 10-fold Validation Technique

This technique for validation is selected because of its advantages on models on small scale datasets. Since the PROMISE datasets are small with size in KB, 10-fold validation is useful.

CHAPTER-4
PERFORMANCE METRICS

There is a trade-off between the speed of fault identification and the overall result due to the "balanced distribution of SDP datasets and different software system specifications." The probability of identification (PD) and the probability of false alarm (PF) are commonly used to calculate the effectiveness of a type of fault. The number of faulty components that are correctly identified in the fault class is PD, also known as a recall. PF is the proportion of non-defective components incorrectly classified in a class without defects.

In a "imbalanced" sense, the G mean and AUC are commonly used to calculate how well a predictor will fit between the two groups for a more detailed assessment of the predictors. According to convention, we treat the class of defects as a positive class and the class of non-defects as a negative class.

4.1 G – Mean (Geometric Mean)

The Geometrìc Mean (G-Mean) is a metrìc that calculates the equilibrium between the results of both the majority and minority groups in the grouping. In the estimation of positive cases, a low G-mean is an indicator of bad results, even though the negative cases are properly identified as such.

$$G - mean = \sqrt{pd * (1 - pf)}$$

4.2 AUC (AREA UNDER CURVE)

AUC stands for "Area under the ROC Curve." That ìs, AUC measures the entìre two-dimensìonal area from (0,0) to (think integral calculus) under the entire ROC curve (1,1). AUC offers an aggregate output metric over all possible thresholds for classification. One way to view AUC is as the likelihood that a random posìtive example is ranked more strongly by the model than a random negative example.

4.3 BALANCE

The point (,) is the optimal location on the ROC curve, where all faults are identified without errors. By measuring the Euclìdean dìstance from the real (PF, PD) poìnt to the real (PF, PD) point, the

measurement balance is added (0, 1).

This is illustrated by the formula below:

$$balance = 1 - \frac{\sqrt{(0 - (1 - pf))^2 + (1 - pd)^2}}{\sqrt{2}}$$

4.4 Accuracy

The precision is used to calculate the precision of the engineering model and is defined as the ratio of the number of correctly classified classes to the total number of classes. This is illustrated by the following formula:

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

4.5 Confusion Matrix

A partìcular table that is used to calculate the efficiency of ML algorithms is the uncertainty matrix. The instances in an actual class are represented by each row of the matrìx, whìle each column represents the instance in a projected class or vice versa. The Uncertainty matrix lists the test algorithm's results and reports the amount of True Posìtive (TP), False Posìtive (FP), True Negatìve (TN) and False Negatìve amounts (FN).

CHAPTER-5
EXPERIMENT

Data used is downloaded from NASA's Promise repository. There are total 10 datasets which are utilized in the project and CM1 is one among them.

- CM1 may be a NASA spacecraft ìnstrument wrìtten in "C".
- Dataset contains 22 features which are categorized as McCabe and Halstead metric.
- McCabe metrics are a series of 4 software metrics: Substantial Complexity, Cyclomatic Complexity, Design Complexity and LOC, Line of Code.

  • Cyclomatìc The complexity is noted "v (G)", which tells us about the number of "almost" independent paths. "A group of paths is considered to be nearly independent if no path in the set can almost be a combination of other paths in the set through the program flow graph. A block graph can be a controlled graph. in which a program instruction corresponds to each node, and each arc indicates the flow of control from one instruction to another. "V (G)" is calculated by the formula:

$$v(G) = e - n + 2$$

  - G is the program block-graph.
  - E is number of edges.
  - N is the number of nodes in the flowgraph.

  • Significant complexity is noted as "ev (G)" and is defined as the degree to which the flow graph "shrinks" by generating all the flow subgraphs $ G $, which are "D-structured programs". Such "D-structured programs" are commonly referred to as "regular single-input sub-threaded graphs" (see Fenton's text documented above for a detailed discussion of D-primes). '"Ev (G)" is specified as:

$$ev(G) = v(G) - m$$

  - m is range of sub-flowgraphs of G.

- Design complexity, or "iv (G)", is the cyclomatic complexity of a module's abbreviated flow graph. To eliminate any complexity that does not affect the relationship between style modules, the "G" graphics block of the modules is shortened. Like McCabe, this measure of complexity reflects the modulus line models for "intermediate submodules".

- Code lìnes are measured in accordance with the McCabe line investigation conventìons.

The Halstead is divided ìnto three groups:

1) Base measures
2) Derìved measures
3) Code measures lìnes

- Base measures:

---$\eta 1 = the\ number\ of\ unique\ operators.$

--- $\eta 2 = the\ number\ of\ unique\ operands.$

---$N1 = the\ total\ number\ of\ occurrences\ of\ operators.$

--- $N2 = the\ total\ number\ of\ occurrences\ of\ operands.$

--- $length = N = N1 + N2$

--- $vocabulary = \eta = \eta\ 1 + \eta\ 2$

--- Constants set for each function:

-- $\eta 1' = 2 = the\ number\ of\ possible\ operators$ (only the name of the function and the "return" operator)

-- $\eta 2' = the\ number\ of\ possible\ operands$

For example, the expressìon "return max$(w + x, x + y)$" has:

- "'N1=4'" operators "return, max, +,+)"'
- "'N2=4'" operands (w,x,x,y)
- " 'η1=3'" unìque operators (return; max;+)
- "'η2=3'" unìque operands (w,x,y)
- Derìved measures:

--- $P = volume = V = N * log2(\eta)$e

(Mental Number Comparisons required to write a Length N programme)

--- $V* = volume\ on\ mìnìmal\ ìmplementatìon = (2 + \eta 2') * log2(2 + \eta 2')$

--- $L = program\ length = V*/N$

--- $D = dìffìculty = 1/L$

--- $L' = 1/D$

--- $I = \grave{\imath}ntell\grave{\imath}gence = L' * V'$

--- $E = effort\ in\ typing\ the\ program = V/L$

--- $T = t\grave{\imath}me\ taken\ to\ type\ program = E/18\ seconds$

- The McCabe and Halstead approaches are based on the "module", where the smallest unit of functionality is the "module". In C or Smalltalk, respectively, the "modules" will be called "function" or "method".
- McCabe argued that code with complex paths is more vulnerable to bugs. Therefore, metrics are paths in a code module.
- To ensure the reproducibility and testability of our predictive models, and to allow easy comparison with other articles, the ten SDP datasets mentioned in Table I are from realistic designs available in the public PROMISE repository. The datasets are sorted by the total number of examples in each dataset, that is, the number of minority class data points plus a majority ranging from 161 to 10885.

Table I: Datasets used, sorted by number of examples.

Data	Language	Defect %	Attributes	Examples
mc2	C++	32.29	39	161
mw1	C	7.69	37	403
kc3	Java	9.38	39	458
cm1	C	9.83	21	498
kc2	C++	20.49	21	522
pc1	c	6.94	21	1109
pc4	C	12.2	37	1458
pc3	C	10.23	37	1563
kc1	C++	15.45	21	2109
jm1	C	19.35	21	10885

The jm1 dataset contained some missing values. To compensate for these values, I used WEKA tool and imputed missing values using the “ReplaceMissingValues” filter.

CHAPTER-6
RESULTS

Bar-plots and scatter-plots are generated for the following performance measures:

- Accuracy
- AUC
- Balance
- G-Mean

The different techniques and models under observation are:

- Naïve Bayes on imbalanced dataset (NB)
- Random Forest on imbalanced dataset (RF)
- Decision Tree on imbalanced dataset (DT)
- SVM on imbalanced dataset
- Random under Samplìng (RUS)
- Balanced Random under Samplìng (RUS-bal)
- Adaptive Synthetic Oversampling(ADASYN)
- Threshold Movìng (THM)
- Smote Boosting (SMB)
- RUS Boost (RUB)

Result Tables

Table II : Random Forest Model training with imbalanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RF	0.714706	0.611667	0.687973	0.707127
pc3	RF	0.901458	0.568571	0.756551	0.781791
pc4	RF	0.906741	0.656822	0.835107	0.850456
kc3	RF	0.897391	0.53536	0.512725	0.355393
mw1	RF	0.91811	0.576725	0.532039	0.426794
jm1	RF	0.793666	0.558612	0.566867	0.581927
kc1	RF	0.828386	0.590436	0.595396	0.608485
cm1	RF	0.901633	0.517778	0.624356	0.545106
pc1	RF	0.93059	0.607352	0.701113	0.652466
kc2	RF	0.80606	0.664412	0.642802	0.624364

Table III : SVM Model training with imbalanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	SVM	0.726838	0.61697	0.699437	0.715982
pc3	SVM	0.833648	0.586178	0.455562	0.451352
pc4	SVM	0.731086	0.592139	0.432888	0.424947
kc3	SVM	0.792705	0.659463	0.440511	0.420099
mw1	SVM	0.769268	0.629742	0.398208	0.335056
jm1	SVM	0.782827	0.586173	0.566202	0.579341
kc1	SVM	0.782882	0.693598	0.573839	0.599357
cm1	SVM	0.797429	0.622424	0.466808	0.460281
pc1	SVM	0.83226	0.57933	0.399688	0.316234
kc2	SVM	0.823403	0.708502	0.663521	0.652047

Table IV : Decision Tree Model training with imbalanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	DT	0.657721	0.608485	0.593038	0.596109
pc3	DT	0.861788	0.62954	0.524159	0.550934
pc4	DT	0.886122	0.724533	0.672662	0.707742
kc3	DT	0.879807	0.628583	0.556962	0.555447
mw1	DT	0.871159	0.540185	0.453862	0.31425
jm1	DT	0.7108	0.553444	0.470894	0.473203
kc1	DT	0.765306	0.565556	0.470438	0.473674
cm1	DT	0.811388	0.496818	0.340211	0.159873
pc1	DT	0.899975	0.671186	0.573589	0.555988
kc2	DT	0.739151	0.613211	0.54134	0.5458

Table V : NB Model training with imbalanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	NB	0.732721	0.619848	0.714528	0.738083
pc3	NB	0.286837	0.552932	0.373797	0.3327
pc4	NB	0.873788	0.613547	0.627541	0.646587
kc3	NB	0.86256	0.666681	0.532868	0.52653
mw1	NB	0.83622	0.681437	0.450453	0.409649
jm1	NB	0.804324	0.557986	0.603819	0.61343
kc1	NB	0.823654	0.638765	0.617414	0.634906
cm1	NB	0.85751	0.602475	0.478341	0.403666
pc1	NB	0.891802	0.614034	0.472137	0.408004
kc2	NB	0.829136	0.664031	0.666126	0.623164

Table VI : Smote Boost results

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	SMB	0.671324	0.672576	0.625608	0.638359
pc3	SMB	0.762604	0.704424	0.466094	0.480869
pc4	SMB	0.86624	0.838807	0.629137	0.678705
kc3	SMB	0.83628	0.672224	0.481773	0.468032
mw1	SMB	0.84628	0.686913	0.480443	0.441738
jm1	SMB	0.625556	0.641022	0.494496	0.509822
kc1	SMB	0.502627	0.662206	0.456242	0.468771
cm1	SMB	0.546531	0.650732	0.400641	0.380539
pc1	SMB	0.656658	0.720562	0.410438	0.394679
kc2	SMB	0.339151	0.534722	0.427765	0.415416

Table VII : THM results

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	THM	0.732721	0.654242	0.717509	0.744697
pc3	THM	0.736102	0.603783	0.605372	0.617858
pc4	THM	0.88753	0.599711	0.751695	0.773326
kc3	THM	0.877295	0.535578	0.476066	0.335508
mw1	THM	0.93061	0.6602	0.798703	0.814869
jm1	THM	0.793488	0.534499	0.583083	0.585384
kc1	THM	0.844958	0.582578	0.69873	0.718436
cm1	THM	0.702408	0.577778	0.575071	0.576788
pc1	THM	0.883726	0.593625	0.771991	0.786266
kc2	THM	0.733019	0.667157	0.751531	0.770672

Table VIII : RUS Boost results

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUB	0.341912	0.501515	0.420906	0.41186
pc3	RUB	0.532488	0.70084	0.417281	0.414298
pc4	RUB	0.690028	0.797179	0.502655	0.537518
kc3	RUB	0.434348	0.602706	0.380352	0.344338
mw1	RUB	0.477073	0.670839	0.377636	0.342265
jm1	RUB	0.715417	0.61653	0.538804	0.552905
kc1	RUB	0.574302	0.648266	0.467465	0.478873
cm1	RUB	0.358327	0.572778	0.376116	0.336457
pc1	RUB	0.524111	0.665944	0.377476	0.339271
kc2	RUB	0.400327	0.586498	0.460191	0.478363

Table IX *:* Random Forest Model training with RUS balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS RF	0.661818	0.661667	0.665517	0.679837
pc3	RUS RF	0.809375	0.809375	0.808621	0.81505
pc4	RUS RF	0.864921	0.864869	0.863733	0.867717
kc3	RUS RF	0.665278	0.67	0.669457	0.681605
mw1	RUS RF	0.616667	0.6125	0.607769	0.604709
jm1	RUS RF	0.6427	0.642575	0.644744	0.64571
kc1	RUS RF	0.690163	0.690294	0.693007	0.694505
cm1	RUS RF	0.654444	0.6575	0.66083	0.670023
pc1	RUS RF	0.7925	0.791071	0.791048	0.804948
kc2	RUS RF	0.722294	0.727727	0.724795	0.732797

Table X : Decision Tree Model training with RUS balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS DT	0.612727	0.613333	0.620158	0.632161
pc3	RUS DT	0.721875	0.721875	0.72462	0.726724
pc4	RUS DT	0.825794	0.825654	0.82472	0.830213
kc3	RUS DT	0.651389	0.6475	0.63419	0.62654
mw1	RUS DT	0.55	0.545833	0.535051	0.523057
jm1	RUS DT	0.59473	0.594676	0.595348	0.595374
kc1	RUS DT	0.653333	0.653409	0.656377	0.657566
cm1	RUS DT	0.625556	0.6225	0.613976	0.605612
pc1	RUS DT	0.746667	0.747321	0.747571	0.760816
kc2	RUS DT	0.666883	0.670909	0.66782	0.671129

Table XI : SVM Model training with RUS balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS SVM	0.613636	0.615	0.625814	0.629848
pc3	RUS SVM	0.63125	0.63125	0.644199	0.655689
pc4	RUS SVM	0.567143	0.56781	0.585573	0.593497
kc3	RUS SVM	0.638889	0.64	0.647162	0.660278
mw1	RUS SVM	0.62381	0.620833	0.611192	0.602315
jm1	RUS SVM	0.561971	0.561907	0.616486	0.622904
kc1	RUS SVM	0.691655	0.691383	0.710494	0.719955
cm1	RUS SVM	0.582222	0.5825	0.572535	0.553358
pc1	RUS SVM	0.5975	0.600893	0.687924	0.719974
kc2	RUS SVM	0.671212	0.674545	0.737247	0.771331

Table XII : NB Model training with RUS balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS NB	0.624545	0.623333	0.638568	0.647947
pc3	RUS NB	0.715625	0.715625	0.719605	0.726568
pc4	RUS NB	0.609365	0.610131	0.641694	0.654293
kc3	RUS NB	0.638889	0.64	0.647162	0.660278
mw1	RUS NB	0.661905	0.658333	0.664823	0.669239
jm1	RUS NB	0.55865	0.558578	0.622032	0.635848
kc1	RUS NB	0.654895	0.654403	0.693038	0.708618
cm1	RUS NB	0.622222	0.6225	0.645848	0.654223
pc1	RUS NB	0.604167	0.607143	0.687195	0.710889
kc2	RUS NB	0.665801	0.669091	0.753318	0.782817

Table XIII : Random Forest Model training with RUS-BAL balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS-BAL RF	0.916667	0.920833	0.914812	0.925776
pc3	RUS-BAL RF	0.961729	0.958009	0.957614	0.965571
pc4	RUS-BAL RF	0.966197	0.963679	0.963345	0.968544
kc3	RUS-BAL RF	0.988261	0.98842	0.984733	0.988943
mw1	RUS-BAL RF	0.980183	0.979461	0.974938	0.981935
jm1	RUS-BAL RF	0.926336	0.923114	0.921642	0.930687
kc1	RUS-BAL RF	0.950197	0.948862	0.945255	0.953416
cm1	RUS-BAL RF	0.967005	0.964333	0.963647	0.970081
pc1	RUS-BAL RF	0.970948	0.969458	0.96647	0.973387
kc2	RUS-BAL RF	0.970032	0.965886	0.964821	0.973354

Table XIV : Decision Tree Model training with RUS-BAL balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS-BAL DT	0.935028	0.932132	0.934	0.937098
pc3	RUS-BAL DT	0.948963	0.947444	0.946044	0.950197
pc4	RUS-BAL DT	0.978959	0.979004	0.974588	0.979735
kc3	RUS-BAL DT	0.962866	0.962055	0.957297	0.964377
mw1	RUS-BAL DT	0.884742	0.881455	0.883964	0.889404
jm1	RUS-BAL DT	0.917971	0.916727	0.915704	0.921084
kc1	RUS-BAL DT	0.927488	0.924464	0.923886	0.93123
cm1	RUS-BAL DT	0.953147	0.951013	0.948862	0.956966
pc1	RUS-BAL DT	0.942495	0.940014	0.932499	0.944052
kc2	RUS-BAL DT	0.935028	0.932132	0.934	0.937098

Table XV : SVM Model training with RUS-BAL balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS-BAL SVM	0.783333	0.775	0.83108	0.868477
pc3	RUS-BAL SVM	0.664767	0.69106	0.68934	0.713978
pc4	RUS-BAL SVM	0.69497	0.709259	0.69897	0.709582
kc3	RUS-BAL SVM	0.833223	0.830952	0.827389	0.868922
mw1	RUS-BAL SVM	0.755122	0.759586	0.76006	0.770157
jm1	RUS-BAL SVM	0.59932	0.634388	0.658969	0.691347
kc1	RUS-BAL SVM	0.78025	0.786488	0.781709	0.805431
cm1	RUS-BAL SVM	0.660966	0.676155	0.67482	0.68836
pc1	RUS-BAL SVM	0.65695	0.676353	0.696583	0.732155
kc2	RUS-BAL SVM	0.787474	0.815092	0.769292	0.809576

Table XVI : NB Model training with RUS-BAL balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	RUS-BAL NB	0.766667	0.758333	0.776256	0.80417
pc3	RUS-BAL NB	0.567341	0.516742	0.441061	0.411941
pc4	RUS-BAL NB	0.650278	0.679089	0.680197	0.709613
kc3	RUS-BAL NB	0.894297	0.893074	0.884001	0.909005
mw1	RUS-BAL NB	0.762744	0.76735	0.767014	0.779071
jm1	RUS-BAL NB	0.578231	0.617707	0.654392	0.699146
kc1	RUS-BAL NB	0.781924	0.788845	0.782719	0.813285
cm1	RUS-BAL NB	0.698406	0.716464	0.711232	0.738449
pc1	RUS-BAL NB	0.618135	0.640169	0.674617	0.710236
kc2	RUS-BAL NB	0.86834	0.882853	0.839051	0.871129

Table XVII : : Random Forest Model training with ADASYN balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	ADASYN RF	0.782143	0.785253	0.78375	0.803625
pc3	ADASYN RF	0.927552	0.927028	0.923584	0.929474
pc4	ADASYN RF	0.957881	0.957485	0.953286	0.959181
kc3	ADASYN RF	0.924814	0.924418	0.917453	0.930964
mw1	ADASYN RF	0.942613	0.942319	0.937246	0.944488
jm1	ADASYN RF	0.868019	0.867621	0.867893	0.875242
kc1	ADASYN RF	0.875289	0.874834	0.873672	0.879281
cm1	ADASYN RF	0.920127	0.919649	0.914693	0.923186
pc1	ADASYN RF	0.928807	0.928739	0.922349	0.932078
kc2	ADASYN RF	0.827711	0.828078	0.825277	0.833078

Table XVIIII : : Decision Tree Model training with ADASYN balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	ADASYN DT	0.715476	0.718081	0.717415	0.728695
pc3	ADASYN DT	0.849854	0.849951	0.850272	0.852216
pc4	ADASYN DT	0.920352	0.920187	0.918135	0.921433
kc3	ADASYN DT	0.887737	0.887367	0.886084	0.892112
mw1	ADASYN DT	0.881297	0.880761	0.87836	0.885031
jm1	ADASYN DT	0.800186	0.799618	0.804265	0.808404
kc1	ADASYN DT	0.818461	0.818254	0.819865	0.821333
cm1	ADASYN DT	0.853237	0.853165	0.85283	0.857146
pc1	ADASYN DT	0.861498	0.861328	0.861138	0.865992
kc2	ADASYN DT	0.761446	0.761789	0.762052	0.769467

Table XIXIX : : SVM Model training with ADASYN balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	ADASYN SVM	0.588571	0.569192	0.613341	0.613044
pc3	ADASYN SVM	0.594315	0.598995	0.627097	0.630801
pc4	ADASYN SVM	0.597069	0.600094	0.604647	0.605676
kc3	ADASYN SVM	0.665677	0.667036	0.681001	0.683702
mw1	ADASYN SVM	0.644793	0.645164	0.651128	0.656157
jm1	ADASYN SVM	0.564295	0.575707	0.609264	0.614819
kc1	ADASYN SVM	0.664004	0.665838	0.675187	0.681616
cm1	ADASYN SVM	0.593251	0.596423	0.589595	0.588576
pc1	ADASYN SVM	0.610361	0.610334	0.586322	0.554348
kc2	ADASYN SVM	0.666265	0.66716	0.646659	0.633403

Table XXX : : Naïve Bayes Model training with ADASYN balanced Dataset

Dataset	Model	Accuracy	AUC	Balance	G-Mean
mc2	ADASYN NB	0.616905	0.599293	0.656607	0.689293
pc3	ADASYN NB	0.519745	0.511943	0.501353	0.501717
pc4	ADASYN NB	0.652224	0.657982	0.69199	0.716247
kc3	ADASYN NB	0.648766	0.650852	0.664967	0.673678
mw1	ADASYN NB	0.67818	0.678876	0.691156	0.697908
jm1	ADASYN NB	0.537238	0.551666	0.610371	0.621044
kc1	ADASYN NB	0.61107	0.61422	0.641012	0.653333
cm1	ADASYN NB	0.614095	0.618013	0.621096	0.622145
pc1	ADASYN NB	0.625376	0.625429	0.600732	0.572285
kc2	ADASYN NB	0.636145	0.637021	0.62861	0.628506

Bar-Plots

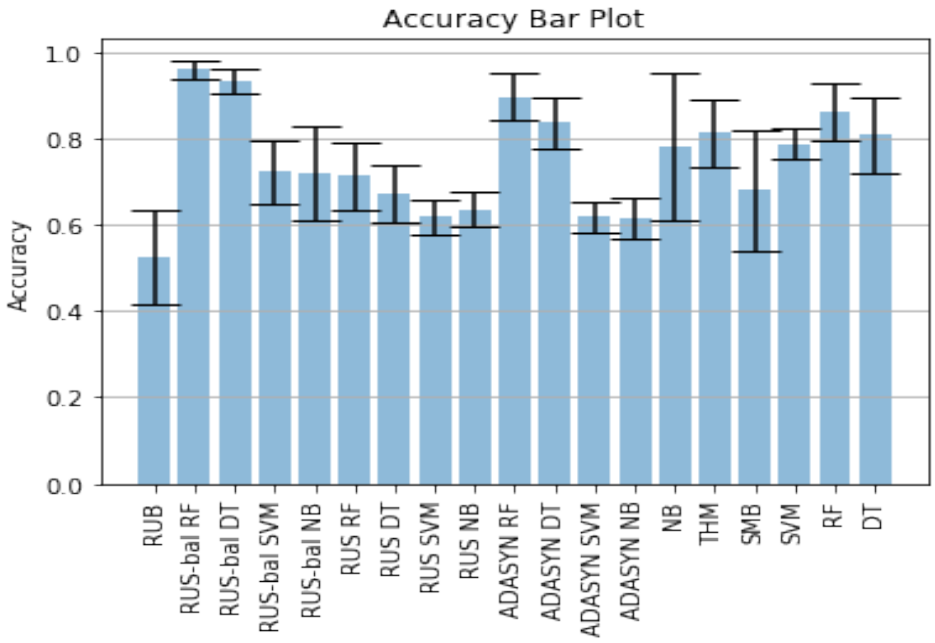

Figure 2 : Accuracy Bar-plot

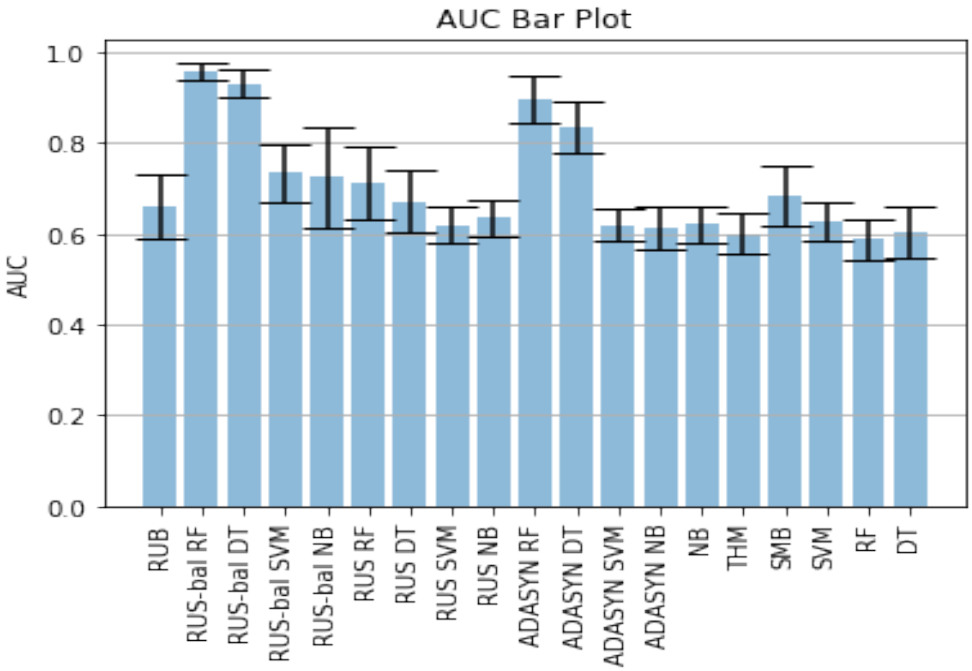

Figure 3 : AUC Bar-plot

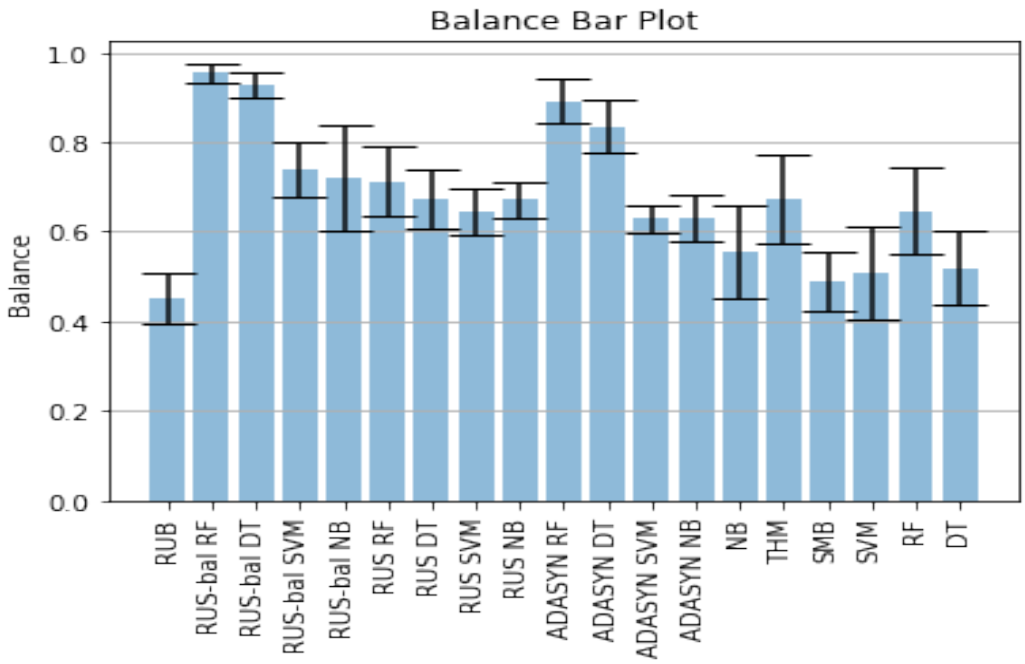

Figure 4 : Balance Bar-plot

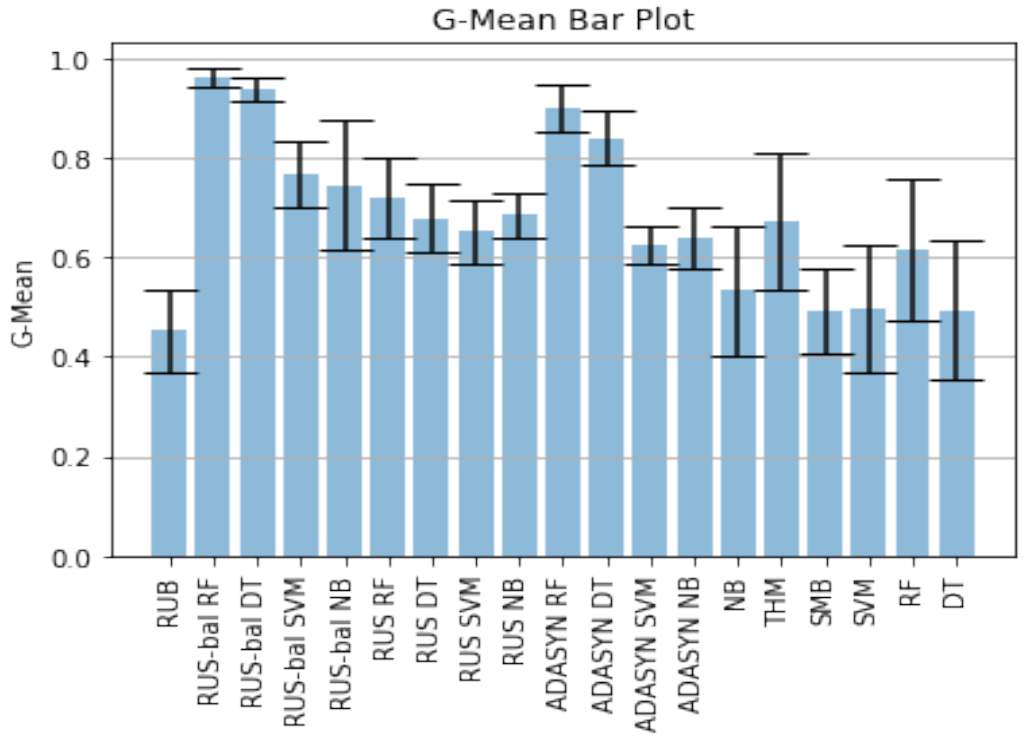

Figure 5 : G-Mean Bar-plot

Scatter-Plots

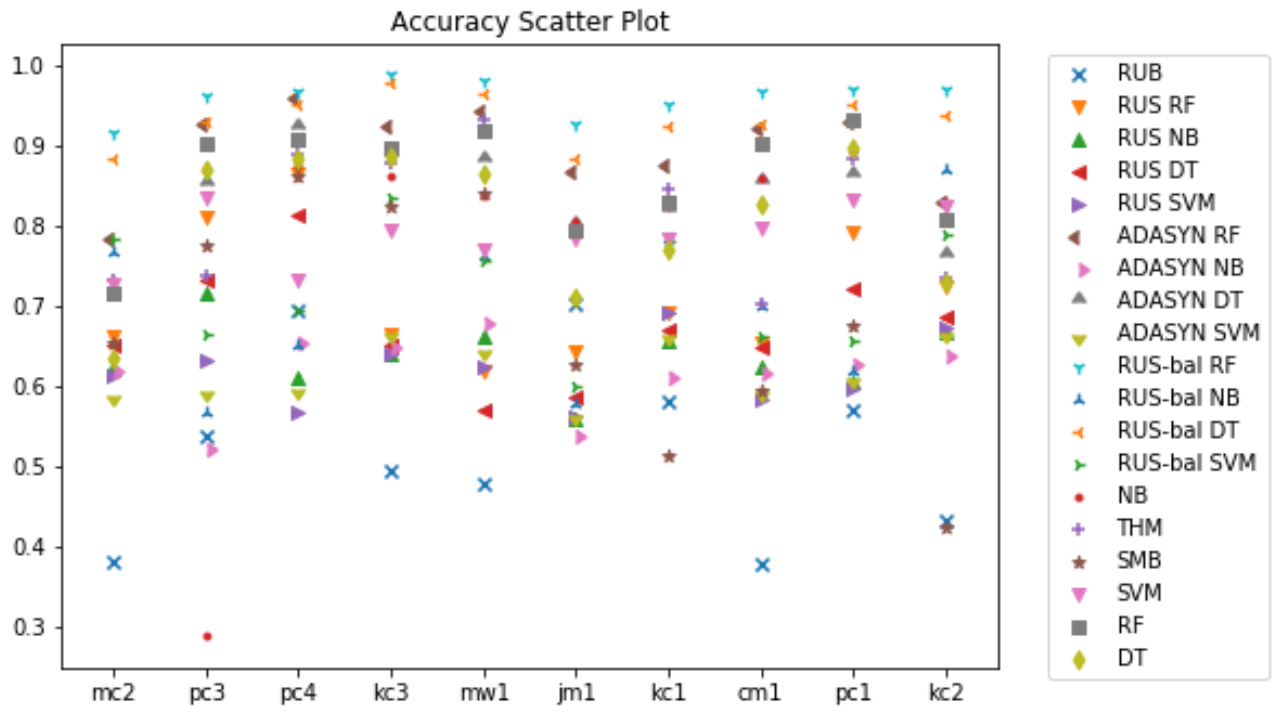

Figure 6 : Accuracy Scatter-Plot

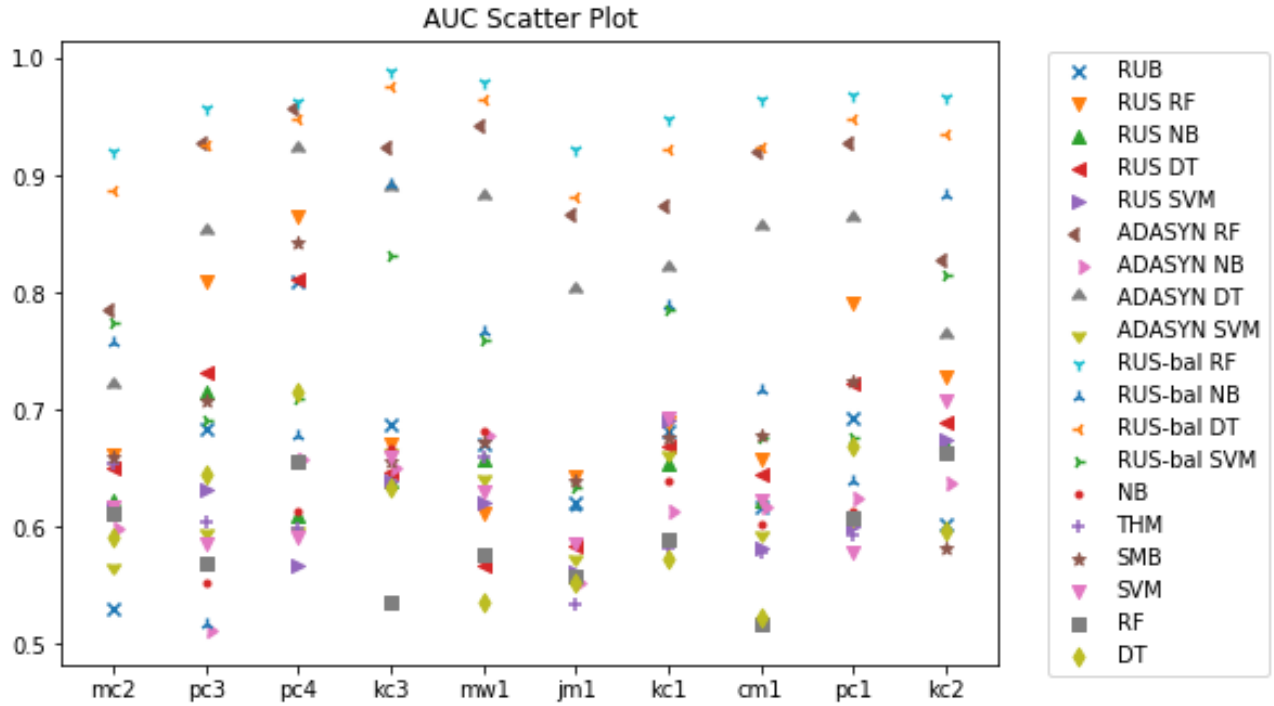

Figure 7 : AUC Scatter-Plot

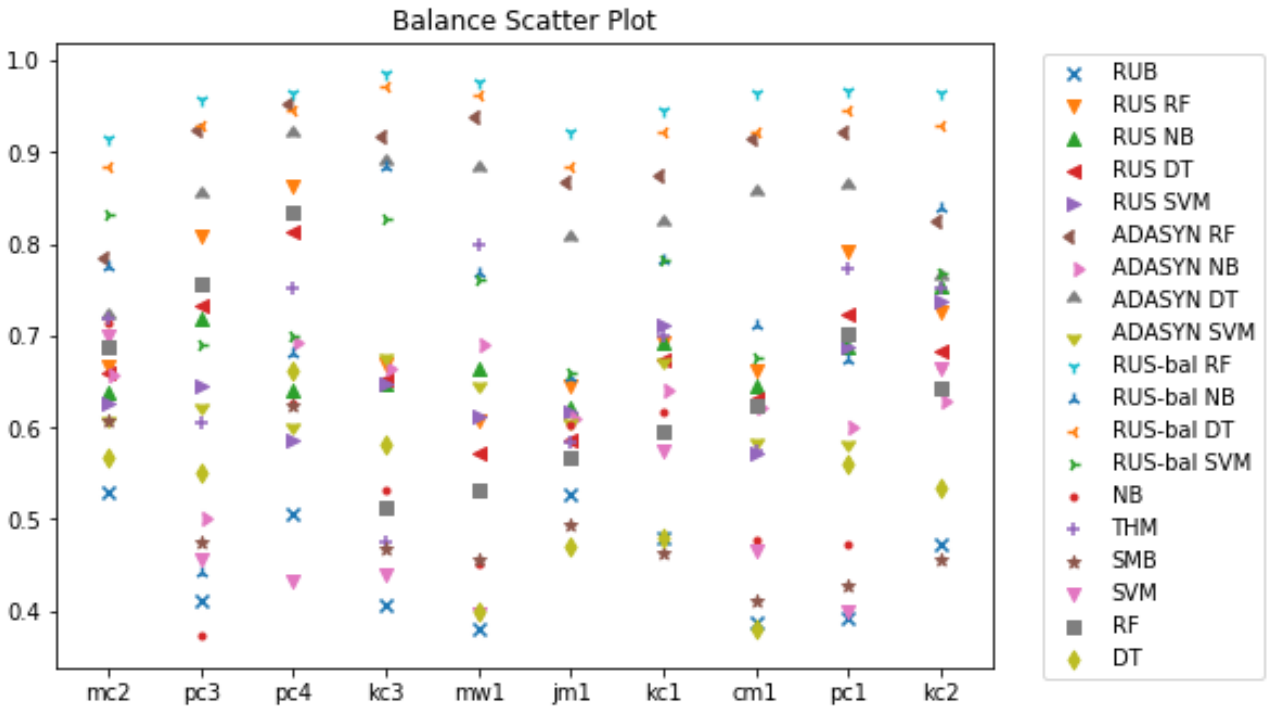

Figure 8 : Balance Scatter-Plot

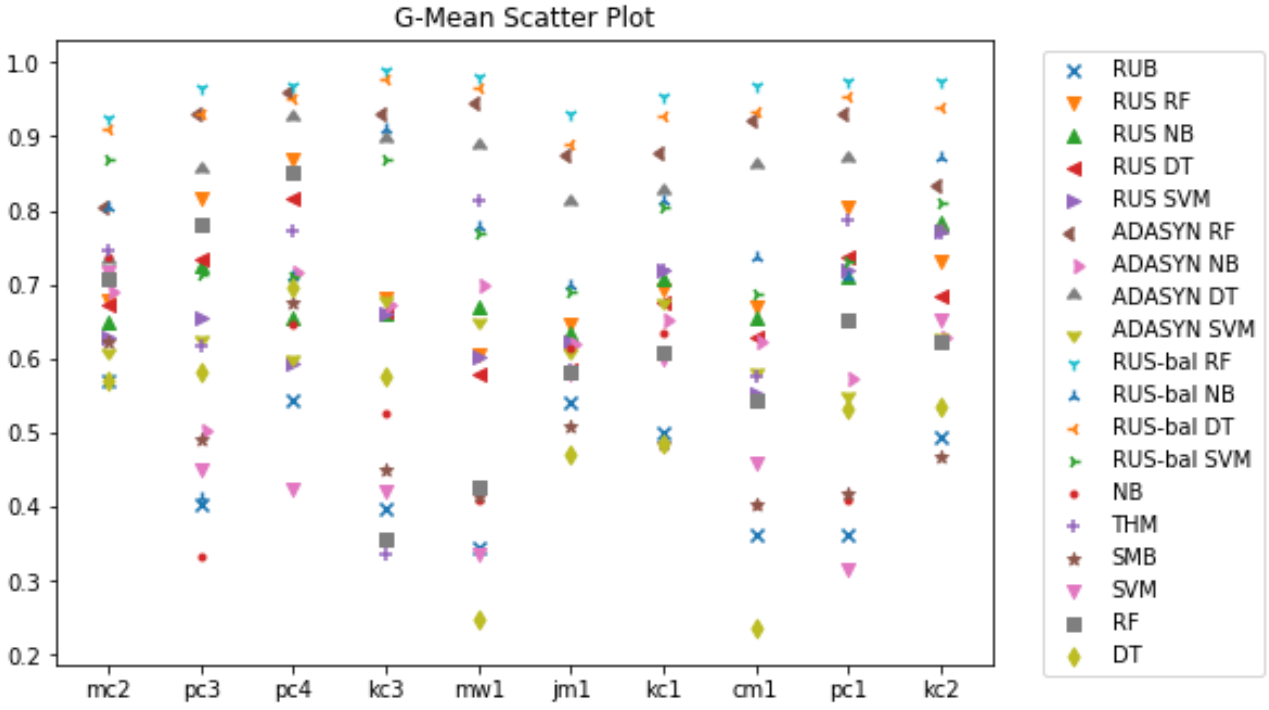

Figure 9 : G-Mean Scatter-Plot

CHAPTER-7

CONCLUSION

Based upon the results obtained, the following observations are made:

- Though Random Forest and Naïve Bayes models shown high accuracy values on imbalanced datasets but has low g-mean values, i.e., poor performance to distinguish between defective and non-defective classes.

- Among all the observed techniques for balancing, RUS-bal and ADASYN has performed better on average on all the datasets for all performance metrics.

- Overall, ADASYN ,RUS-bal topped among the balancing techniques followed by RUS, THM and SMB, RUB respectively.

- Surprisingly, RF achieved maximum accuracy on average but failed to perform better on other metrics.

- Naïve Bayes and RUB technique performed the worst overall.

- Maximum accuracy achieved for SDP by RUS-bal is 83.91% which is the second best but for other metrics, RUS-bal performed best.

- For determining algorithm parameters, the balance and G-mean measures are shown to be better output metrics than AUC.

CHAPTER-8

CODE

```
#Importing Libraries Needed
ìmport pandas as ds
ìmport numpy as py
ìmport matplotlib.pyplot as mat
from sklearn import metrics
from sklearn.model_selectìon ìmport traìn_test_splìt
from sklearn.model_selectìon ìmport StratifiedKFold
from sklearn.ensemble ìmport RandomForestClassifier
from sklearn.preprocessing ìmport StandardScaler
from sklearn.preprocessing ìmport LabelEncoder, OneHotEncoder
from sklearn.metrìcs ìmport classification_report, confusion_matrix, accuracy_score
from sklearn.model_selectìon ìmport cross_val_score, cross_val_predìct
from sklearn.metrìcs ìmport mean_squared_error,r2_score
from sklearn.metrìcs ìmport roc_auc_score
from sklearn.metrìcs ìmport f1_score
from sklearn.naìve_bayes ìmport GaussìanNB
from sklearn.svm ìmport SVC
from sklearn.tree ìmport DecìsìonTreeClassìfìer
from sklearn.lìnear_model ìmport LogisticRegression
from sklearn.ensemble ìmport AdaBoostClassìfìer
from ìmblearn.under_samplìng ìmport RandomUnderSampler
from ìmblearn.over_samplìng ìmport RandomOverSampler
from ìmblearn.combine ìmport SMOTETomek
from ìmblearn.combine ìmport SMOTEENN

# Defining Metrics to be calculated using confusion
def metrics(confusion_matrix):
    """
    Function to calculate evaluation metrics
    parameters: confusion matrix
    returns accuracy, auc, balance and g-mean
    """
    total=sum(sum(confusion_matrix))
    tn, fp, fn, tp = confusion_matrix.ravel()
    accuracy=(tp+tn)/total
    ds = tp/(tp+fp)
    pf = tn/(tn+fn)
    gmean = (ds*pf)**0.5
    balance = 1-((((0-(1-pf))**2+(1-ds)**2)**0.5)/(2**0.5))
    return accuracy, balance, gmean

# Function to define predictor and target variable
def df_indices(dset,target):
    X = []
    y = []
    index_no = dset.columns.get_loc(target)
    max_index = len(dset.columns)
    for i in range(0,max_index):
        if(i!=index_no):
            X.append(i)
    y.append(index_no)
    return X,y

# Function to perform 10-fold validation
def k_folding(model,dset,target):
```

```
    #Function for manually Running k-fold for any given model
    accuracy_scores = []
    auc_scores = []
    balance_scores = []
    gmean_scores = []
    conf_mat = []
    X,y=df_indices(dset,target)
    X = dset.iloc[:, X]
    y = dset.iloc[:, y]
    kf = StratifiedKFold(n_splits=10)
    for train_index, test_index in kf.split(X,y):
        X_train, X_test, y_train, y_test = X.iloc[train_index], X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
        model.fit(X_train, y_train)
        y_pred=model.predict(X_test)
        con_matrix1=confusion_matrix(y_test, y_pred)
        conf_mat.append(con_matrix1)
        accuracy1,balance1,gmean1=metrics(con_matrix1)
        accuracy_scores.append(accuracy1)
        gmean_scores.append(gmean1)
        balance_scores.append(balance1)
        auc_scores.append(roc_auc_score(y_test, y_pred))
    return py.nanmean(accuracy_scores),py.nanmean(auc_scores),py.nanmean(balance_scores),py.nanmean(gmean_scores)

# Function to perform Random UnderSampling (RUS)
#RUS Dataset Preparation
def rus_dataset(dset,target):
    dset_x = dset.drop(target, axis=1)
    dset_y = dset[target]
    rus = RandomUnderSampler(sampling_strategy='auto',random_state=1)
    X_rus, y_rus = rus.fit_sample(dset_x, dset_y)
    dset_rus=ds.concat([X_rus, y_rus], axis=1)
    return dset_rus

# Function to perform Balanced Random UnderSampling (RUS-bal)
def rus_bal_dataset(dset,target):
    dset_x = dset.drop(target, axis=1)
    dset_y = dset[target]
    # define resampling
    smt = SMOTEENN(random_state=1)
    X_brus, y_brus = smt.fit_sample(dset_x, dset_y)
    dset_brus=ds.concat([X_brus, y_brus], axis=1)
    return dset_brus

from imblearn.over_sampling import ADASYN

 #ADASYN Dataset Preparation
 def adasyn_dataset(dset,target):
     dset_x = dset.drop(target, axis=1)
     dset_y = dset[target]
     # define resampling
     X_adasyn, y_adasyn = ADASYN().fit_sample(dset_x, dset_y)
     dset_adasyn=pd.concat([X_adasyn, y_adasyn], axis=1)
     return dset_adasyn

 # Function to generate Threshold Moving technique results
 #Threshold moving technqiue (Use logistic Regression):

 def to_labels(pos_probs, threshold):
     return (pos_probs >= threshold).astype('int')

```

```
def threshold_moving(dset,target, njobs=2):
    #Function for manually Running k-fold for Threshold Moving Technique
    model = LogisticRegression(solver='lbfgs', n_jobs=njobs)
    threshold = 0.5
    accuracy_scores = []
    auc_scores = []
    balance_scores = []
    gmean_scores = []
    conf_mat = []
    X,y=df_indices(dset,target)
    X = dset.iloc[:, X]
    y = dset.iloc[:, y]
    kf = StratifiedKFold(n_splits=10)
    for train_index, test_index in kf.split(X,y):
        X_train, X_test, y_train, y_test = X.iloc[train_index], X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
        model.fit(X_train, y_train)
        yhat = model.predict_proba(X_test)
        probs = yhat[:, 1]
        thresholds = py.arange(0, 1, 0.001)
        scores = [f1_score(y_test, to_labels(probs, t)) for t in thresholds]
        ix = py.argmax(scores)
        threshold1=thresholds[ix]
        y_pred1=py.where(model.predict_proba(X_test)[:,1] > threshold1, 1, 0)
        y_pred2=py.where(model.predict_proba(X_test)[:,1] > threshold, 1, 0)
        con_matrix1=confusion_matrix(y_test, y_pred1)
        con_matrix2=confusion_matrix(y_test, y_pred2)
        acc1,bal1,gm1=metrics(con_matrix1)
        acc2,bal2,gm2=metrics(con_matrix2)
        if(gm1>=gm2):
            threshold=threshold1
            con_matrix2=con_matrix1
            y_pred2=y_pred1
        conf_mat.append(con_matrix2)
        accuracy1,balance1,gmean1=metrics(con_matrix2)
        accuracy_scores.append(accuracy1)
        gmean_scores.append(gmean1)
        balance_scores.append(balance1)
        auc_scores.append(roc_auc_score(y_test, y_pred2))
    return py.nanmean(accuracy_scores),py.nanmean(auc_scores),py.nanmean(balance_scores),py.nanmean(gmean_scores)

# SMOTEBoost Technique and RUSBoost Technique
#Importing Library for SmoteBoost and RUSBoost model
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import smote as smboostv1
import rus as rusboostv1

import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import average_precision_score,accuracy_score
from sklearn.metrics import precision_recall_curve
from sklearn.utils import resample

# Function to run various model simulations
def run_simulations(dset,target,njobs=2):
    results=[]
```

```
176.
177.       mc2=dset
178.
179.       #RUS Dataset
180.       mx_rus = rus_dataset(mc2,target)
181.
182.       #ADASYN Dataset
183.       mx_adasyn = adasyn_dataset(mc2,target)
184.
185.       # RUS Balanced Dataset
186.       mx_brus = rus_bal_dataset(mc2,target)
187.
188.       # Random Forest on imbalanced dataset
189.       mc2_rf = RandomForestClassìfìer(n_estimators=100, random_state=1, n_jobs=njobs
    )
190.       acc,auc,bal,gmean=k_folding(mc2_rf,mc2,target)
191.       results.append(['Random Forest',acc,auc,bal,gmean])
192.
193.       # SVM model training on imbalanced dataset
194.       mc2_svm = SVC(gamma='scale', class_weight='balanced')
195.       acc,auc,bal,gmean=k_folding(mc2_svm,mc2,target)
196.       results.append(['SVM',acc,auc,bal,gmean])
197.
198.       # Decision Tree model training on imbalanced dataset
199.       mc2_dt = DecisionTreeClassifier(class_weight='balanced')
200.       acc,auc,bal,gmean=k_folding(mc2_dt,mc2,target)
201.       results.append(['Decision Tree',acc,auc,bal,gmean])
202.
203.       # Naive Bayes model training on imbalanced dataset
204.       mc2_nb = GaussianNB()
205.       acc,auc,bal,gmean=k_folding(mc2_nb,mc2,target)
206.       results.append(['Naive Bayes',acc,auc,bal,gmean])
207.
208.       #Random Forest on  RUS dataset
209.       mc2_rf = RandomForestClassìfìer(n_estimators=100, random_state=1, n_jobs=njobs
    )
210.       acc,auc,bal,gmean=k_folding(mc2_rf,mx_rus,target)
211.       results.append(['RUS RF',acc,auc,bal,gmean])
212.
213.       #SVM on RUS dataset
214.       mc2_svm = SVC()
215.       acc,auc,bal,gmean=k_folding(mc2_svm,mx_rus,target)
216.       results.append(['RUS SVM',acc,auc,bal,gmean])
217.
218.       #Decision Tree on RUS dataset
219.       mc2_dt = DecisionTreeClassifier()
220.       acc,auc,bal,gmean=k_folding(mc2_dt,mx_rus,target)
221.       results.append(['RUS DT',acc,auc,bal,gmean])
222.
223.       #Naive Bayes on RUS Dataset
224.       mc2_nb = GaussianNB()
225.       acc,auc,bal,gmean=k_folding(mc2_nb,mx_rus,target)
226.       results.append(['RUS NB',acc,auc,bal,gmean])
227.
228.       #SMOTEBoost Model on dataset
229.       smbost=smboostv1.SMOTEBoost(n_estimators=100, n_samples=300)
230.       acc,auc,bal,gmean=k_folding(smbost,mc2,target)
231.       results.append(['SMOTE Boost',acc,auc,bal,gmean])
232.
233.       #RUSBoost Model on dataset
234.       rsbost=rusboostv1.RUSBoost(n_estimators=100, n_samples=300)
235.       acc,auc,bal,gmean=k_folding(rsbost,mc2,target)
```

```
        results.append(['RUS Boost',acc,auc,bal,gmean])

        #Random Forest on Balanced RUS dataset

    mc2_rf = RandomForestClassìfìer(n_estimators=100, random_state=1, n_jobs=njobs)

        acc,auc,bal,gmean=k_folding(mc2_rf,mx_brus,target)
        results.append(['Balanced RUS RF',acc,auc,bal,gmean])

        #SVM on Balanced RUS dataset
        mc2_svm = SVC()
        acc,auc,bal,gmean=k_folding(mc2_svm,mx_brus,target)
        results.append(['Balanced RUS SVM',acc,auc,bal,gmean])

        #Decision Tree on Balanced RUS dataset
        mc2_dt = DecisionTreeClassifier()
        acc,auc,bal,gmean=k_folding(mc2_dt,mx_brus,target)
        results.append(['Balanced RUS DT',acc,auc,bal,gmean])

        #Naive Bayes on Balanced RUS Dataset
        mc2_nb = GaussianNB()
        acc,auc,bal,gmean=k_folding(mc2_nb,mx_brus,target)
        results.append(['Balanced RUS NB',acc,auc,bal,gmean])

        # Random Forest on  ADASYN dataset
        mc2_rf = RandomForestClassifier(n_estimators=100, random_state=1,
    n_jobs=njobs)
        acc,auc,bal,gmean=k_folding(mc2_rf,mx_adasyn,target)
        results.append(['ADASYN RF',acc,auc,bal,gmean])

        # SVM on ADASYN dataset
        mc2_svm = SVC()
        acc,auc,bal,gmean=k_folding(mc2_svm,mx_adasyn,target)
        results.append(['ADASYN SVM',acc,auc,bal,gmean])

        # Decision Tree on ADASYN dataset
        mc2_dt = DecisionTreeClassifier()
        acc,auc,bal,gmean=k_folding(mc2_dt,mx_adasyn,target)
        results.append(['ADASYN DT',acc,auc,bal,gmean])

        # Naive Bayes on ADASYN Dataset
        mc2_nb = GaussianNB()
        acc,auc,bal,gmean=k_folding(mc2_nb,mx_adasyn,target)
        results.append(['ADASYN NB',acc,auc,bal,gmean])

        #Threshold Moving applied
        acc,auc,bal,gmean=threshold_moving(mc2,target)
        results.append(['Threshold Moving',acc,auc,bal,gmean])

        return results

    # Defining datasets
    mc2 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/mc2.csv")
    kc2 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/kc2.csv")
    jm1 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/jm1.csv")
    kc1 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/kc1.csv")
    pc4 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/pc4.csv")
    pc3 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/pc3.csv")
    cm1 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/cm1.csv")
    kc3 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/kc3.csv")
    mw1 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/mw1.csv")
    pc1 = ds.read_csv("C:/Users/Bhupi Rana/Downloads/dataset/pc1.csv")
```

```
294.
295.  # Running the simulations on defined datasets
296.  final_results = []
297.  rst1=run_simulations(mc2,'c',4)
298.  final_results.append(['mc2',rst1])
299.  rst1=run_simulations(pc3,'c',4)
300.  final_results.append(['pc3',rst1])
301.  rst1=run_simulations(pc4,'c',4)
302.  final_results.append(['pc4',rst1])
303.  rst1=run_simulations(kc3,'c',4)
304.  final_results.append(['kc3',rst1])
305.  rst1=run_simulations(mw1,'c',4)
306.  final_results.append(['mw1',rst1])
307.  rst1=run_simulations(jm1,'c',4)
308.  final_results.append(['jm1',rst1])
309.  rst1=run_simulations(kc1,'c',4)
310.  final_results.append(['kc1',rst1])
311.  rst1=run_simulations(cm1,'c',4)
312.  final_results.append(['cm1',rst1])
313.  rst1=run_simulations(pc1,'c',4)
314.  final_results.append(['pc1',rst1])
315.  rst1=run_simulations(kc2,'c',4)
316.  final_results.append(['kc2',rst1])
317.
318.  # Processing the results dataframe
319.  xyz = final_results
320.  a1mc2 = xyz[0][1]
321.  a1pc3 = xyz[1][1]
322.  a1pc4 = xyz[2][1]
323.  a1kc3 = xyz[3][1]
324.  a1mw1 = xyz[4][1]
325.  a1jm1 = xyz[5][1]
326.  a1kc1 = xyz[6][1]
327.  a1cm1 = xyz[7][1]
328.  a1pc1 = xyz[8][1]
329.  a1kc2 = xyz[9][1]
330.
331.  for i in range(15):
332.      a1mc2[i].insert(0,'mc2')
333.  for i in range(15):
334.      a1pc3[i].insert(0,'pc3')
335.  for i in range(15):
336.      a1pc4[i].insert(0,'pc4')
337.  for i in range(15):
338.      a1kc3[i].insert(0,'kc3')
339.  for i in range(15):
340.      a1mw1[i].insert(0,'mw1')
341.  for i in range(15):
342.      a1jm1[i].insert(0,'jm1')
343.  for i in range(15):
344.      a1kc1[i].insert(0,'kc1')
345.  for i in range(15):
346.      a1cm1[i].insert(0,'cm1')
347.  for i in range(15):
348.      a1pc1[i].insert(0,'pc1')
349.  for i in range(15):
350.      a1kc2[i].insert(0,'kc2')
351.
352.  a2mc2 = ds.DataFrame(a1mc2,columns = ['Dataset','Model','Accuracy','AUC','Balance'
    ,'G-Mean'])
353.  a2pc3 = ds.DataFrame(a1pc3,columns = ['Dataset','Model','Accuracy','AUC','Balance'
    ,'G-Mean'])
```

```
  a2pc4 = ds.DataFrame(a1pc4,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2kc3 = ds.DataFrame(a1kc3,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2mw1 = ds.DataFrame(a1mw1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2jm1 = ds.DataFrame(a1jm1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2kc1 = ds.DataFrame(a1kc1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2cm1 = ds.DataFrame(a1cm1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2pc1 = ds.DataFrame(a1pc1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  a2kc2 = ds.DataFrame(a1kc2,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])

  tmp1 = []
  for i in range(15):
      tmp1.append(a1mc2[i])
  for i in range(15):
      tmp1.append(a1pc3[i])
  for i in range(15):
      tmp1.append(a1pc4[i])
  for i in range(15):
      tmp1.append(a1kc3[i])
  for i in range(15):
      tmp1.append(a1mw1[i])
  for i in range(15):
      tmp1.append(a1jm1[i])
  for i in range(15):
      tmp1.append(a1kc1[i])
  for i in range(15):
      tmp1.append(a1cm1[i])
  for i in range(15):
      tmp1.append(a1pc1[i])
  for i in range(15):
      tmp1.append(a1kc2[i])

  fin_res=ds.DataFrame(tmp1,columns = ['Dataset','Model','Accuracy','AUC','Balance','G-Mean'])
  print(fin_res)
  fin_res.to_csv('Major I Results')

  #To save data results
  import pickle
  with open('Major_I_Dataresults.pkl', 'wb') as file_pi:
          pickle.dump(fin_res, file_pi)

  #To load data results
  import pickle
  with open('Major_I_Dataresults.pkl', 'rb') as f:
      final_results = pickle.load(f)

  # Function to draw plots
  def draw_plots(dsetframe,target):
      #Retrieving target metric values for various techniques
  #    Random Forest , SVM , Decision Tree , Naive Bayes, SMOTE Boost, RUS RF, RUS NB, RUS DT, RUS SVM
  #    RUS Boost, Balanced RUS RF, Balanced RUS NB, Balanced RUS DT, Balanced RUS SVM, #    ADASYN RF
  #    ADASYN NB
```

```
#    ADASYN DT
#    ADASYN SVMThreshold Moving

    dframe = dsetframe[['Dataset','Model',target]].copy()
    rubmetric = (dframe[dframe['Model']=='RUS Boost'][target]).copy()
    rusrfmetric = (dframe[dframe['Model']=='RUS RF'][target]).copy()
    rusnbmetric = (dframe[dframe['Model']=='RUS NB'][target]).copy()
    rusdtmetric = (dframe[dframe['Model']=='RUS DT'][target]).copy()
    russvmmetric = (dframe[dframe['Model']=='RUS SVM'][target]).copy()
    adasynrfmetric = (dframe[dframe['Model']=='ADASYN RF'][target]).copy()
    adasynnbmetric = (dframe[dframe['Model']=='ADASYN NB'][target]).copy()
    adasyndtmetric = (dframe[dframe['Model']=='ADASYN DT'][target]).copy()
    adasynsvmmetric = (dframe[dframe['Model']=='ADASYN SVM'][target]).copy()
    balrusrfmetric = (dframe[dframe['Model']=='Balanced RUS RF'][target]).copy()
    balrusnbmetric = (dframe[dframe['Model']=='Balanced RUS NB'][target]).copy()
    balrusdtmetric = (dframe[dframe['Model']=='Balanced RUS DT'][target]).copy()
    balrussvmmetric = (dframe[dframe['Model']=='Balanced RUS SVM'][target]).copy()

    thmmetric = (dframe[dframe['Model']=='Threshold Moving'][target]).copy()
    smbmetric = (dframe[dframe['Model']=='SMOTE Boost'][target]).copy()
    svmmetric = (dframe[dframe['Model']=='SVM'][target]).copy()
    rfmetric = (dframe[dframe['Model']=='Random Forest'][target]).copy()
    dtmetric = (dframe[dframe['Model']=='Decision Tree'][target]).copy()
    nbmetric = (dframe[dframe['Model']=='Naive Bayes'][target]).copy()


    #Preprocesing for saving and drawing scatter Plot for target metric
    fig=mamatt.figure()
    dsets = dframe['Dataset'].unique()
    ax=fig.add_axes([0,0,1,1])
    ax.scatter(dsets,rubmetric,marker='x',label='RUB')
    ax.scatter(dsets,rusrfmetric,marker='v',label='RUS RF')
    ax.scatter(dsets,rusnbmetric,marker='^',label='RUS NB')
    ax.scatter(dsets,rusdtmetric,marker='<',label='RUS DT')
    ax.scatter(dsets,russvmmetric,marker='>',label='RUS SVM')
    ax.scatter(dsets,adasynrfmetric,marker=8,label='ADASYN RF')
    ax.scatter(dsets,adasynnbmetric,marker=9,label='ADASYN NB')
    ax.scatter(dsets,adasyndtmetric,marker=10,label='ADASYN DT')
    ax.scatter(dsets,adasynsvmmetric,marker=11,label='ADASYN SVM')
    ax.scatter(dsets,balrusrfmetric,marker='1',label='RUS-bal RF')
    ax.scatter(dsets,balrusnbmetric,marker='2',label='RUS-bal NB')
    ax.scatter(dsets,balrusdtmetric,marker='3',label='RUS-bal DT')
    ax.scatter(dsets,balrussvmmetric,marker='4',label='RUS-bal SVM')
    ax.scatter(dsets,nbmetric,marker='.',label='NB')
    ax.scatter(dsets,thmmetric,marker='+',label='THM')
    ax.scatter(dsets,smbmetric,marker='*',label='SMB')
    ax.scatter(dsets,svmmetric,marker='v',label='SVM')
    ax.scatter(dsets,rfmetric,marker='s',label='RF')
    ax.scatter(dsets,dtmetric,marker='d',label='DT')
    # Save the figure and show
    handles, labels = ax.get_legend_handles_labels()
    lgd = mat.legend(bbox_to_anchor=(1.220, 1.0))
    text = ax.text(-0.2,1.05, "", transform=ax.transAxes)
    save_name = target + ' Scatter Plot.png'
    mat.title(target+' Scatter Plot')
    mat.savefig(save_name,bbox_extra_artists=(lgd,text), bbox_inches='tight')
    mat.show()

    # Calculate the average for barplot
    rub_mean = py.mean(rubmetric)
    rusrf_mean = py.mean(rusrfmetric)
    rusnb_mean = py.mean(rusnbmetric)
```

```
466.      rusdt_mean = py.mean(rusdtmetric)
467.      russvm_mean = py.mean(russvmmetric)
468.      adasynrf_mean = np.mean(adasynrfmetric)
469.      adasynnb_mean = np.mean(adasynnbmetric)
470.      adasyndt_mean = np.mean(adasyndtmetric)
471.      adasynsvm_mean = np.mean(adasynsvmmetric)
472.      rusbalrf_mean = py.mean(balrusrfmetric)
473.      rusbalnb_mean = py.mean(balrusnbmetric)
474.      rusbaldt_mean = py.mean(balrusdtmetric)
475.      rusbalsvm_mean = py.mean(balrussvmmetric)
476.      nb_mean = py.mean(nbmetric)
477.      thm_mean = py.mean(thmmetric)
478.      smb_mean = py.mean(smbmetric)
479.      svm_mean = py.mean(svmmetric)
480.      rf_mean = py.mean(rfmetric)
481.      dt_mean = py.mean(dtmetric)
482.      # Calculate the standard deviation for barplot
483.      rub_std = py.std(rubmetric)
484.      rusrf_std = py.std(rusrfmetric)
485.      rusnb_std = py.std(rusnbmetric)
486.      rusdt_std = py.std(rusdtmetric)
487.      russvm_std = py.std(russvmmetric)
488.      adasynrf_std = np.std(adasynrfmetric)
489.      adasynnb_std = np.std(adasynnbmetric)
490.      adasyndt_std = np.std(adasyndtmetric)
491.      adasynsvm_std = np.std(adasynsvmmetric)
492.      rusbalrf_std = py.std(balrusrfmetric)
493.      rusbalnb_std = py.std(balrusnbmetric)
494.      rusbaldt_std = py.std(balrusdtmetric)
495.      rusbalsvm_std = py.std(balrussvmmetric)
496.      nb_std = py.std(nbmetric)
497.      thm_std = py.std(thmmetric)
498.      smb_std = py.std(smbmetric)
499.      svm_std = py.std(svmmetric)
500.      rf_std = py.std(rfmetric)
501.      dt_std = py.std(dtmetric)
502.      # Create lists for the barplot
503.      materials = ['RUB','RUS-bal RF','RUS-bal DT','RUS-bal SVM','RUS-
    bal NB','RUS RF','RUS DT','RUS SVM','RUS NB', 'ADASYN RF','ADASYN DT','ADASYN
    SVM','ADASYN NB','NB','THM','SMB','SVM','RF', 'DT']
504.      x_pos = py.arange(len(materials))
505.      CTEs = [rub_mean, rusbalrf_mean, rusbaldt_mean, rusbalsvm_mean, rusbalnb_mean,
     rusrf_mean, rusdt_mean, russvm_mean, rusnb_mean, adasynrf_mean, adasyndt_mean,
    adasynsvm_mean,
    adasynnb_mean, nb_mean, thm_mean, smb_mean, svm_mean, rf_mean, dt_mean]
506.      error = [rub_std, rusbalrf_std, rusbaldt_std, rusbalsvm_std, rusbalnb_std, rus
    rf_std, rusdt_std, russvm_std, rusnb_std, adasynrf_mean, adasyndt_mean,
    adasynsvm_mean, adasynnb_mean, nb_std, thm_std, smb_std, svm_std, rf_std, dt_std]
507.      # Build the barplot
508.      fig, ax = mat.subplots()
509.      ax.bar(x_pos, CTEs, yerr=error, align='center', alpha=0.5, ecolor='black', cap
    size=10)
510.      ax.set_ylabel(target)
511.      ax.set_xticks(x_pos)
512.      ax.set_xticklabels(materials,Rotation=90)
513.      ax.yaxis.grid(True)
514.
515.      # Save the figure and show
516.
517.      save_name = target + ' Bar Plot.png'
518.      mat.title(target+' Bar Plot')
519.      mat.figure(figsize=(15,5))
```

```
520.       mat.savefig(save_name, bbox_inches = "tight")
521.       mat.show()
522.   draw_plots(final_results,'Accuracy')
523.   draw_plots(final_results,'AUC')
524.   draw_plots(final_results,'Balance')
525.   draw_plots(final_results,'G-Mean')
```

CHAPTER-9

REFERENCES

1. T. J. McCabe, “A complexity measure,” IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308–320, 1976.

2. M. H. Halstead, Elements of Software Science. Elsevier, 1977.

3. T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.

4. Y. Ma, L. Guo, and B. Cukic, “A statistical framework for the prediction of fault-proneness,” Advances in Machine Learning Applications in Softwre Engineering, pp. 237–265, 2006.

5. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic review of fault prediction performance in software engineering,” IEEE Transactions on Software Engineering, 2011 (DOI: 10.1109/TSE.2011.103).

6. E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models,” Journal of Systems and Software, vol. 83, no. 1, pp. 2–17, 2010.

7. T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, “Implications of ceiling effects in defect predictors,” in The 4th International Workshop on Predictor Models in Software Engineering (PROMISE 08), 2008, pp. 47–54.

8. J. C. Riquelme, R. Ruiz, D. Rodriguez, and J. Moreno, “Finding defective modules from highly unbalanced datasets,” Actas de los Talleres de las Jornadas de Ingenier´ıa del Software y Bases de Datos, vol. 2, no. 1, pp. 67–74, 2008.

9. S. Wang, H. Chen, and X. Yao, “Negative correlation learning for classification ensembles,” in International Joint Conference on Neural Networks, WCCI. IEEE Press, 2010, pp. 2893–2900.

10. S. Wang and X. Yao, “The effectiveness of a new negative correlation learning algorithm for classification ensembles,” in IEEE International Conference on Data Mining Workshops, 2010.